- Оглавление
- Начало работы
- Концепции
  - Глоссарий
  - Аналитические возможности
    - Обзор
    - Детально
      - Ввод данных
      - Хранение данных
      - Выполнение запросов
      - Федеративные запросы
      - Трансформация и подготовка данных (ELT/ETL)
      - BI-аналитика и визуализация данных
      - Machine Learning
    - Основные функции
  - Потоковая обработка
    - Потоковые запросы
    - Watermarks
  - Архитектура
    - Обзор архитектуры
    - Сервисы распространения метаданных
  - Соединение с БД
  - Схемные объекты
    - Пространство имён
    - Директории
    - Таблицы
    - Представления
    - Топики
    - Узлы координации
    - Секреты
    - Внешние источники данных
    - Внешние таблицы
    - Служебные объекты
      - Коллекции резервных копий
  - Топология кластера
  - Выполнение запросов
    - Обзор
    - Процесс выполнения запросов
    - Оптимизатор запросов
    - Многоверсионное управление конкурентным доступом (MVCC)
    - Вторичные индексы
    - Векторный поиск
    - Полнотекстовый поиск
    - Гибридный поиск
    - Поиск по JSON
    - Локальные индексы
    - Федеративные запросы
      - Обзор
      - Архитектура
      - Работа с базами данных ClickHouse
      - Работа с базами данных Greenplum
      - Работа с базами данных Microsoft SQL Server
      - Работа с базами данных MySQL
      - Работа с базами данных PostgreSQL
      - Работа с базами данных YDB
      - Работа с бакетами S3
        - Чтение из бакетов S3 через внешние источники данных
        - Чтение из бакетов S3 через внешние таблицы
        - Запись данных в бакеты S3
        - Форматы данных и алгоритмы сжатия
        - Отображение типов при чтении и записи в формате parquet
        - Партицирование данных
        - Расширенное партицирование
      - Импорт и экспорт данных
    - Спиллинг
    - Скан запросы
    - YQL запросы к топикам
  - Транзакции
  - Вторичные индексы
  - Change Data Capture (CDC)
  - Time to Live (TTL) и вытеснение данных во внешнее хранилище

    - Ограничения базы данных
    - Резервное копирование и восстановление
    - Асинхронная репликация
    - Режим bridge
    - Трансфер данных
- Администрирование кластеров
    - Концепции
        - Системные требования
        - Настройки CPU для production
        - Планирование ёмкости хранилища
        - Версионирование
        - Обслуживание
        - SelfHeal
            - SelfHeal хранилища
            - SelfHeal подсистем распространения метаданных
        - Аутентификация и авторизация узлов баз данных
    - Управление конфигурацией кластера
        - Конфигурация V1
            - Обзор конфигурации
            - Статическая конфигурация
            - Динамическая конфигурация
            - Временная конфигурация
            - DSL конфигурации кластера
            - Изменение конфигураций через CMS
            - Изменение конфигурации актор-системы
            - Расширение кластера
            - Конфигурирование State Storage
            - Перемещение State Storage
            - Перемещение статической группы
            - Замена FQDN узла
            - Аутентификация и авторизация узлов баз данных
        - Конфигурация V2
            - Обзор конфигурации
            - Обновление конфигурации
            - DSL конфигурации кластера
            - Параметры конфигурации
            - Расширение кластера
            - Удаление узла из кластера
            - Конфигурирование State Storage
            - Перемещение State Storage
            - SelfHeal статической группы
            - Перемещение статической группы
            - Замена FQDN узла
            - Аутентификация и авторизация узлов баз данных
        - Миграции
            - На конфигурацию V2
            - На конфигурацию V1
        - Проверка версии конфигурации
        - Сравнение конфигураций V1 и V2
    - Способы развёртывания
        - Ansible
            - Первоначальное развёртывание
                - Подготовка к развёртыванию
                - Развёртывание кластера с использованием конфигурации V1
                - Развёртывание кластера с использованием конфигурации V2
            - Подготовка виртуальных машин с Terraform
            - Перезапуск
            - Обновление конфигурации
            - Обновление исполняемого файла
            - Наблюдаемость
                - Логирование
        - Kubernetes
            - Первоначальное развёртывание
        - Вручную
            - Первоначальное развёртывание
                - Подготовка к развёртыванию
                - Развёртывание кластера с использованием конфигурации V1
                - Развёртывание кластера с использованием конфигурации V2
            - Обновление конфигурации
            - Обслуживание кластера
            - Обновление исполняемого файла
            - Управление кластером в режиме bridge
            - Управление дисковой подсистемой кластера

        - Обзор
        - Добавление групп хранения
        - Безопасный рестарт и выключение узлов
        - Включение и выключение Scrubbing
        - Декомиссия ЦОД
        - Декомиссия групп с использованием виртуальных групп
        - Перевоз VDisk'ов
        - Предотвращение выхода за модель отказа
        - Балансировка нагрузки на диски
        - Освобождение места на физических устройствах
        - BlobDepot
        - Декомиссия BlobDepot
      - Федеративные запросы
        - Развёртывание коннектора
  - Enterprise Manager
    - Первоначальное развёртывание
    - Настройка SSO
    - Настройка резервного копирования в S3
    - Настройка AI-ассистента
  - Наблюдаемость
    - Мониторинг
    - Логирование
    - Минидампы
    - Системные представления
    - Трассировка
    - Алертинг
    - Диагностирование
  - Резервное копирование и восстановление
  - Первоначальная синхронизация асинхронной репликации
- Для разработчиков приложений
  - Начало работы
  - Потоковые запросы
    - Типичные шаблоны
    - Запись в таблицы
    - Обогащение данных
    - Форматы данных при чтении/записи топиков
    - Гарантии доставки данных
    - Чекпоинты
    - Watermarks
  - Примеры приложений
    - C++
    - C#
    - Go
    - Java
    - JavaScript
    - PHP
    - Python
    - Rust
  - Генерация кода из YQL
  - Туториал по YQL
    - Создание таблицы
    - Добавление данных в таблицу
    - Выборка данных из всех колонок
    - Выборка данных из определенных колонок
    - Сортировка и фильтрация
    - Агрегирование данных
    - Дополнительные условия выборки
    - Объединение таблиц с помощью JOIN
    - Вставка и модификация данных с помощью REPLACE
    - Вставка и модификация данных с помощью UPSERT
    - Вставка данных с помощью INSERT
    - Обновление данных с помощью UPDATE
    - Удаление данных
    - Добавление и удаление колонок
    - Удаление таблицы
  - Первичный ключ
    - Строковые
    - Колоночные
  - Партицирование
    - Строковые
      - Как работает партицирование
      - Автоматическое партицирование и ограничения
      - Как выбрать количество партиций

        - Анти-паттерны
      - Колоночные
    - Вторичные индексы
    - Векторные индексы
    - Векторный индекс вида vector_kmeans_tree
    - Полнотекстовые индексы
    - Гибридный поиск
    - JSON-индексы
    - Локальные индексы
      - Блум-индексы
      - min_max-индекс
    - Оптимизация запросов
      - План выполнения запроса
      - Графический план запроса
        - Расположение информации в плане запроса
        - Структура фактического плана запроса
        - Визуализация метрик запроса
      - Подсказки оптимизатора
      - Параметризованные запросы и повторная компиляция
    - Пакетная загрузка
    - Постраничный вывод
    - Таймауты
    - Системные представления
    - Change Data Capture
    - Устройство и ограничения разделяемого читателя топика
    - Terraform
    - Пользовательские атрибуты
    - Управление потреблением ресурсов
- Для аналитиков и инженеров данных
    - Быстрый старт
    - Импорт датасетов
      - Обзор
      - Chess Position Evaluations
      - Video Game Sales
      - E-Commerce Behavior Data
      - COVID-19 Open Research Dataset
      - Netflix Movies and TV Shows
      - Animal Crossing New Horizons Catalog
    - Практические руководства
      - Историзация данных (SCD)
        - Обзор
        - SCD1 с использованием TRANSFER
        - SCD2 с использованием TRANSFER
        - SCD2 с использованием подмерживания изменений
    - Ограничения
- Для инженеров безопасности
    - Аутентификация
      - Кеширование результатов аутентификации
    - Авторизация
    - Начальная настройка безопасности
    - Аудитный лог
    - Шифрование
      - При хранении
      - При передаче
    - Краткая запись управления доступом
- Для контрибьюторов
    - Работа над изменениями
    - Сборка и тесты Ya
    - Логирование
    - Релизы
    - Документация
      - Процесс ревью
      - Руководство по стилю
      - Структура
      - Жанры
      - Добавление медиа в публичные материалы
      - Документирование новых команд CLI
      - Функциональность под фича-флагами
    - Ядро
      - Общая схема
      - Акторная система
        - Жизненный цикл актора
        - События и обмен сообщениями

        - Сбои и завершение работы
        - Акторы на корутинах
        - Примитивы корутин
        - Интеграция корутин
        - Кеш кадров корутин
        - Тестирование акторов
      - Распределённое хранилище
        - NodeWarden
        - Группы прямого блочного доступа
        - DDisk
        - PersistentBuffer
        - Отчёт об использовании места VDisk
      - Подсистемы распространения метаданных
      - LocalDB
        - Персистентные незакомиченные изменения
      - Hive
        - Запуск таблеток
      - DataShard
        - Локи и видимость изменений в транзакциях
        - Распределённые транзакции
      - Конфигурация V2
      - Режим bridge
      - Spilling Service
      - Тестирование с помощью нагружающих акторов
        - KqpLoad
        - KeyValueLoad
        - StorageLoad
        - NbsDbgLikeLoad
        - VDiskLoad
        - PDiskWriteLoad
        - PDiskReadLoad
        - PDiskLogLoad
        - MemoryLoad
        - Stop
    - Тестирование
      - Хаос-тестирование
- Справка
  - YQL
    - Обзор
    - Типы данных
      - Обзор
      - Простые
      - Серийные
      - Опциональные
      - Контейнеры
      - Специальные
      - Преобразования типов
      - Текстовое представление типов данных
      - Передача параметров в формате JSON
    - Синтаксис
      - Обзор
      - Лексическая структура
      - Выражения
      - Ограничения
        - Неподдерживаемые конструкции
        - Коррелированные подзапросы
      - ACTION
      - ALTER ASYNC REPLICATION
      - ALTER DATABASE
        - Обзор
        - OWNER TO
        - SET
      - ALTER GROUP
      - ALTER TABLE
        - Обзор
        - INDEX
        - COLUMN
          - FAMILY
          - NOT NULL
        - SET
        - CHANGEFEED
        - RENAME
        - COMPACT

- ALTER VIEW
- ALTER RESOURCE POOL
- ALTER RESOURCE POOL CLASSIFIER
- ALTER STREAMING QUERY
- ALTER TOPIC
- ALTER TRANSFER
- ALTER SECRET
- ALTER USER
- ANALYZE
- BATCH DELETE
- BATCH UPDATE
- BACKUP
- CREATE ASYNC REPLICATION
- CREATE BACKUP COLLECTION
- CREATE GROUP
- CREATE EXTERNAL DATA SOURCE
- CREATE STREAMING QUERY
- CREATE EXTERNAL TABLE
- CREATE OBJECT TYPE SECRET
- CREATE OBJECT TYPE SECRET_ACCESS
- CREATE SECRET
- CREATE TABLE
  - Обзор
  - SECONDARY INDEX
  - VECTOR INDEX
  - FULLTEXT INDEX
  - JSON INDEX
  - BLOOM INDEX
  - min_max INDEX
  - FAMILY
  - WITH
  - AS SELECT
- CREATE TRANSFER
- CREATE VIEW
- CREATE RESOURCE POOL
- CREATE RESOURCE POOL CLASSIFIER
- CREATE TOPIC
- CREATE USER
- COMMIT
- DECLARE
- DELETE
- DISCARD
- DROP ASYNC REPLICATION
- DROP BACKUP COLLECTION
- DROP GROUP
- DROP EXTERNAL DATA SOURCE
- DROP EXTERNAL TABLE
- DROP OBJECT TYPE SECRET
- DROP OBJECT TYPE SECRET_ACCESS
- DROP STREAMING QUERY
- DROP SECRET
- DROP TABLE
- DROP TRANSFER
- DROP VIEW
- DROP RESOURCE POOL
- DROP RESOURCE POOL CLASSIFIER
- DROP TOPIC
- DROP USER
- GRANT
- INSERT
- INTO RESULT
- KILL SESSION
- PRAGMA
- REPLACE
- REVOKE
- RESTORE
- SELECT
  - Обзор
  - FROM
  - FROM AS_TABLE
  - FROM SELECT
  - FROM Topic
  - FLATTEN

- GROUP BY
- JOIN
- WINDOW
- DISTINCT
- UNIQUE DISTINCT
- UNION
- VIEW SECONDARY INDEX
- VIEW VECTOR INDEX
- VIEW FULLTEXT INDEX
- VIEW JSON INDEX
- ORDER BY HybridRank
- WITH
- WITHOUT
- WHERE
- ORDER BY
- ASSUME ORDER BY
- LIMIT OFFSET
- SHOW CREATE
- TRUNCATE TABLE
- UPDATE
- UPSERT
- UPSERT OBJECT TYPE SECRET
- VALUES
- Встроенные функции
- Обзор
- Базовые
- Полнотекстовый поиск
- Агрегатные
- Оконные
- Для работы со списками
- Для работы со словарями
- Для работы со структурами
- Для работы с типами
- Для работы с генерацией кода
- Для работы с JSON
- Библиотеки C++
- Обзор
- DateTime
- Digest
- Histogram
- Hyperscan
- Ip
- Knn
- Math
- Pcre
- Pire
- PostgreSQL
- Roaring
- Re2
- String
- Unicode
- Url
- Yson
- Планы запросов
- YDB UI
- Обзор
- YDB Monitoring
- Hive web-viewer
- Обзор соединений
- Логи
- Графики
- Интеграции
- Графические пользовательские интерфейсы
- DBeaver
- DBeaver Plugin
- VS Code
- DataGrip
- Jupyter Notebook
- Визуализация данных (Business Intelligence, BI)
- Apache Superset
- DataLens
- Grafana data source
- Оркестрация

        - Apache Airflow™
    - Поставка данных
        - Сбор логов с помощью FluentBit
        - Сбор логов с помощью Logstash
    - Миграции данных
        - Импорт из источников данных JDBC
        - Импорт из MySQL
    - Миграции схемы
        - dbt
        - Flyway
        - goose
        - Liquibase
    - Движки выполнения запросов
        - Apache Spark™
    - Объектно-реляционное отображение
        - Hibernate
        - Spring Data JDBC
        - JOOQ
        - Dapper
        - Entity Framework
        - Linq To Db
        - SQLAlchemy
        - Django
        - Kotlin Exposed
        - Drizzle ORM
    - Spring
        - Автоматический повтор транзакций
    - Векторный поиск
        - LangChain
        - Mem0
    - Конвертация SQL-диалектов
        - SQLGlot
        - Конвертер SQL-диалектов в YQL
- YDB CLI
    - Обзор
    - Установка
    - Соединение с БД и аутентификация
    - Все команды по алфавиту
    - Сервисные команды
    - Глобальные параметры
    - Команды YDB CLI
        - table attribute add
        - table attribute drop
    - Работа со схемой БД
        - Список объектов
        - Информация об объекте
        - Права
        - Директории
        - Вторичные индексы
        - Копирование таблиц
        - Переименование таблиц
        - Установка параметров TTL
        - Сброс параметров TTL
        - Удаление таблицы
        - Генерация скрипта создания таблицы
    - Работа с данными
        - Получение плана исполнения запроса и AST
        - Потоковое чтение таблицы
    - Выгрузка и загрузка данных
        - Обзор
        - Файловая структура выгрузки
        - Выгрузка в файловую систему
        - Загрузка из файловой системы
        - Соединение и аутентификация с S3
        - Выгрузка в S3
        - Загрузка из S3
        - Выгрузка в NFS
        - Загрузка из NFS
        - Импорт данных из файла в существующую таблицу
    - Работа с топиками
        - Команды для работы с топиком
        - Создание топика
        - Изменение топика

        - Удаление топика
        - Добавление читателя топика
        - Удаление читателя топика
        - Сохранение позиции чтения
        - Чтение из топика
        - Запись в топик
        - Конвейерная обработка сообщений
      - Выполнение SQL-запросов
        - Обзор
        - Выполнение запросов
        - Выполнение параметризованных запросов
        - Интерактивный режим выполнения запросов
        - Режим AI-ассистента
        - Скан запросы
      - Управление фоновыми операциями
        - Получение списка фоновых операций
        - Получение статуса фоновой операции
        - Отмена фоновой операции
        - Удаление фоновой операции из списка
      - Управление профилями
        - Обзор
        - Создание и изменение профиля
        - Использование профиля в запросах
        - Получение информации о профиле
        - Удаление профиля
        - Активированный профиль
      - Информационные сервисы
        - Список эндпоинтов
        - Проверка аутентификации
        - Получение токена аутентификации
        - Просмотр параметров соединения
        - Вывод версии YDB CLI
        - Проверка состояния базы данных
      - Нагрузочное тестирование
        - Обзор
        - Stock нагрузка
        - ClickBench нагрузка
        - Key-Value нагрузка
        - Topic нагрузка
        - Transfer нагрузка
        - TPC-C нагрузка
        - TPC-H нагрузка
        - TPC-DS нагрузка
        - Пользовательская нагрузка
        - Fulltext нагрузка
        - Vector нагрузка
      - Работа с конфигурацией
        - Обзор
        - Команды кластера
          - Генерация конфигурации V2
          - Получение текущей конфигурации
          - Замена конфигурации
        - Команды узла
          - Инициализация конфигурационной директории
      - Управление кластером в режиме bridge
        - Обзор
        - ydb admin cluster bridge list
        - ydb admin cluster bridge switchover
        - ydb admin cluster bridge failover
        - ydb admin cluster bridge takedown
        - ydb admin cluster bridge rejoin
    - YDB Native SDK
      - Обзор
      - Установка SDK
      - Аутентификация в SDK
      - Параметризованные запросы
      - Форматы данных
        - Protobuf (Value)
        - Apache Arrow
      - Работа с топиками
      - Работа с узлами координации
      - Обработка ошибок в API
        - Статусы завершения сервера YDB

        - Статусы завершения gRPC
    - gRPC API
        - Обзор
        - Заголовки gRPC
        - Health Check API
    - Наблюдаемость
        - Обзор
        - Логирование
            - Включение логирования
            - Логи SDK в OpenTelemetry
        - Метрики
            - Метрики SDK в OpenTelemetry
            - Метрики SDK в Prometheus
        - Трассировка
            - Трассировка SDK в OpenTelemetry
            - Трассировка SDK в Jaeger
    - Сравнение возможностей SDK
- Языки и API
    - ADO.NET
        - Начало работы
        - Установка
        - Основное использование
        - Параметры подключения
        - Соответствие типов
        - Подключение к Yandex Cloud
    - JDBC-драйвер
        - Быстрый старт
        - Использование с Maven
        - Режимы аутентификации
        - Свойства
        - Сборка
    - Model Context Protocol
- AI-инструменты
    - YDB AI Skills
- Kafka API
    - Обзор
    - Аутентификация
    - Примеры чтения и записи
    - Kafka Connect
        - Обзор
        - Настройка Kafka Connect. Пошаговая инструкция
        - Примеры настройки коннекторов
    - Ограничения
- Amazon SQS API
    - Обзор
    - Аутентификация
    - Примеры
- Конфигурация
    - actor_system_config
    - auth_config
    - aws_client_config
    - blob_storage_config
    - bridge_config
    - client_certificate_authorization
    - cms_config
    - domains_config
    - feature_flags
    - query_service_config
    - healthcheck_config
    - hive_config
    - host_configs
    - hosts
    - immediate_controls_config
    - kafka_proxy_config
    - log_config
    - memory_controller_config
    - monitoring_config
    - node_broker_config
    - resource_broker_config
    - security_config
    - self_management_config
    - system_tablet_backup_config
    - tls

        - table_service_config
        - tli_config
    - Observability
        - Обзор
        - Метрики
            - Справка по метрикам
            - Дашборды Grafana
        - Трассировка
            - Настройка
            - Передача внешнего trace-id
    - YDB DStool
        - Обзор
        - Установка
        - Глобальные опции
        - Список устройств
        - Заполнение PDisk
    - ydbops
        - Обзор
        - Установка
        - Настройка
        - Сценарий перезапуска кластера
    - Docker
        - Именование тегов
        - Установка Docker
        - Работа с Docker
        - Настройка
        - Скрипты инициализации
        - Остановка и очистка
- Рецепты
    - Рецепты резервного копирования
        - Рецепты коллекций резервных копий
            - Начало работы
            - Настройка для разных сред
            - Стратегия для микросервисов
            - Экспорт во внешнее хранилище
            - Импорт и восстановление
            - Обслуживание и очистка
            - Проверка и тестирование
        - Рецепты резервного копирования системных таблеток
            - Включение и настройка
            - Восстановление
        - Резервное копирование в NFS
    - YDB SDK и фреймворки
        - Обзор
        - Инициализация драйвера
        - Аутентификация
            - Обзор
            - С помощью токена
            - Анонимная
            - Файл сервисного аккаунта
            - Сервис метаданных
            - С помощью переменных окружения
            - С помощью логина и пароля
        - Балансировка
            - Обзор
            - Равномерный случайный выбор
            - Предпочитать ближайший дата-центр
            - Предпочитать конкретную зону доступности
            - Предпочитать пайл с конкретным состоянием
        - Выполнение повторных запросов
        - Установить размер пула сессий
        - Вставка данных
        - Пакетная вставка данных
        - Установка режима выполнения транзакции
        - Настройка TTL
        - Векторный поиск
        - Координация
            - Распределённая блокировка
            - Выбор лидера
            - Обнаружение сервисов
            - Публикация конфигурации
    - YDB CLI
        - Конвертация типов таблиц

        - Нагрузочное тестирование
        - Настройка TTL таблиц
    - YQL
        - Обзор
        - Доступ к JSON
        - Модификация JSON
        - Настройка TTL
    - Векторный поиск
        - Векторный индекс - быстрый старт
        - Векторный индекс с загрузкой внешнего набора данных
    - Полнотекстовый поиск
        - Полнотекстовый индекс — быстрый старт
    - Поиск по JSON
        - JSON-индекс — быстрый старт
        - Каталог со вложенными атрибутами
        - Параметризованные запросы и переменные JsonPath
        - Проверка типа поля и наличия пути
    - Локальные индексы
        - Блум-индекс — быстрый старт
        - min_max-индекс — быстрый старт
    - Трансфер
        - Трансфер — быстрый старт
        - Трансфер — поставка access логов NGINX в таблицу
    - Потоковые запросы
        - Быстрый старт: чтение и запись в топики
        - Отладочное чтение из топика
- Диагностика проблем
    - Проблемы с производительностью
        - Инфраструктура
            - Сетевые проблемы
            - Сбои датацентров
            - Учения и работы по обслуживанию
            - Аппаратные проблемы
        - Нехватка ресурсов
            - Процессор
            - Оперативная память
            - Пропускная способность
            - Дисковое пространство
        - Операционная система
            - Расхождение системного времени
        - Конфигурация YDB
            - Процедура последовательного перезапуска
            - Частые переезды таблеток между узлами
        - Схемы таблиц
            - Перегруженные таблетки data shard
            - Избыточные разделения и слияния партиций таблиц
        - Клиентские приложения
            - Инвалидация блокировок транзакций
            - Логирование TLI
            - Ошибки «overloaded»
            - Каскадный эффект при частых повторных попытках
    - Проблемы со спиллингом
        - Permission denied
        - Spilling Service not started
        - Total size limit exceeded
    - Примеры диагностики
        - Перегруженный шард
        - Перегруженный шард и недостаточно ресурсов CPU
- Вопросы и ответы
    - Общие вопросы
    - SDK
    - Ошибки
    - YQL
    - Аналитика
    - Известные проблемы
    - Все вопросы на одной странице
- Загрузки
    - YDB Open-Source Database
    - Yandex Enterprise Database
    - YDB CLI
    - YDB DSTool
    - YDB Ops
    - YDB Ansible

- Публичные материалы
  - Видеоматериалы
    - 2026
    - 2025
    - 2024
    - 2023
    - 2022
    - 2021
    - 2020
    - 2019
    - 2018
    - 2017
  - Статьи
    - 2026
    - 2025
    - 2024
    - 2023
    - 2021
  - Подкасты
- Список изменений
  - YDB Server
  - Yandex Enterprise Database
  - YDB CLI
  - Список изменений безопасности
- Обучение
  - Курс для администраторов
  - Курс для разработчиков
  - Сертификация для администраторов
  - Сертификация для разработчиков
- Карьера
  - Команда
  - Студентам

Быстрый старт с YDB

В этом руководстве вы установите одноузловой локальный кластер YDB и выполните простые запросы к вашей базе данных.

Обычно YDB хранит данные напрямую на нескольких дисковых устройствах SSD/NVMe или HDD без использования файловой системы. Однако для простоты данное руководство эмулирует диски в оперативной памяти или с использованием файла в обычной файловой системе. Таким образом, эта конфигурация не подходит для использования в промышленном окружении и даже для проведения тестов производительности. Ознакомьтесь с разделом Администрирование кластеров для запуска YDB в production окружении.

Установите и запустите YDB

Linux x86_64

> **Примечание**
>
> Рекомендуемая среда для запуска YDB - это x86_64 Linux. Если у вас нет доступа к такой системе, можете переключиться на инструкции во вкладке "Docker".

1. Создайте каталог для тестирования YDB и используйте его в качестве текущего рабочего каталога:

   ```
   mkdir ~/ydbd && cd ~/ydbd
   ```

2. Загрузите и запустите скрипт установки:

   ```
   curl https://install.ydb.tech | bash
   ```

   Это действие загрузит и распакует архив с исполняемым файлом `ydbd`, библиотеками, файлами конфигурации и скриптами, необходимыми для запуска и остановки локального кластера.

   Скрипт выполняется с текущими привилегиями пользователя (обратите внимание на отсутствие `sudo`). Таким образом, он не может сделать многого в системе. Чтобы проверить какие именно команды он выполняет — откройте этот же URL в вашем браузере.

3. Запустите кластер в одном из следующих режимов хранения данных:

   - Данные в оперативной памяти:

     ```
     ./start.sh ram
     ```

     В этом случае все данные хранятся только в оперативной памяти, и они будут потеряны при остановке кластера.

   - Данные в файле на диске:

     ```
     ./start.sh disk
     ```

     При первом запуске скрипта будет создан файл `ydb.data` размером 80 ГБ в рабочем каталоге. Убедитесь, что у вас достаточно свободного места на диске для его создания. Этот файл будет использоваться для эмуляции дискового устройства, которое использовалось бы в промышленном окружении.

   - Данные на дисковом устройстве:

     ```
     ./start.sh drive "/dev/$DRIVE_NAME"
     ```

     Замените `/dev/$DRIVE_NAME` на настоящее имя устройства, которое не используется для других целей, например `/dev/sdb`. В первый раз при запуске этой команды указанный дисковый накопитель будет полностью очищен, а затем будет подключен для хранения данных YDB. Рекомендуется использовать NVMe или SSD-диск с объемом не менее 800 ГБ. Такая конфигурация может быть использована для тестирования производительности на одном узле или других сред, которые **не** имеют каких-либо требований к отказоустойчивости.

   Результат:

   ```
   Starting storage process...
   Initializing storage ...
   Registering database ...
   Starting database process...

   Database started. Connection options for YDB CLI:

   -e grpc://localhost:2136 -d /Root/test
   ```

Docker x86_64

> **Примечание**
>
> При использовании Mac с процессором Apple Silicon, набор процессорных инструкций x86_64 можно эмулировать с помощью Rosetta:
>
> - colima с параметрами `colima start --arch aarch64 --vm-type=vz --vz-rosetta`;
> - Docker Desktop с установленной и включённой Rosetta 2.
>
> Если не включена Rosetta 2, в команду запуска Docker-контейнера необходимо добавить параметр `-e YDB_USE_IN_MEMORY_PDISKS=true`. См. Настройка Docker-контейнера YDB.

1. Создайте каталог для тестирования YDB и используйте его в качестве текущего рабочего каталога:

```
mkdir ~/ydbd && cd ~/ydbd
mkdir ydb_data
mkdir ydb_certs
```

2. Запустите Docker-контейнер:

```
docker run -d --rm --name ydb-local -h localhost \
  --platform linux/amd64 \
  -p 2135:2135 -p 2136:2136 -p 8765:8765 -p 9092:9092 \
  -v $(pwd)/ydb_certs:/ydb_certs -v $(pwd)/ydb_data:/ydb_data \
  -e GRPC_TLS_PORT=2135 -e GRPC_PORT=2136 -e MON_PORT=8765 \
  -e YDB_KAFKA_PROXY_PORT=9092 \
  ydbplatform/local-ydb:latest
```

   Если контейнер успешно запустился, вы увидите его идентификатор. Контейнеру может потребоваться несколько секунд для инициализации. База данных будет недоступна до окончания инициализации.

Minikube

1. Установите интерфейс командной строки Kubernetes kubectl и менеджер пакетов Helm 3.
2. Установите и запустите Minikube.
3. Склонируйте репозиторий с YDB Kubernetes Operator:

```
git clone https://github.com/ydb-platform/ydb-kubernetes-operator && cd ydb-kubernetes-operator
```

4. Установите на кластер контроллер YDB:

```
helm upgrade --install ydb-operator deploy/ydb-operator --set metrics.enabled=false
kubectl rollout status deployment/ydb-operator --timeout=5m
```

5. Примените манифест для создания кластера YDB:

   Сначала проверьте, что API-сервер может вызвать admission webhook контроллера и принять манифест, не создавая ресурс:

```
kubectl apply --dry-run=server -f samples/minikube/storage.yaml
```

   Даже после завершения `rollout status` webhook может быть ещё недоступен. Если проверка сообщает `connection refused` или `no endpoints available` при обращении к webhook, подождите несколько секунд и повторите её. Другие ошибки требуют устранения причины. Продолжайте только после успешной проверки:

```
kubectl apply -f samples/minikube/storage.yaml
```

6. Подождите, пока `kubectl get storages.ydb.tech` станет `Ready`.
7. Примените манифест для создания базы данных:

```
kubectl apply -f samples/minikube/database.yaml
```

8. Подождите, пока `kubectl get databases.ydb.tech` станет `Ready`.
9. После обработки манифеста создаётся объект StatefulSet, описывающий набор динамических узлов. Для подключения из того же пространства имён Kubernetes используйте отдельные сервисы:
   - `database-minikube-sample-grpc:2135` (gRPC);
   - `database-minikube-sample-status:8765` (YDB UI).

   Путь базы данных: `/Root/database-minikube-sample`.
10. Получите доступ к порту 8765 снаружи Kubernetes через `kubectl port-forward database-minikube-sample-0 8765` для продолжения.

Kind

1. Установите интерфейс командной строки Kubernetes kubectl и менеджер пакетов Helm 3.
2. Установите Kind.
3. Склонируйте репозиторий с YDB Kubernetes Operator:

```
git clone https://github.com/ydb-platform/ydb-kubernetes-operator && cd ydb-kubernetes-operator
```

4. Создайте Kind кластер и дождитесь его готовности:

```
kind create cluster --config=samples/kind/kind-config.yaml  --wait 5m
```

5. Установите на кластер контроллер YDB:

```
helm upgrade --install ydb-operator deploy/ydb-operator --set metrics.enabled=false
kubectl rollout status deployment/ydb-operator --timeout=5m
```

6. Примените манифест для создания хранилища данных YDB:

   Сначала проверьте, что API-сервер может вызвать admission webhook контроллера и принять манифест, не создавая ресурс:

```
kubectl apply --dry-run=server -f samples/kind/storage.yaml
```

   Даже после завершения `rollout status` webhook может быть ещё недоступен. Если проверка сообщает `connection refused` или `no endpoints available` при обращении к webhook, подождите несколько секунд и повторите её. Другие ошибки требуют устранения причины. Продолжайте только после успешной проверки:

```
kubectl apply -f samples/kind/storage.yaml
```

7. Подождите, пока `kubectl get storages.ydb.tech` станет `Ready`.
8. Примените манифест для создания базы данных:

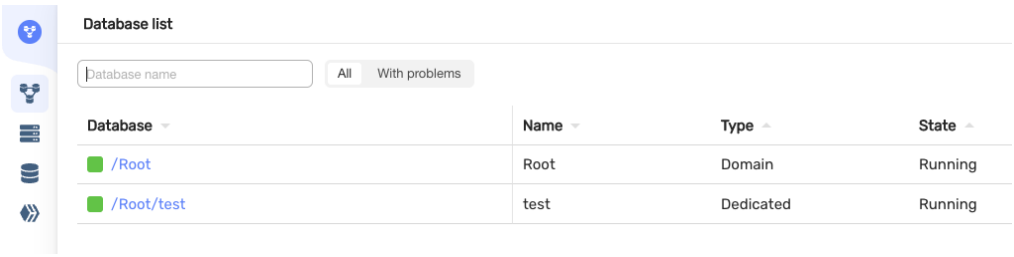
```
kubectl apply -f samples/kind/database.yaml
```

9. Подождите, пока `kubectl get databases.ydb.tech` станет `Ready`.
10. После обработки манифеста создаётся объект StatefulSet, описывающий набор динамических узлов. Для подключения из того же пространства имён Kubernetes используйте отдельные сервисы:
    - `database-kind-sample-grpc:2135` (gRPC);
    - `database-kind-sample-status:8765` (YDB UI).

    Путь базы данных: `/Root/database-kind-sample`.
11. Получите доступ к порту 8765 снаружи Kubernetes через `kubectl port-forward database-kind-sample-0 8765` для продолжения.

Запустите первый запрос "Hello, world!"

Самый простой способ выполнить свой первый запрос к YDB — это YDB UI. Он запускается по умолчанию на порту 8765 сервера YDB, поэтому, если вы запустили его локально, вам нужно открыть localhost:8765 в вашем веб-браузере. Если нет, замените `localhost` на имя хоста вашего сервера в этом URL, либо используйте `ssh -L 8765:localhost:8765 my-server-hostname-or-ip.example.com` для настройки проброса порта и все равно откройте localhost:8765. Вы увидите страницу подобного вида:



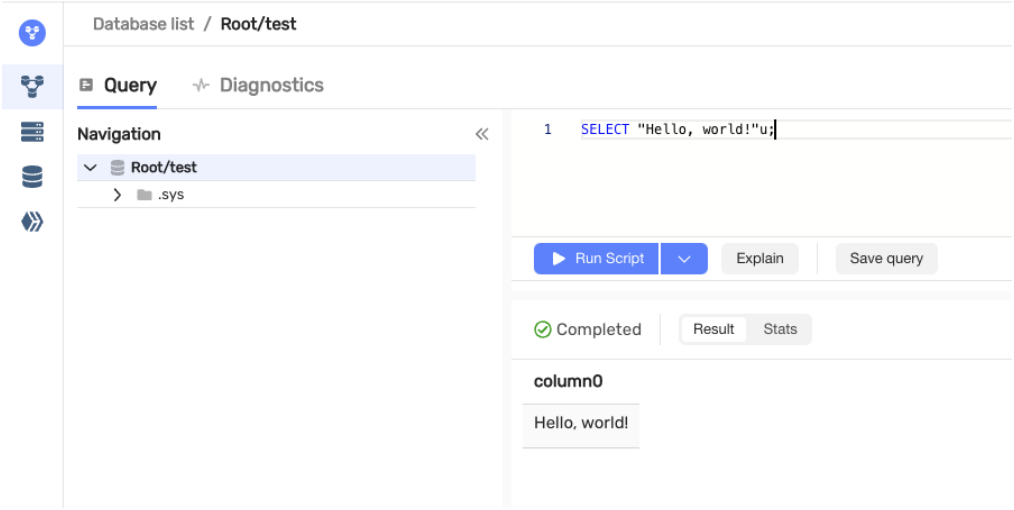
YDB спроектирован как многопользовательская система с возможностью одновременной работы тысяч пользователей с одним кластером. Как следствие, большинство логических сущностей внутри кластера YDB находятся в гибкой иерархической структуре, больше похожей на виртуальную файловую систему Unix, чем на схему с фиксированной глубиной, с которой вы, возможно, знакомы из других систем управления базами данных. Как видите, первый уровень иерархии состоит из баз данных, работающих в одном процессе YDB, которые могут принадлежать разным пользователям. `/Root` предназначена для системных целей. Для выполнения запросов выберите базу данных, созданную на предыдущем шаге:

Способ установки	База данных
Linux x86_64	`/Root/test`
Docker x86_64	`/local`
Minikube	`/Root/database-minikube-sample`
Kind	`/Root/database-kind-sample`

Нажмите на соответствующую базу данных, затем введите первый запрос и нажмите кнопку запуска:

```
SELECT "Hello, world!"u;
```

Запрос возвращает приветствие, как и задумано:



Примечание

Заметили странный суффикс `u`? YDB и её язык запросов YQL являются строго типизированными. Обычные строки в YDB могут содержать любые двоичные данные, в то время как этот суффикс указывает, что этот литерал имеет тип данных `Utf8`, который может содержать только валидные последовательности UTF-8. Узнайте больше о системе типов YDB.

Второй по простоте способ выполнения SQL-запроса с использованием YDB - это интерфейс командной строки (CLI). Большинство реальных приложений же, скорее всего, будут работать с YDB через один из доступных наборов инструментов для разработчиков программного обеспечения (SDK). Если вы чувствуете себя уверенно — можете продолжить прохождение данного руководства с помощью одного из этих методов вместо веб-интерфейса.

Создайте свою первую таблицу

Основная цель существования систем управления базами данных - сохранение данных для последующего извлечения. Как система, базирующаяся на SQL, основной абстракцией YDB для хранения данных является таблица. Чтобы создать нашу первую таблицу, выполните следующий запрос:

```
CREATE TABLE example
(
    key UInt64,
    value String,
    PRIMARY KEY (key)
);
```

Как видите, это простая таблица ключ-значение. Давайте пройдемся по ней пошагово:

- Каждый тип оператора SQL вроде `CREATE TABLE` имеет подробное описание в справке по YQL.
- `example` - это идентификатор имени таблицы, а `key` и `value` - идентификаторы имен столбцов. Рекомендуется использовать простые имена для таких идентификаторов, но если вам нужно использовать имя с необычными символами, оберните его в обратные кавычки.
- `UInt64` и `String` - это названия типов данных. `String` представляет собой двоичную строку, а `UInt64` - 64-разрядное беззнаковое целое число. Таким образом, наша таблица-пример хранит строковые значения, идентифицируемые беззнаковыми целочисленными ключами. Подробнее о типах данных.
- `PRIMARY KEY` - одно из основных понятий SQL, которое оказывает огромное влияние на логику приложения и производительность. В соответствии со стандартом SQL, первичный ключ также подразумевает ограничение уникальности, поэтому таблица не может иметь несколько строк с одинаковыми первичными ключами. В этой примерной таблице довольно просто определить, какой столбец пользователь должен выбрать в качестве первичного ключа, и мы указываем его как `(key)` в круглых скобках после соответствующего ключевого слова. В реальных сценариях таблицы часто содержат десятки столбцов, и первичные ключи могут быть составными (состоять из нескольких столбцов в указанном порядке), поэтому выбор правильного первичного ключа становится больше похожим на искусство. Если вас интересует эта тема, есть руководство по выбору первичного ключа для максимизации производительности. YDB таблицы обязаны иметь первичный ключ.

Добавление тестовых данных

Теперь давайте заполним нашу таблицу первыми данными. Самый простой способ - использовать литералы:

```
INSERT INTO example (key, value)
VALUES (123, "hello"),
       (321, "world");
```

Пошаговое описание:

- `INSERT INTO` - это классический оператор SQL для добавления новых строк в таблицу. Однако он не является наиболее производительным, так как согласно стандарту SQL он должен проверить, существуют ли в таблице строки с заданными значениями первичного ключа, и выдать ошибку, если они уже есть. Таким образом, если вы запустите этот запрос несколько раз, все попытки, кроме первой, вернут ошибку. Если логика вашего приложения не требует такого поведения, лучше использовать `UPSERT INTO` вместо `INSERT INTO`. Upsert (от "update or insert") будет безусловно записывать предоставленные значения, перезаписывая существующие строки, если они есть. Остальной синтаксис будет таким же.
- `(key, value)` указывает имена столбцов, которые мы вставляем, и их порядок. Предоставленные значения должны соответствовать этому описанию как по количеству столбцов, так и по их типам данных.

- После ключевого слова `VALUES` следует список кортежей, каждый из которых представляет собой строку таблицы. В этом примере у нас есть две строки, идентифицируемые числами 123 и 321 в столбце `key`, и значениями "hello" и "world" в столбце `value` соответственно.

Чтобы убедиться, что строки были действительно добавлены в таблицу, существует распространённый запрос, который в данном случае должен вернуть `2`:

```
SELECT COUNT(*) FROM example;
```

Несколько деталей в этом запросе:

- Во `FROM` указывают таблицу, из которой извлекаются данные.
- `COUNT` - это агрегатная функция, подсчитывающая количество значений. По умолчанию, когда нет других специальных выражений вокруг, наличие любой агрегатной функции приводит к сворачиванию результата в одну строку, содержащую агрегаты по всему входным данным (таблице `example` в данном случае).
- Астериск `*` является спецсимволом, который обычно означает "все столбцы"; таким образом, `COUNT` вернет общее количество число строк.

Еще один распространённый способ заполнить таблицу данными - это объединить операции `INSERT INTO` (или `UPSERT INTO`) и `SELECT`. В этом случае значения для сохранения вычисляются внутри базы данных, а не предоставляются клиентом в виде литералов. Для демонстрации этого подхода мы используем немного более реалистичный запрос:

```
$subquery = SELECT ListFromRange(1000, 10000) AS keys;

UPSERT INTO example
SELECT
    key,
    CAST(RandomUuid(key) AS String) AS value
FROM $subquery
FLATTEN LIST BY keys AS key
```

В этом запросе происходит немало интересного, давайте рассмотрим его подробнее:

- `$subquery` - это именованное выражение. Этот синтаксис является расширением YQL по сравнению со стандартным SQL и позволяет делать более читаемые сложные запросы. Он работает так же, как если бы вы написали первый `SELECT` по месту, где `$subquery` используется в последней строке. Однако, использование именованных выражений позволяет легче понять, что происходит шаг за шагом, подобно переменным в обычных языках программирования.
- `ListFromRange` - это функция, которая создает список последовательных целых чисел, начиная с значения, указанного в первом аргументе, и заканчивая значением, указанным во втором аргументе. Также есть третий необязательный аргумент, который позволяет пропускать числа с указанным шагом, но мы не используем его в нашем примере — по умолчанию возвращаются все целые числа в указанном диапазоне. `List` является одним из наиболее распространенных контейнерных типов данных.
- `AS` - это ключевое слово, используемое для задания имени значения, которое мы возвращаем из `SELECT`; в данном примере - `keys`.
- В части `FROM ... FLATTEN LIST BY ... AS ...` есть несколько значимых моментов:
  - Другой `SELECT`, использованный в выражении `FROM`, называется подзапросом. Поэтому мы выбрали это имя для нашего именованного выражения `$subquery`, но мы могли бы выбрать что-то более значимое, чтобы объяснить, что это такое. Подзапросы обычно не материализуются; они просто передают вывод одного `SELECT` вводу другого на лету. Они могут использоваться в качестве средства для создания произвольно сложных графов выполнения, особенно в сочетании с другими возможностями YQL.
  - Ключевая фраза `FLATTEN LIST BY` изменяет входные данные, передаваемые через `FROM`, следующим образом: для каждой строки во входных данных она берет столбец типа данных `List` и создает несколько строк в соответствии с количеством элементов в этом списке. Обычно этот столбец списка заменяется столбцом с текущим одиночным элементом, но ключевое слово `AS` в данном контексте позволяет получить доступ и ко всему списку (по исходному имени) и текущему элементу (по имени после `AS`).
- `RandomUuid` - это функция, которая возвращает псевдослучайный UUID версии 4. В отличие от большинства других функций, она не использует значение своего аргумента (столбец `key`); вместо этого этот аргумент указывает на то, что нам нужно вызывать функцию для каждой строки. Смотрите ссылку с дополнительными примерами работы этой функции.
- `CAST(... AS ...)` - это часто используемая функция для преобразования значений в указанный тип данных. В этом контексте после `AS` ожидается указание типа данных (в данном случае `String`), а не произвольного имени.
- `UPSERT INTO` слепо записывает значения в указанные таблицы, как мы обсуждали ранее. При использовании в сочетании с `SELECT` он не требует явного указания имен столбцов `(key, value)`, так как столбцы теперь могут быть просто сопоставлены по именам, возвращённым из `SELECT`.

Короткий вопрос!

Что теперь вернёт запрос `SELECT COUNT(*) FROM example;`?

Остановка кластера

Остановите локальный кластер YDB после завершения экспериментов:

Linux x86_64

Чтобы остановить локальный кластер, выполните следующую команду:

```
~/ydbd/stop.sh
```

При желании вы можете удалить вашу рабочую директорию с помощью команды `rm -rf ~/ydbd` для очистки файловой системы. Все данные внутри локального кластера YDB будут потеряны.

Docker

Чтобы остановить Docker контейнер с локальным кластером, выполните следующую команду:

```
docker kill ydb-local
```

При желании вы можете удалить вашу рабочую директорию с помощью команды `rm -rf ~/ydbd` для очистки файловой системы. Все данные внутри локального кластера YDB будут потеряны.

Minikube

Чтобы удалить базу данных YDB, достаточно удалить сопоставленный с ней ресурс Database:

```
kubectl delete database.ydb.tech database-minikube-sample
```

Чтобы удалить кластер YDB, выполните следующие команды (все данные будут потеряны):

```
kubectl delete storage.ydb.tech storage-minikube-sample
```

Чтобы удалить контроллер YDB из кластера Kubernetes, удалите релиз, созданный Helm:

```
helm delete ydb-operator
```

Kind

Чтобы удалить базу данных YDB, достаточно удалить сопоставленный с ней ресурс Database:

```
kubectl delete database.ydb.tech database-kind-sample
```

Чтобы удалить кластер YDB, выполните следующие команды (все данные будут потеряны):

```
kubectl delete storage.ydb.tech storage-kind-sample
```

Чтобы удалить контроллер YDB из кластера Kubernetes, удалите релиз, созданный Helm:

```
helm delete ydb-operator
```

Чтобы удалить кластер Kind целиком, выполните следующую команду:

```
kind delete cluster
```

Готово! Что дальше?

После освоения базовых операций из этого руководства, пора переходить перейти к более глубоким темам. Выберите то, что кажется наиболее актуальным в зависимости от вашего сценария использования и роли:

- Пройдите более подробный курс по YQL, который сосредоточен на написании запросов.
- Попробуйте создать свое первое приложение для хранения данных в YDB с использованием одного из SDK.
- Узнайте, как настроить развертывание YDB в готовую к промышленной эксплуатации среду.
- Почитайте об используемых в YDB концепциях.

Концепции YDB

Данный раздел документации охватывает фундаментальные концепции и архитектурные принципы YDB. Эта информация поможет вам лучше понимать, что происходит при изучении более практического контента, который можно найти в других разделах, предназначенных для конкретных ролей.

Следующие темы обеспечивают всестороннее освещение основной функциональности YDB, от высокоуровневой архитектуры и модели данных до продвинутых возможностей, таких как транзакции, индексирование и федеративные запросы.

- Глоссарий
- Аналитические возможности
- Потоковая обработка
- Архитектура
- Соединение с БД
- Схемные объекты
- Топология кластера
- Выполнение запросов
- Транзакции
- Change Data Capture (CDC)
- Time to Live (TTL) и вытеснение данных во внешнее хранилище
- Ограничения базы данных
- Асинхронная репликация
- Режим bridge
- Трансфер данных

См. также

- Администрирование кластеров
- YDB для разработчиков приложений
- YDB для инженеров по безопасности

Глоссарий YDB

Эта статья представляет собой обзор терминов и определений, используемых в YDB и её документации. Он начинается с ключевых терминов, с которыми полезно ознакомиться на раннем этапе работы с YDB, а остальная часть статьи содержит более продвинутые термины, которые могут быть полезны в дальнейшем.

Ключевая терминология

Этот раздел описывает термины, которые полезны любому человеку, работающему с YDB, независимо от его роли и сценария использования.

Кластер

Кластер или **cluster** YDB представляет собой набор взаимосвязанных узлов YDB, которые обмениваются данными для выполнения пользовательских запросов и надёжного хранения данных. Эти узлы формируют одну из поддерживаемых топологий кластера, что напрямую влияет на его характеристики надёжности и производительности.

Кластеры YDB являются мультитенантными (многопользовательскими) и могут содержать несколько изолированных баз данных.

База данных

Как и в большинстве систем управления базами данных, **база данных** в YDB представляет собой логический контейнер для других сущностей, таких как таблицы. Однако в YDB пространство имён внутри баз данных иерархическое, как в виртуальных файловых системах, и таким образом директории позволяют более структурированно организовывать сущности.

Ещё одной важной характеристикой баз данных YDB является то, что им обычно выделяются отдельные вычислительные ресурсы. Как следствие, создание базы данных требует дополнительных действий администраторов кластера.

Узел

YDB **узел** или **node** — это серверный процесс, выполняющий исполняемый файл под названием `ydbd`. На одном физическом сервере или виртуальной машине могут работать несколько узлов YDB, что является обычной практикой. Таким образом, в контексте YDB узлы **не** являются синонимами хостов.

Так как YDB использует подход раздельных слоёв хранения и вычислений (storage and compute separation), `ydbd` имеет несколько режимов работы, определяющих тип узла. Доступные типы узлов описаны ниже.

Узел базы данных

Узлы базы данных (также известные как **тенантные узлы**, **узлы вычислений**, **database nodes**, **tenant nodes** или **compute nodes**) обрабатывают пользовательские запросы, адресованные конкретной логической базе данных. Их состояние находится только в оперативной памяти и может быть восстановлено из распределённого хранилища. Совокупность узлов баз данных заданного кластера YDB можно считать вычислительным слоем этого кластера. Таким образом, добавление узлов баз данных и выделение им дополнительных ресурсов (CPU и RAM) — это основные способы увеличения вычислительных ресурсов базы данных.

Основная роль узлов базы данных — запуск различных таблеток и акторов, а также приём входящих запросов по сети.

Узел хранения

Узлы хранения или **storage nodes** являются узлами с состоянием, отвечающими за долгосрочное хранение фрагментов данных. Совокупность узлов хранения заданного кластера YDB называются распределённым хранилищем и могут рассматриваться как слой хранения этого кластера. Таким образом, добавление дополнительных узлов хранения и их дисков — это основные способы увеличения ёмкости хранилища и пропускной способности ввода/вывода кластера.

Гибридный узел

Гибридный узел — это процесс, который одновременно выполняет обе роли узла базы данных и узла хранения. Гибридные узлы часто используются в целях разработки. Например, вы можете запустить контейнер с полнофункциональным YDB, содержащим только один процесс `ydbd` в гибридном режиме. В производственных окружениях они используются редко.

Статический узел

Статические узлы или **static nodes** конфигурируются вручную во время первоначальной инициализации или перенастройки кластера. Как правило, они выполняют роль узлов хранения, но технически возможно настроить их и в качестве узлов баз данных.

Динамический узел

Динамические узлы или **dynamic nodes** добавляются и удаляются из кластера на ходу. Они могут выполнять только роль узлов базы данных.

Распределённое хранилище

Распределённое хранилище, **Distributed storage**, **Blob storage** или **BlobStorage** — это распределённый отказоустойчивый слой хранения данных в YDB. Оно имеет специализированный API, предназначенный для хранения неизменяемых фрагментов данных таблетки.

Множество терминов, связанных с реализацией распределённого хранилища, рассматриваются ниже.

Группа хранения

Группа хранения, **группа распределённого хранилища**, **storage group** или **группа Blob storage** — это место для надёжного хранения данных, аналогичное RAID, но использующее диски нескольких серверов. В зависимости от выбранной топологии кластера группы хранения используют различные алгоритмы для обеспечения высокой доступности, аналогично стандартным уровням RAID.

Распределённое хранилище обычно управляет большим количеством относительно небольших групп хранения. Каждую группу можно назначить конкретной базе данных для увеличения ёмкости дискового пространства и пропускной способности ввода/вывода, доступных для этой базы данных.

Статические и динамические группы хранения являются физическими, то есть их данные размещаются непосредственно на VDisk'ax

Статическая группа

Статическая группа или **static group** — это специальная группа хранения, созданная во время изначального развёртывания кластера. Её основная роль заключается в хранении данных системных таблеток, которые можно рассматривать как метаданные уровня всего кластера.

Статическая группа может требовать особого внимания во время крупного обслуживания кластера, такого как вывод зоны доступности из эксплуатации.

Динамическая группа

Обычные группы хранения, которые не являются статическими, называются **динамическими группами** или **dynamic group**. Их называют динамическими, потому что они могут быть созданы и удалены на лету во время работы кластера.

Виртуальная группа хранения

Виртуальная группа хранения или **virtual storage group** — это сущность, которая фактически не является группой хранения, но со стороны выглядит таковой (предоставляет аналогичный внешний интерфейс). Может хранить свои данные в других группах хранения или в S3.

BlobDepot

BlobDepot — системная таблетка распределённого хранилища, которая обслуживает виртуальные группы хранения. Она хранит данные виртуальной группы в других группах хранения и используется, в частности, при выводе физических групп из эксплуатации.

Пул хранения

Пул хранения или **storage pool** — это набор устройств хранения данных со схожими характеристиками. Каждому пулу хранения присваивается уникальное имя в рамках кластера YDB. Технически, каждый пул хранения состоит из множества физических дисков (PDisk). Каждая группа хранения создаётся в определённом пуле хранения, который определяет характеристики производительности группы хранения через выбор соответствующих устройств хранения. Обычно отдельные пулы хранения создают для устройств разных типов (например, NVMe, SSD и HDD) или для конкретных моделей этих устройств, обладающих различной ёмкостью и скоростью доступа.

Актор

Модель акторов — это один из основных подходов к параллелизму выполнения, используемый в YDB. В этой модели **акторы** или **actor** — это легковесные процессы в пользовательском пространстве, которые могут иметь и изменять своё частное состояние, но могут влиять друг на друга только косвенно через обмен сообщениями. YDB имеет собственную реализацию этой модели, которая описана ниже.

В YDB акторы с надёжно сохраняемым состоянием называются таблетками.

Таблетка

Таблетка — это один из основных строительных блоков и абстракций YDB. Она представляет собой сущность, ответственную за относительно небольшой сегмент пользовательских или системных данных. Обычно таблетка управляет объёмом данных до нескольких гигабайт, однако некоторые типы таблеток могут обрабатывать и больший объём.

Например, строковая пользовательская таблица управляется одной или несколькими таблетками типа DataShard, причём каждая таблетка отвечает за непрерывный диапазон первичных ключей и соответствующих им данных.

Конечным пользователям, отправляющим запросы в кластер YDB для выполнения, не нужно знать подробностей о таблетках, их видах или принципах работы, но это может быть полезно, например, для оптимизации производительности.

Технически таблетки являются акторами с состоянием, надёжно сохранённым в распределённом хранилище. Это состояние позволяет таблетке продолжать работу на другом узле базы данных, если предыдущий вышел из строя или перегружен.

Подробности реализации таблеток и связанные термины, а также основные типы таблеток, рассматриваются ниже.

Транзакции

YDB реализует **транзакции** (**transactions**) на двух основных уровнях:

- Локальная база данных и остальная инфраструктура таблеток позволяют таблеткам манипулировать своим состоянием, используя **локальные транзакции** с уровнем изоляции serializable. Технически они не являются локальными для одного узла, так как это состояние сохраняется удалённо в распределённом хранилище.
- В контексте YDB термин **распределённые транзакции** обычно относится к транзакциям, охватывающим несколько таблеток. Например, транзакции между таблицами или даже строками одной таблицы часто являются распределёнными.
- **Одношардовые** транзакции охватывают одну таблетку и выполняются быстрее. Например, транзакции между строками одной партиции таблицы часто являются одношардовыми.

Эти механизмы позволяют YDB обеспечивать строгую согласованность.

Реализация распределённых транзакций рассмотрена в отдельной статье DataShard: распределённые транзакции, а ниже приведён список нескольких связанных терминов.

Сессии

Логические соединения с базой данных, которые хранят контекст, необходимый для выполнения запросов и управления транзакциями. Более подробно сессии описаны в разделе Сессии.

Клиентский таймаут

Клиентский таймаут или **client-side timeout** — ограничение времени, которое приложение или YDB SDK ждёт завершения операции с базой (например, выполнения запроса или приёма ответа по вызову gRPC). По истечении этого времени клиент обычно прерывает ожидание: закрывает соединение или поток данных, получает ошибку по транспорту или из SDK — ещё до того, как сервер успел вернуть явный ответ (см. коды ответа сервера YDB).

Если клиентский таймаут меньше длительности выполнения запроса на стороне YDB, то из-за особенностей обработки запросов в кластере прерванный на клиенте запрос может некоторое время продолжать выполняться на сервере. Если подобная ситуация возникает массово, то сервер оказывается перегружен запросами, ответ на которые клиент не ждет. Поэтому частые повторы одного и того же запроса сразу после таймаута могут усугублять перегрузку. Подробнее см. в статьях Каскадный эффект при частых повторных попытках и Ошибки «overloaded»; политики повторов в SDK — в разделе Обработка ошибок.

Повтор транзакций

Повтор транзакций (**transaction retry**) — клиентская практика повторного выполнения транзакции целиком с начала при ошибке, допускающей повтор (например, временный сбой сети или конфликт оптимистических блокировок). В YDB повторы следует выполнять на уровне транзакции, а не отдельных запросов внутри неё. Встроенные политики повторов в SDK YDB и интеграциях (например, spring-ydb-retry) реализуют этот подход. Подробнее см. Обработка ошибок.

Экспоненциальная задержка

Экспоненциальная задержка (**exponential backoff**, **backoff**) — стратегия паузы между попытками повтора транзакций: интервал ожидания растёт экспоненциально с каждой попыткой, обычно с верхним пределом. SDK YDB часто используют два уровня backoff — быстрый и медленный — в зависимости от типа ошибки. Подробнее см. Обработка временных сбоев (retryable errors).

Джиттер

Джиттер (**jitter**) — небольшая случайная вариация, добавляемая к задержкам экспоненциальной задержки. Она помогает избежать одновременных повторов многих клиентов после общего сбоя («шторма повторов») и равномернее распределяет нагрузку.

Идемпотентность

Идемпотентность (**idempotency**) — свойство операции: повторное выполнение даёт тот же эффект, что и однократное (например, `UPSERT` с детерминированным первичным ключом или операции чтения). Повтор транзакций безопасен только для идемпотентных операций или при ошибках повтора, при которых сервер гарантирует, что транзакция не была зафиксирована. SDK и клиентские библиотеки YDB могут расширять набор повторяемых кодов статуса, если вызывающий код помечает операцию как идемпотентную.

Перехватчик транзакций

Перехватчик транзакций (**transaction interceptor**) — компонент Spring Framework, который оборачивает методы с аннотацией `@Transactional` и управляет границами транзакции. Модули вроде spring-ydb-retry подменяют стандартный перехватчик Spring, добавляя логику повтора транзакций вокруг транзакционных методов.

Неявные транзакции

Неявная транзакция — это режим выполнения запросов, при котором режим транзакции не указан. В этом случае YDB самостоятельно определяет, нужно ли обернуть их в транзакцию. Этот режим описан более подробно в Неявные транзакции.

Интерактивные транзакции

Интерактивная транзакция — это транзакция, в которой выполняется несколько запросов, между которыми может использоваться логика приложения. Она позволяет программе отправлять несколько запросов и принимать решения о дальнейшем ходе транзакции на основании полученных данных. Такая транзакция может явно начинаться отдельным шагом (либо открываться вместе с первым запросом) и завершаться отдельно. В отличие от одного запроса, который выполняется и фиксируется атомарно, интерактивная транзакция позволяет строить сложные рабочие процессы, в которых несколько связанных операций рассматриваются как единая атомарная единица. При этом она приводит к накладным расходам на систему, так как требует дольше хранить оптимистичные блокировки и незавершённые изменения, а также повышает сложность кода на стороне приложения.

Многоверсионное управление параллелизмом

Многоверсионное управление параллелизмом, **multi-version concurrency control** или **MVCC** — это метод, используемый YDB для одновременного доступа нескольких параллельных транзакций к базе данных без взаимных помех. Он описан более подробно в отдельной статье Многоверсионное управление конкурентным доступом (MVCC).

Потоковые запросы

Тип запроса, предназначенный для потоковой обработки неограниченного потока данных. В отличие от обычных запросов, потоковые запросы не имеют ограничений на длительность выполнения, автоматически перезапускаются при ошибках и периодически сохраняют своё состояние в виде чекпоинтов для обеспечения отказоустойчивости. Для отслеживания прогресса обработки по времени события используются watermarks.

Потоковые запросы описаны более подробно в отдельной статье Потоковые запросы.

Чекпоинты потоковых запросов

Периодически сохраняемое состояние потокового запроса, необходимое для автоматического восстановления его работы после сбоев в распределённой системе. Подробнее про чекпоинты в статье Чекпоинты.

Watermarks потоковых запросов

Монотонно возрастающая нижняя оценка времён событий в потоковом запросе, которые ещё могут поступить в поток. Когда watermark достигает значения X, система объявляет, что все события с временем меньше X с высокой вероятностью уже получены. Подробнее про watermarks в статье Watermarks.

Топология

YDB поддерживает несколько **топологий** кластера (или **topology**), описанных более подробно в отдельной статье Топология кластера YDB. Ниже объяснены несколько связанных терминов.

Зоны доступности и регионы

Зона доступности — это дата-центр или его изолированный сегмент с минимальным физическим расстоянием между узлами и минимальным риском отказа одновременно с другими зонами доступности. Таким образом, зоны доступности не должны использовать общую инфраструктуру, такую как электропитание, охлаждение или внешние сетевые соединения.

Регион — это большая географическая область, содержащая несколько зон доступности. Расстояние между зонами доступности в одном регионе должно составлять около 500 км или менее. YDB выполняет записи данных в каждую зону доступности в регионе синхронно, обеспечивая разумную задержку и бесперебойную работу в случае отказа одной из зон доступности.

Стойка

Стойка, **серверная стойка**, **rack** или **server rack** — это оборудование, используемое для организации размещения нескольких серверов. Серверы в одной стойке с большей вероятностью могут стать недоступными одновременно из-за

проблем на уровне стойки, связанных с электропитанием, охлаждением и т. д. YDB может учитывать информацию о том, какой сервер находится в какой стойке, при размещении каждого фрагмента данных в средах, основанных на физических серверах.

Pile

Pile или **пайл** — это набор узлов, которые могут отказать или быть отключены одновременно с сохранением работоспособности других частей кластера (pile). Pile может сохранять работоспособность при отключении других узлов кластера. Pile используются в режиме bridge для разделения кластера на несколько частей, между которыми осуществляется синхронная репликация. Pile может состоять из узлов одного или нескольких регионов.

Режим bridge

Режим bridge — это особая топология кластера, в рамках которой данные хранятся с синхронной репликацией между несколькими pile. Особенности режима описаны в Режим bridge, а также в Режим работы кластера Bridge.

Таблица

Таблица или **table** — это структурированный фрагмент информации, организованный в строки и колонки. Каждая строка представляет собой одну запись или элемент, а каждая колонка — это конкретный атрибут или поле с определённым типом данных.

Существуют два основных подхода к представлению табличных данных в оперативной памяти или на дисках: строковый (строка за строкой) и колоночный (колонка за колонкой). Выбранный подход сильно влияет на характеристики производительности операций с этими данными: первый больше подходит для транзакционных нагрузок (OLTP), а второй — для аналитических (OLAP). YDB поддерживает оба подхода.

Строковая таблица

Строковые таблицы или **row-oriented tables** хранят данные для всех или большинства колонок каждой строки физически рядом друг с другом. Они описаны более подробно в Строковые таблицы.

Колоночная таблица

Колоночные таблицы, **column-oriented table** или **columnar table** хранят данные для каждой колонки отдельно. Они оптимизированы для построения агрегатов по небольшому числу колонок, но менее подходят для доступа к конкретным строкам, так как строки нужно восстанавливать из их ячеек на лету. Они описаны более подробно в Колоночные таблицы.

Первичный ключ

Первичный ключ или **primary key** — это упорядоченный список колонок, значения которых однозначно идентифицируют строку. Он используется для создания первичного индекса таблицы. Он задаётся пользователем YDB при создании таблицы и существенно влияет на производительность операций с этой таблицей.

Руководство по выбору первичных ключей представлено в Выбор первичного ключа.

Первичный индекс

Первичный индекс, **индекс первичного ключа**, **primary index** или **primary key index** — это основная структура данных, используемая для нахождения строк в таблице. Он создаётся на основе выбранного первичного ключа и определяет физический порядок строк в таблице; таким образом, у каждой таблицы может быть только один первичный индекс. Первичный индекс является уникальным.

Вторичный индекс

Вторичный индекс или **secondary index** — это дополнительная структура данных, используемая для нахождения строк в таблице, обычно когда это нельзя эффективно сделать с помощью первичного индекса. В отличие от первичного индекса, вторичные индексы управляются независимо от основных данных таблицы. Таким образом, у таблицы может быть несколько вторичных индексов для различных сценариев. Возможности YDB в отношении вторичных индексов описаны в отдельной статье Вторичные индексы. Вторичный индекс может быть как уникальным, так и неуникальным.

Отдельно выделяются специальные виды вторичных индексов - векторный индекс, полнотекстовый индекс и JSON-индекс.

Векторный индекс

Векторный индекс или **vector index** — это дополнительная структура данных, используемая для ускорения решения задачи векторного поиска, когда данных достаточно много и точный векторный поиск без индекса не работает удовлетворительно. Возможности YDB по приближённому поиску ближайших соседей (ANN search) с помощью векторных индексов описаны в отдельной статье Векторные индексы.

Векторный индекс — это специализированный тип вторичного индекса, предназначенный для поиска на основе схожести, в отличие от традиционных вторичных индексов, оптимизированных для поиска по равенству или диапазону.

Полнотекстовый индекс

Полнотекстовый индекс — это дополнительная структура данных, используемая для ускорения текстового поиска по колонке таблицы (по словам и их сочетаниям, а при использовании N-грамм — и по подстрокам).

Возможности полнотекстового поиска и параметры индекса описаны в статьях Полнотекстовые индексы и Полнотекстовый поиск.

JSON-индекс

JSON-индекс или **JSON index** — это дополнительная структура данных, используемая для ускорения предикатов с функциями JSON_EXISTS и JSON_VALUE по колонке типа `Json` или `JsonDocument`. В отличие от традиционных вторичных индексов, оптимизированных для поиска по равенству или диапазону отдельных колонок таблицы, JSON-индекс работает с произвольными путями JsonPath внутри JSON-документа.

JSON-индекс, как и полнотекстовый индекс, реализован поверх инвертированного индекса, но использует собственный токенизатор JSON-документов. Возможности поиска по JSON описаны в статьях JSON-индексы и Поиск по содержимому JSON-документов.

Локальный индекс

Локальный индекс — вспомогательная структура, которая хранится вместе с данными таблицы (в отличие от глобального вторичного индекса, материализующего отдельную индексную таблицу). Локальный индекс применяется при чтении основной таблицы на стороне хранилища. Подробнее: локальные индексы.

Фильтр Блума

Фильтр Блума — вероятностная структура данных, которая позволяет быстро проверять принадлежность элемента множеству. Возможны ложноположительные срабатывания, но ложноотрицательных нет.

Локальный блум-индекс

Локальный Блум-индекс — частный случай локального индекса: вероятностный фильтр по значениям колонок на основе фильтра Блума, ускоряющий селективные запросы за счёт пропуска фрагментов данных, в которых искомое значение гарантированно отсутствует. Подробнее: Блум-индексы, локальные индексы.

Локальный min_max-индекс

Локальный min_max-индекс — частный случай локального индекса: фильтр по диапазонам значений, который хранит минимальное и максимальное значение одной колонки для каждого фрагмента данных и пропускает фрагменты, диапазон которых не может удовлетворить предикату запроса. Подробнее: min_max-индекс, локальные индексы.

Семейство колонок

Семейство колонок, **группа колонок**, **column family** или **column group** — это функция, позволяющая хранить подмножества колонок строковой таблицы отдельно в отдельном семействе или группе. Основной сценарий использования — хранение части колонок на других типах дисков (перенос менее важных колонок на HDD) или с другими настройками компрессии. Если рабочая нагрузка требует многих семейств колонок, рассмотрите возможность использования колоночных таблиц.

Кодирование колонок

Кодирование колонок или **column encoding** — это механизм оптимизации хранения данных в колонках таблицы, который позволяет уменьшить объем занимаемого места на диске и ускорить выполнение некоторых операций.

Время жизни

Время жизни, **time to live** или **TTL** — это механизм для автоматического удаления старых строк из таблицы асинхронно в фоновом режиме. Он описан в отдельной статье Time to Live (TTL) и вытеснение данных во внешнее хранилище.

Представление

Представление или **view** — это способ сохранить запрос и обращаться к его результатам как к настоящей таблице. Само представление не хранит данных, кроме текста запроса. Запрос, хранящийся в представлении, выполняется при каждом SELECT из него, генерируя возвращаемый результат. Любые изменения в таблицах, на которые ссылается представление, немедленно отражаются в результатах чтения из него.

Представления бывают пользовательские и системные.

Пользовательские представления

Пользовательские представления создаются пользователем с помощью команды CREATE VIEW. Они описаны более подробно в Представления (VIEW).

Системные представления

Системные представления, **system views** — это специальные представления, автоматически создаваемые системой для мониторинга состояния базы данных и кластера. Они располагаются в специальной директории `.sys`, находящейся в корневой папке каждой базы данных. Системные представления для баз данных описаны в Системные представления базы данных; системные представления для кластера, а также вопросы управления доступом к ним — в Системные представления кластера.

Топик

Очередь сообщений используется для надёжной асинхронной связи между различными системами посредством передачи сообщений. YDB предоставляет инфраструктуру, обеспечивающую семантику "exactly once" (ровно один раз) в таких коммуникациях. С её использованием можно добиться гарантии отсутствия потерянных сообщений и случайных дубликатов.

Топик, **тема**, **topic** — это именованная сущность в очереди сообщений, предназначенная для взаимодействия писателей и читателей.

Несколько терминов, связанных с топиками, приведены ниже. Как работают топики YDB объясняется более подробно в отдельной статье Топик.

Партиция

Для целей горизонтального масштабирования топики разделяются на отдельные элементы, называемые **партициями** или **partitions**. Таким образом, партиции являются единицей параллелизма внутри топика. Сообщения внутри каждой партиции упорядочены.

Однако подмножества данных, управляемые одним data shard или column shard, также могут называться партициями.

Смещение

Смещение или **offset** — это порядковый номер, идентифицирующий сообщение внутри партиции.

Писатель

Писатель, **производитель** или **producer** — это сущность, записывающая новые сообщения в топик.

Читатель

Читатель, **потребитель** или **consumer** — это сущность, читающая сообщения из топика.

Выполнение запроса

Ниже приведены термины, описывающие план выполнения запроса и распределённое выполнение его стадий.

Оператор

Оператор — элемент плана выполнения запроса, который описывает действие над данными (чтение таблицы, фильтрация, соединение, агрегация и т. п.).

Логический оператор

Логический оператор описывает, **что** нужно сделать с данными, без указания конкретного способа выполнения на кластере. Логические операторы формируют логический план на раннем этапе компиляции.

Физический оператор

Физический оператор описывает, **что** и **как** выполняется на кластере: выбранный алгоритм (например, `TableFullScan` или `InnerJoin (Map)`), распределение работы по стадиям обработки и участие в обмене данными между узлами.

Каналы связи

Каналы связи — элементы плана выполнения запроса, обеспечивающие передачу данных между стадиями обработки. Состоят из выходного канала (передаёт данные из одной стадии обработки) и входного канала (принимает данные в другую стадию обработки). Каналы связи используются для передачи данных между вычислительными стадиями, а также между стадиями хранения и вычислительными стадиями — при этом для разных типов соединений могут применяться различные технические механизмы (например, hash shuffle или операция merge).

Стадии обработки

Стадии обработки — неделимые блоки обработки данных в плане выполнения запроса, на которые разбивается SQL-запрос в процессе компиляции. Каждая стадия может включать несколько параллельно выполняемых задач (в том числе на разных узлах) и содержит логические и физические операторы; стадии бывают двух типов: те, что выполняются непосредственно в системе хранения данных (например, в колоночных таблицах или шардах), и те, что только обрабатывают данные (не хранят их). Стадии последовательно обрабатывают данные — поток идёт от хранилища данных к результату выполнения запроса, образуя в общем случае ориентированный ациклический граф.

Задача

Задача (task) — единица распределённого выполнения внутри стадии обработки: один поток вычислений на узле кластера, обрабатывающий часть данных этой стадии.

Захват изменений данных

Захват изменений данных, **change data capture** или **CDC** — это механизм, позволяющий подписываться на **поток изменений** в конкретной таблице. Технически он реализован поверх топиков. Он описан более подробно в отдельной статье Change Data Capture (CDC).

Поток изменений

Поток изменений — упорядоченный список изменений таблицы, помещенный в топик.

Коллекция резервных копий

Коллекция резервных копий — это схемный объект, который организует полные и инкрементальные резервные копии для выбранных строковых таблиц. Коллекции обеспечивают восстановление на момент времени, поддерживая цепочки резервных копий и гарантируя согласованное восстановление нескольких таблиц. Таблица может принадлежать только одной коллекции резервных копий одновременно.

Подробнее см. Коллекция резервных копий.

Резервная копия

Резервная копия — это копия данных на определённый момент времени, которая может использоваться для восстановления данных. В контексте коллекций резервных копий существует два типа:

- **Полная резервная копия**: Полный снимок всех данных в коллекции. Служит основой для цепочек резервных копий и может быть восстановлена независимо.
- **Инкрементальная резервная копия**: Захватывает только изменения (вставки, обновления, удаления) с момента предыдущей резервной копии. Требует всей цепочки резервных копий для восстановления.

Цепочка резервных копий

Цепочка резервных копий — упорядоченная последовательность резервных копий, начинающаяся с полной резервной копии, за которой следует ноль или более инкрементальных резервных копий. Каждая инкрементальная резервная копия зависит от всех предыдущих резервных копий в цепочке. Удаление любой резервной копии в цепочке делает последующие инкрементальные резервные копии невосстановимыми.

Экземпляр асинхронной репликации

Экземпляр асинхронной репликации — это именованная сущность, хранящая настройки асинхронной репликации (настройки подключения, список реплицируемых объектов и т.д.). Также с его помощью можно получить информацию о состоянии асинхронной репликации: прогресс первоначального сканирования, отставание, ошибки и т.д.

Реплицируемый объект

Реплицируемый объект — объект (например, таблица), для которого настроена асинхронная репликация.

Объект-реплика

Объект-реплика — «зеркальная копия» реплицируемого объекта, автоматически созданная экземпляром асинхронной репликации. Как правило, доступен только для чтения.

Экземпляр трансфера

Экземпляр трансфера — это именованная сущность, хранящая настройки трансфера, включая настройки подключения и правила преобразования данных. Также с его помощью можно получить информацию о состоянии трансфера, например ошибки.

Узел координации

Узел координации или **coordination node** — это объект схемы, который позволяет клиентским приложениям создавать семафоры для координации своих действий. Узлы координации применяются для реализации распределённых блокировок, обнаружения сервисов, выбора лидера и других сценариев. Более подробно об узлах координации.

Семафор

Семафор — это объект внутри узла координации, который предоставляет механизм синхронизации для распределённых приложений. Семафоры могут быть постоянными или временными и поддерживают операции создания, захвата, освобождения и мониторинга. Более подробно о семафорах в YDB.

Пул ресурсов

Пул ресурсов или **resource pool** — объект схемы, который описывает ограничения, накладываемые на ресурсы (CPU, RAM и т.п.) доступные для выполнения запросов в этом пуле ресурсов. Запрос всегда выполняется в каком-то пуле ресурсов. По умолчанию все запросы выполняются в пуле ресурсов с именем `default`, который не накладывает каких-либо ограничений. Подробнее об использовании пулов ресурсов можно узнать в статье Workload Manager — управление потреблением ресурсов.

Классификатор пулов ресурсов

Классификатор пулов ресурсов или **resource pool classifier** — объект, предназначенный для управления распределением запросов между пулами ресурсов. Он описывает правила, по которым для каждого запроса выбирается пул ресурсов. Эти классификаторы глобальны для всей базы данных и применяются ко всем запросам, поступающим в неё. Подробнее об их использовании можно узнать в статье Workload Manager — управление потреблением ресурсов.

YQL

YQL (YDB Query Language) — это высокоуровневый язык для работы с системой. Он является диалектом ANSI SQL. Существует много материалов, посвящённых YQL, включая туториал, справочник и рецепты.

Федеративные запросы

Федеративные запросы или **federated queries** — это функциональность, позволяющая выполнять запросы к данным, хранящимся в системах, внешних по отношению к кластеру YDB.

Ниже объяснены несколько терминов, связанных с федеративными запросами. Как работают федеративные запросы в YDB объясняется более подробно в отдельной статье Федеративные запросы.

Внешний источник данных

Внешний источник данных, **внешнее подключение**, **external data source** или **external connection** — это метаданные, описывающие, как подключиться к поддерживаемой внешней системе для выполнения федеративных запросов.

Внешняя таблица

Внешняя таблица или **external table** — это метаданные, описывающие конкретный набор данных, который можно извлечь из внешнего источника данных.

Секрет

Секрет или **secret** — это конфиденциальные метаданные, требующие особого обращения. Например, секреты могут использоваться в определениях внешних источников данных и представляют собой такие сущности, как пароли и токены.

Аутентификационный токен

Аутентификационный токен или **auth token** — токен, используемый для аутентификации в YDB.

YDB поддерживает разные виды аутентификации и различные типы токенов.

Токен пользователя

Токен пользователя — внутренний результат проверки аутентификационного токена. Он содержит SID пользователя и групп и используется при авторизации. Токен пользователя используется и хранится только на создавшем его узле и не передаётся клиенту. Подробнее см. в статье Кеширование результатов аутентификации.

mTLS

mTLS (mutual TLS, взаимный TLS) — режим TLS, при котором не только клиент проверяет сертификат сервера, но и сервер запрашивает и проверяет клиентский сертификат при установлении TLS-соединения.

Клиентский сертификат

Клиентский сертификат (или **сертификат клиента**) — цифровой сертификат, выдаваемый и используемый клиентом — приложением, пользователем или узлом YDB — для аутентификации устройств на этапе TLS-соединения и для аутентификации клиентов по сертификату на уровне запросов.

Схема кластера

Схема кластера YDB — это иерархическое пространство имён кластера YDB. Верхнеуровневым элементом этого пространства имён является корень схемы кластера. Дочерними элементами корня схемы кластера выступают базы данных. Внутри каждой базы данных можно создавать произвольную иерархию объектов (таблицы, топики и т.п.) с помощью вложенных директорий.

Схема базы данных

Схема базы данных — это подмножество иерархического пространства имён кластера, принадлежащее базе данных.

Корень базы данных

Корень базы данных — это путь к базе данных в схеме кластера.

Корень схемы

Корень схемы кластера — это корневой элемент пространства имён YDB, дочерними элементами которого выступают базы данных.

Объект схемы

Схема базы данных состоит из **схемных объектов** или **объектов схемы**, которыми могут быть базы данных, таблицы (включая внешние таблицы), топики, папки и т.д.

Для удобства организации, схемные объекты образуют иерархию с помощью папок.

Папка

Как и в файловых системах, **папка**, **каталог**, **folder** или **directory** является контейнером для схемных объектов.

Папки могут содержать подпапки и такая вложенность может быть произвольной глубины.

Объект доступа

Объект доступа или **access object** при авторизации — сущность, для которой настраиваются права и ограничения доступа. В YDB объектами доступа являются объекты схемы.

У каждого объекта схемы есть владелец и список прав на этот объект, выданных пользователям и группам (субъектам доступа).

Субъект доступа

Субъект доступа или **access subject** — сущность, которая может обращаться к объектам доступа и выполнять определённые действия в системе.

Получение доступа при этих обращениях и действиях зависит от настроенных списков прав и уровня доступа субъекта.

Субъектом доступа может быть пользователь или группа.

Право доступа

Право доступа или **access right** — сущность, отражающая разрешение субъекту доступа выполнять конкретный набор операций в кластере или базе данных над конкретным объектом доступа.

Наследование прав доступа

Наследование прав доступа — механизм, при котором права доступа, выданные на родительские объекты доступа, наследуются дочерними объектами в иерархической структуре базы данных. Это гарантирует, что разрешения, предоставленные на более высоком уровне иерархии, применяются ко всем нижестоящим уровням, если они не переопределены явно.

Список прав

Список прав, **access control list** или **ACL** — список всех прав, предоставленных субъектам доступа (пользователям и группам) на конкретный объект доступа.

Уровень доступа

Уровень доступа предоставляет субъекту доступа дополнительные возможности при работе с объектами схемы, а также возможность выполнения операций над кластером в целом. YDB использует иерархические уровни доступа:

- Database;
- Viewer;
- Monitoring;
- Administration.

Уровень доступа для субъекта настраивается с помощью списков уровня доступа.

Список уровня доступа

Список уровня доступа или **список разрешений** — список SID'ов субъектов доступа, которым разрешён определённый уровень доступа.

В YDB существует несколько таких списков, которые определяют, кто и какими уровнями доступа обладает.

Подробная информация о списках уровней доступа, их иерархии и принципах работы приведена в разделе Списки уровней доступа документации по авторизации.

Владелец

Владелец - субъект доступа (пользователь или группа) имеющий полный объем прав на конкретный объект доступа.

Пользователь

Пользователь - лицо, использующее YDB для выполнения конкретной функции.

В YDB существуют разные виды пользователей в зависимости от способа создания:

- локальные пользователи в базах данных YDB;
- внешние пользователи из сторонних каталогов.

Пользователь идентифицируется SID.

Локальный пользователь

Пользователь, чья учётная запись создана непосредственно в YDB с использованием YQL-команды `CREATE USER` или при начальной конфигурации безопасности.

Внешний пользователь

Пользователь YDB, чья учётная запись создана в стороннем каталоге, например в LDAP-каталоге или IAM-системе.

Группа

Группа или **группа доступа** - именованное множество пользователей и других групп с равными возможностями для своих членов.

Группа идентифицируется SID.

Роль

Роль — это именованное множество прав доступа, используемое для назначения пользователям или группам пользователей.

Роли в YDB реализованы с помощью групп, которые создаются при первоначальном развёртывании кластера, и которым назначается определённый список прав на корне схемы кластера. Подробнее о ролях читайте в статье Начальная настройка безопасности кластера.

SID

SID или **security identifier** — строка вида `<name>` или `<name>@<auth-domain>`, идентифицирующая субъект доступа. Используется при аутентификации, авторизации, в списках прав и в списках уровней доступа.

SID идентифицирует индивидуального пользователя или группу пользователей.

Опциональный суффикс `@<auth-domain>` идентифицирует источник субъекта доступа, то есть внешний каталог или систему, откуда он был получен. Например, пользователи или группы из LDAP-каталога могут иметь суффикс `@ldap`. Отсутствие суффикса означает, что пользователь или группа созданы и существуют непосредственно в YDB.

Оптимизатор запросов

Оптимизатор запросов — набор компонентов YDB, отвечающих за преобразование логического представления запроса в конкретный физически исполнимый план получения запрошенного результата. Основная цель оптимизатора — выбрать среди всех возможных планов выполнения запроса достаточно эффективный с точки зрения прогнозируемого времени исполнения и потребления ресурсов кластера. Он описан более подробно в отдельной статье Оптимизация запросов в YDB.

Кардинальность

Кардинальность — оценка оптимизатора запросов числа строк (записей) на выходе оператора плана выполнения запроса или в результате соединения указанных таблиц.

Кэш компиляции

Кэш компиляции или **compile cache** — кэш скомпилированных запросов на каждой ноде кластера. Используется для избежания повторной компиляции: если текст запроса уже есть в кэше ноды, дополнительная компиляция не выполняется. Подробнее см. в разделе Кэш компиляции запросов.

Продвинутая терминология

Этот раздел объясняет термины, которые полезны для YDB контрибьюторов и пользователей, желающих глубже понять, что происходит внутри системы.

Реализация акторов

Система акторов

Система акторов или **actor system** — это C++ библиотека с реализацией акторной модели для нужд YDB.

Акторный сервис

Акторный сервис или **actor service** — это актор, который имеет известное имя и обычно выполняется в единственном экземпляре на узле.

ActorId

ActorId — это уникальный идентификатор актора или таблетки в кластере.

Интерконнект акторной системы

Интерконнект акторной системы, **интерконнект**, **actor system interconnect** или **interconnect** — это внутренний сетевой слой кластера. Все акторы взаимодействуют друг с другом в системе через интерконнект.

Локал

Локал или **local** — это акторный сервис, работающий на каждом узле. Он напрямую управляет таблетками на своём узле и взаимодействует с Hive. Он регистрируется в Hive и получает команды на запуск таблеток.

Реализация таблеток

Таблетка или **tablet** — это актор с постоянным состоянием. Она включает набор данных, за которые эта таблетка отвечает, и конечный автомат, через который изменяются данные (или состояние) таблетки. Таблетка является отказоустойчивой сущностью, поскольку данные таблетки хранятся в распределённом хранилище, которое переживает сбои дисков и узлов. Таблетка автоматически перезапускается на другом узле в случае отказа или перегрузки предыдущего. Данные в таблетке изменяются последовательно, так как инфраструктура системы гарантирует, что нет более одного лидера таблетки, через которого выполняются изменения данных таблетки.

Таблетка решает ту же задачу, что и алгоритмы Paxos и Raft в других системах, а именно задачу распределённого консенсуса. С технической точки зрения реализация таблетки может быть описана как реплицированная машина состояний (Replicated State Machine, RSM) поверх общего журнала, поскольку состояние таблетки полностью описывается упорядоченным журналом команд, хранящимся в распределённом и отказоустойчивом хранилище.

Во время выполнения машина состояний таблетки управляется тремя компонентами:

1. Общая таблеточная часть обеспечивает согласованность журнала и восстановление в случае сбоев.
2. **Исполнитель** или **executor** — это абстракция локальной базы данных, а именно структуры данных и код, которые организуют работу с данными, хранимыми таблеткой.
3. Актор с пользовательским кодом, реализующим специфическую логику конкретного типа таблетки.

В YDB существует несколько видов специализированных таблеток, хранящих всевозможные данные для различных задач. Многие функции YDB, такие как таблицы и топики, реализованы как разные виды таблеток. Таким образом, повторное использование инфраструктуры таблеток является одним из ключевых средств расширяемости YDB как платформы.

Обычно в кластере YDB работает на порядки больше таблеток по сравнению с процессами или потоками, которые другие системы использовали бы для кластера аналогичного размера. В кластере YDB легко могут одновременно работать сотни тысяч и миллионы таблеток.

Поскольку таблетка хранит своё состояние в распределённом хранилище, она может быть (пере-)запущена на любом узле кластера. Таблетки идентифицируются с помощью TabletID, 64-битного числа, назначаемого при создании таблетки.

Лидер таблетки

Лидер таблетки или **tablet leader** — это текущий активный лидер данной таблетки. Лидер таблетки принимает команды, назначает им порядок и подтверждает их внешнему миру. Гарантируется, что в любой момент времени существует не более одного лидера для каждой таблетки.

Кандидат таблетки

Кандидат таблетки или **tablet candidate** — это один из участников выборов, который хочет стать лидером данной таблетки. Если кандидат выигрывает выборы, он становится лидером таблетки.

Реплика таблетки

Реплика таблетки, **горячий резерв**, **tablet follower** или **hot standby** — это копия лидера таблетки, которая применяет журнал команд, принятых лидером (с некоторой задержкой). У таблетки может быть ноль или больше реплик. Реплики выполняют две основные функции:

- В случае завершения или отказа лидера, реплики являются предпочтительными кандидатами на роль нового лидера, так как они могут стать лидером намного быстрее, чем другие кандидаты, поскольку они применили большую часть журнала.
- Реплики могут отвечать на запросы только для чтения, если клиент явно выбирает опциональный расслабленный режим транзакций, допускающий устаревшие чтения (stale reads).

Поколение таблетки

Поколение таблетки или **tablet generation** — это номер, идентифицирующий реинкарнацию лидера таблетки. Он меняется только при выборе нового лидера и всегда увеличивается.

Локальная база данных таблетки

Локальная база данных таблетки, **локальная база данных**, **tablet local database** или **local database** — это набор структур данных и связанного кода, которые управляют состоянием таблетки и хранимыми ей данными. Логически состояние локальной базы данных представлено набором таблиц, очень похожих на реляционные таблицы. Модификация состояния локальной базы данных осуществляется локальными транзакциями таблетки, создаваемыми пользовательским актором таблетки.

Каждая таблица локальной базы данных хранится как LSM-дерево.

Log-structured merge-tree

Log-structured merge-tree или **LSM-дерево** — это структура данных, разработанная для оптимизации производительности записи и чтения в системах хранения. Она используется в YDB для хранения таблиц локальной базы и данных VDisks.

MemTable

Все данные, записанные в таблицы локальной базы, изначально хранятся в структуре данных в оперативной памяти, называемой **MemTable**. Когда MemTable достигает заданного размера, она сбрасывается на диск как неизменяемая структура данных SST.

Sorted string table

Sorted string table или **SST** — это неизменяемая структура данных, которая хранит строки таблицы, отсортированные по ключу, что облегчает эффективный поиск ключей и работу с диапазонами ключей. Каждая SST состоит из непрерывной серии маленьких страниц данных, обычно размером около 7 КиБ каждая, что дополнительно оптимизирует процесс чтения данных с диска. Обычно SST представляет собой часть LSM-дерева.

Пайп таблетки

Пайп таблетки, **tablet pipe** или **TabletPipe** — это виртуальное соединение, которое может быть установлено с таблеткой. Оно включает поиск лидера таблетки по TabletID. Это рекомендуемый способ работы с таблеткой. Термин **открыть пайп к таблетке** описывает процесс разрешения (поиска) таблетки в кластере и установления с ней виртуального канала связи.

TabletID

TabletID — это уникальный идентификатор таблетки в рамках кластера.

Bootstrapper

Bootstrapper — это основной механизм запуска таблеток, используемый для системных таблеток (например, для Hive, DS controller, корневого SchemeShard). Hive инициализирует остальные таблетки.

Общий кеш

Общий кеш или **shared cache** — это актор, который хранит страницы данных, недавно прочитанные из распределённого хранилища. Кеширование этих страниц снижает количество операций ввода-вывода с диска и ускоряет получение данных, повышая общую производительность системы.

Контроллер памяти

Контроллер памяти или **memory controller**— это актор, который управляет лимитами памяти YDB.

Спиллинг

Спиллинг или **spilling** — это механизм управления памятью в YDB, который временно выгружает промежуточные данные запросов во внешнее хранилище, когда такие данные превышают доступный объём оперативной памяти узла. В YDB для спиллинга в настоящее время используется диск.

Подробнее о спиллинге см. в Спиллинг.

Типы таблеток

Таблетки можно рассматривать как фреймворк для создания надёжных компонентов, работающих в распределённой системе. Многие компоненты YDB — как системные, так и работающие с пользовательскими данными — реализованы с использованием этого фреймворка, основные из них перечислены ниже.

SchemeShard

SchemeShard или **Scheme shard** — это системная таблетка, которая хранит схему базы данных, включая метаданные пользовательских таблиц, топиков и т.д.

Кроме того, существует **корневой SchemeShard**, который хранит информацию о базах данных, созданных в кластере.

DataShard

DataShard или **Data shard** — это таблетка, которая управляет сегментом строковой пользовательской таблицы. Логическая пользовательская таблица делится на сегменты по непрерывным диапазонам первичного ключа таблицы. За каждый такой диапазон отвечает отдельная таблетка DataShard. Сам диапазон также называется партицией. Таблетка DataShard хранит данные построчно, что эффективно для OLTP нагрузок.

ColumnShard

ColumnShard или **Column shard** — это таблетка, которая хранит сегмент данных колоночной пользовательской таблицы.

KeyValue Tablet

KeyValue, **KV Tablet** или **таблетка ключ-значение** — это таблетка, которая реализует простое отображение ключ → значение, где ключи и значения — это строки. У неё также есть несколько специфических функций, таких как блокировки.

PersQueue Tablet

PersQueue или **persistent queue tablet** — это таблетка, которая реализует концепцию топика. Каждый топик состоит из одной или нескольких партиций, и каждая партиция управляется отдельным экземпляром таблетки PQ.

TxAllocator

TxAllocator, **аллокатор транзакций** или **transaction allocator** — это системная таблетка, которая выделяет уникальные идентификаторы транзакций (TxID) в кластере. Обычно в кластере имеется несколько таких таблеток, из которых transaction proxy предварительно выделяет и кэширует диапазоны для локального выпуска в рамках одного процесса.

Coordinator

Координатор или **coordinator** — это системная таблетка, обеспечивающая глобальный порядок транзакций. Задача координатора заключается в присвоении логического времени PlanStep каждой транзакции, спланированной через этого координатора. Каждой транзакции назначается ровно один координатор, выбранный путём хеширования её TxId.

Mediator

Медиатор — это системная таблетка, которая распределяет транзакции, спланированные координаторами, между участниками транзакций. Медиаторы обеспечивают продвижение глобального времени. Каждый участник транзакции ассоциируется ровно с одним медиатором. Медиаторы позволяют избежать необходимости в полном наборе соединений между всеми координаторами и всеми участниками всех транзакций.

Hive

Hive — это системная таблетка, отвечающая за запуск и управление другими таблетками. Её обязанности включают перемещение таблеток между узлами в случае отказа или перегрузки узла. Подробнее о Hive можно узнать в отдельной статье.

CMS

CMS, **система управления кластером** или **cluster management system** — это системная таблетка, отвечающая за управление информацией о текущем состоянии кластера YDB. Эта информация используется для выполнения постепенных перезапусков кластера без воздействия на пользовательские нагрузки, обслуживания, переконфигурации кластера и т.д. В составе CMS работает Sentinel — компонент, который реализует SelfHeal.

SelfHeal

SelfHeal — механизмы автоматического поддержания и восстановления отказоустойчивости кластера. SelfHeal хранилища переносит VDisk групп хранения после длительных отказов узлов или дисков. SelfHeal подсистем распространения метаданных переносит реплики State Storage, Board и SchemeBoard после отказов и может добавлять реплики при появлении новых узлов.

NodeBroker

NodeBroker — это системная таблетка, которая отвечает за регистрацию динамических узлов в кластере.

BSController

BSController, **контроллер распределённого хранилища**, **blob storage controller**, **BS-контроллер** управляет динамической конфигурацией распределённого хранилища, включая информацию о PDisk, VDisk и группах хранения. Он взаимодействует с node warden для запуска различных компонентов распределённого хранилища. Он взаимодействует с Hive для выделения каналов таблеткам.

Console

Console — это системная таблетка, отвечающая за хранение динамической конфигурации и её доставку на узлы кластера.

Kesus

Kesus — это таблетка, которая реализует узел координации.

SysViewProcessor

SysViewProcessor — это таблетка, которая хранит данные некоторых из системных представлений.

SequenceShard

SequenceShard — это таблетка, обслуживающая объекты Sequence, которые используются для реализации серийных типов данных.

ReplicationController

ReplicationController — это таблетка, отвечающая за процесс асинхронной репликации.

StatisticsAggregator

StatisticsAggregator — это таблетка, отвечающая за сбор статистики, используемой в стоимостной оптимизации.

Слот

Слот в YDB может использоваться в двух контекстах:

- **Слот** — это часть ресурсов сервера, выделенная для запуска одного узла YDB. Обычный размер слота — 10 процессорных ядер и 50 ГБ оперативной памяти. Слоты используются, если кластер YDB развернут на серверах или виртуальных машинах с достаточными ресурсами для размещения нескольких слотов.
- **Слот VDisk** или **VSlot** — это доля PDisk, которая может быть выделена одному из VDisk.

Хранилище состояния

Хранилище состояния, **state storage** или **StateStorage** — это распределённый сервис, который хранит информацию о таблетках, а именно:

- Текущий лидер таблетки или его отсутствие.
- Реплики таблетки.
- Поколение и шаг таблетки `(generation:step)`.

Хранилище состояния используется как служба для разрешения имён таблеток, то есть для получения ActorId по TabletID. StateStorage также используется в процессе выбора лидера таблетки.

Информация в хранилище состояния является волатильной. Таким образом, она теряется при отключении питания или перезапуске процесса. Несмотря на название, этот сервис не является постоянным долгосрочным хранилищем. Он содержит только информацию, которую легко восстановить и которая не должна быть долговечной. Однако хранилище состояния хранит информацию на нескольких узлах, чтобы минимизировать влияние отказов узлов. Через этот сервис также можно собрать кворум, который используется для выбора лидеров таблеток.

Из-за своей природы сервис хранилища состояния работает по принципу максимально возможных усилий. Например, отсутствие нескольких лидеров таблеток гарантируется через протокол выбора лидеров на распределённом хранилище, а не на хранилище состояния.

Подробнее об устройстве StateStorage и связанных подсистем — в разделе Сервисы распространения метаданных.

Board

Board — распределённый сервис, предназначенный для хранения метаданных в виде пар ключ-значение. Используется, в том числе, для хранения информации об endpoint-ах.

Подробнее об устройстве Board и связанных подсистем — в разделе Сервисы распространения метаданных.

SchemeBoard

SchemeBoard — распределённый сервис, предназначенный для хранения метаданных в виде пар ключ-значение. Используется, в том числе, для хранения информации о схемах.

Подробнее об устройстве SchemeBoard и связанных подсистем — в разделе Сервисы распространения метаданных.

Компакшн

Компакшн, **компактизация** или **compaction** — это внутренний фоновый процесс перестройки данных LSM-дерева. Данные в VDisk и локальных базах данных организованы в виде LSM-деревьев. Поэтому различают **компакшн VDisk** и **таблеточный компакшн**. Процесс компакшна обычно довольно ресурсоёмкий, поэтому принимаются меры по минимизации накладных расходов, связанных с ним, например, путём ограничения числа одновременно исполняемых компакшнов.

gRPC-прокси

gRPC-прокси или **gRPC proxy** — это прокси-система для внешних пользовательских запросов. Клиентские запросы поступают в систему по протоколу gRPC, затем компонент прокси переводит их во внутренние вызовы для выполнения этих запросов, передаваемых через интерконнект. Этот прокси предоставляет интерфейс как для запросов-ответов, так и для двунаправленной потоковой передачи данных.

Распредёленная конфигурация

Распределённая конфигурация, **Distributed configuration** или **DistConf** - это внутренний механизм конфигурации кластера, который обеспечивает запуск и настройку статических узлов, автоматическое управление статической группой хранения и State Storage. Распределённая конфигурация запускается до старта любых таблеток, групп хранения и State Storage.

Подробнее об устройстве распределённой конфигурации рассказано в Устройство механизма конфигурации V2.

Реализация распределённого хранилища

Распределённое хранилище — это распределённый отказоустойчивый слой хранения данных, который сохраняет бинарные записи, называемые LogoBlob, адресуемые с помощью определённого типа идентификатора, называемого LogoBlobID. Таким образом, распределённое хранилище является хранилищем ключ-значение, которое сопоставляет LogoBlobID строке размером до 10 МБ. Распределённое хранилище состоит из множества групп хранения, каждая из которых является независимым репозиторием данных.

Распределённое хранилище сохраняет неизменяемые данные, при этом каждый неизменяемый блок данных идентифицируется определённым ключом LogoBlobID. API распределённого хранилища очень специфичен, предназначен только для использования таблетками для хранения их данных и журналов изменений. Таким образом, оно не предназначено для хранения данных общего назначения. Данные в распределённом хранилище удаляются с помощью специальных команд барьера. Из-за отсутствия мутаций в его интерфейсе распределённое хранилище может быть реализовано без реализации распределённого консенсуса. Распределённое хранилище является лишь одним из компонентов, который таблетки используют для реализации распределённого консенсуса.

LogoBlob

LogoBlob — это набор двоичных неизменяемых данных, идентифицируемый LogoBlobID и хранящийся в распределённом хранилище. Размер блока данных ограничен на уровне VDisk и выше по стэку. В настоящее время максимальный размер блока данных, который могут обработать VDisk, составляет 10 МБ.

LogoBlobID

LogoBlobID — это идентификатор LogoBlob в распределённом хранилище. Он имеет структуру вида `[TabletID, Generation, Step, Channel, Cookie, BlobSize, PartID]`. Основные элементы LogoBlobID:

- `TabletID` — это ID таблетки, которой принадлежит LogoBlob.
- `Generation` — это поколение таблетки, в котором был записан блок данных.
- `Channel` — это канал таблетки, на котором записан LogoBlob.

- `Step` — это инкрементный счётчик, обычно в пределах поколения таблетки.
- `Cookie` — это уникальный идентификатор блока данных в пределах одного `Step`. Cookie обычно используется при записи нескольких блоков данных в один `Step`.
- `BlobSize` — это размер LogoBlob.
- `PartID` — это идентификатор части блока данных. Он важен, когда оригинальный LogoBlob разбивается на части с использованием кодирования с исправлением ошибок, и части записываются на соответствующие VDisk и группы хранения.

Репликация

Репликация — это процесс, обеспечивающий наличие достаточного количества копий (реплик) данных для поддержания желаемых характеристик доступности кластера YDB. Обычно используется в геораспределённых кластерах YDB.

Кодирование с исправлением ошибок

Кодирование с исправлением ошибок или **erasure coding** — это метод кодирования данных, при котором исходные данные дополняются избыточностью и разделяются на несколько фрагментов, обеспечивая возможность восстановления исходных данных в случае потери одного или нескольких фрагментов. Он широко используется в кластерах YDB с одной зоной доступности в отличие от репликации с 3 репликами. Например, наиболее популярная схема erasure coding 4+2 обеспечивает ту же надёжность, что и три реплики, с избыточностью пространства 1.5 против 3.

PDisk

PDisk, **physical disk** или **физический диск** — это компонент, который контролирует физический дисковый накопитель (блочное устройство). Другими словами, PDisk — это подсистема, которая реализует абстракцию, аналогичную специализированной файловой системе поверх блочных устройств (или файлов, эмулирующих блочное устройство для целей тестирования). PDisk обеспечивает контроль целостности данных (включая кодирование с исправлением ошибок групп секторов для восстановления данных на отдельных повреждённых секторах, контроль целостности с помощью контрольных сумм), прозрачное шифрование всех данных на диске и транзакционные гарантии дисковых операций (подтверждение записи строго после `fsync`).

PDisk содержит планировщик, который обеспечивает совместное использование пропускной способности устройства между несколькими клиентами (VDisk). PDisk делит блочное устройство на блоки, называемые слотами (размером около 128 мегабайт; разрешены и меньшие блоки). В каждый момент времени не более одного VDisk может владеть каждым слотом. PDisk также поддерживает журнал восстановления, общий для записей службы PDisk и всех VDisk.

VDisk

VDisk, **virtual disk** или **виртуальный диск** — это компонент, который реализует хранение данных распределённого хранилища LogoBlob на PDisk. VDisk хранит все свои данные на PDisk. Один VDisk соответствует одному PDisk, но обычно несколько VDisk связаны с одним PDisk. В отличие от PDisk, который скрывает блоки и журналы за собой, VDisk предоставляет интерфейс на уровне LogoBlob и LogoBlobID, например запись LogoBlob, чтение данных LogoBlobID и удаление набора LogoBlob с помощью специальной команды. VDisk является членом группы хранения. Сам VDisk является локальным, но многие VDisk в данной группе обеспечивают надёжное хранение данных. VDisk в группе синхронизируют данные друг с другом и реплицируют данные в случае потерь. Набор VDisk в группе хранения образует распределённый RAID.

Yard

Yard — это название API PDisk. Он позволяет VDisk читать и записывать данные в блоки и журналы, резервировать блоки, удалять блоки и транзакционно получать и возвращать владение блоками. В некоторых контекстах Yard можно считать синонимом PDisk.

Skeleton

Skeleton — это актор, который предоставляет интерфейс к VDisk.

SkeletonFront

SkeletonFront — это прокси-актор для Skeleton, который контролирует поток сообщений, поступающих в Skeleton.

Прокси

Прокси распределённого хранилища, **DS-прокси**, **BS-прокси**, **distributed storage proxy**, **DS-proxy** или **BS-proxy** выполняет роль клиентской библиотеки для выполнения операций с распределённым хранилищем. Пользователями DS-прокси являются таблетки, которые записывают и читают из распределённого хранилища. DS-прокси скрывает распределённую природу распределённого хранилища от пользователя. Задача DS-прокси — запись в кворум VDisk, выполнение повторных попыток при необходимости и контроль потока записи/чтения для предотвращения перегрузки VDisk.

Технически DS-прокси реализован как акторный сервис, запускаемый node warden на каждом узле для каждой группы хранения, обрабатывающий все запросы к группе (запись, чтение и удаление LogoBlob, блокировка группы). При записи данных DS-прокси выполняет кодирование с исправлением ошибок данных, разделяя LogoBlob на части, которые затем отправляются соответствующим VDisk. DS-прокси выполняет обратный процесс при чтении, получая части от VDisk и восстанавливая из них LogoBlob.

Node warden

Node warden или `BS_NODE` — это сервис акторов на каждом узле кластера, запускающий PDisks, VDisks и DS прокси статических групп хранения при запуске узла. Также он взаимодействует с DS контроллером для запуска PDisk, VDisk и DS прокси динамических групп. DS прокси динамических групп запускается по запросу: node warden обрабатывает «недоставленные» сообщения к DS прокси, запускает соответствующие DS прокси и получает конфигурацию групп от DS контроллера.

Область отказа

Область отказа или **fail realm** — это набор доменов отказа, которые могут выйти из строя одновременно по общей причине. Коррелированный сбой двух VDisks в одной и той же области отказа более вероятен, чем сбой двух VDisk из разных областей отказа.

Примером области отказа является набор оборудования, размещённый в одном дата-центре, или зоне доступности, который может выйти из строя целиком вследствие природной катастрофы, масштабного отключения электропитания или другого подобного события.

Домен отказа

Домен отказа или **fail domain** — это набор оборудования, который может выйти из строя одновременно. Коррелированный сбой двух VDisk в пределах одного домена отказа более вероятен, чем сбой двух VDisk из разных доменов отказа. В случае разных доменов отказа вероятность одновременного сбоя также зависит от того, принадлежат ли рассматриваемые домены одной области отказа или разным.

Примером домена отказа является набор дисков, подключённых к одному серверу, поскольку все диски конкретного сервера могут оказаться недоступными при отказе блока питания сервера или сетевого контроллера. Обычно к общему домену отказа относят все серверы, размещённые в одной серверной стойке, поскольку проблемы с электропитанием или сетевыми подключениями на уровне стойки приводят к недоступности всего размещаемого в ней оборудования. Таким образом, типичный домен отказа соответствует серверной стойке (если cluster настроен с учётом топологии размещения оборудования в стойках) или отдельному серверу.

Сбои уровня домена отказа автоматически обрабатываются YDB без остановки кластера.

Канал распределённого хранилища

Канал распределённого хранилища, **канал**, **distributed storage channel**, **DS channel** или **channel** — это логическое соединение между таблеткой и группой хранения. Таблетка может записывать данные в различные каналы, и каждый канал отображается на определённую группу хранения. Наличие нескольких каналов позволяет таблетке:

- Записывать больше данных, чем может содержать одна группа хранения.
- Хранить различные LogoBlob в различных группах хранения, с различными свойствами, такими как кодирование с исправлением ошибок или на различных носителях (HDD, SSD, NVMe).

Реализация распределённых транзакций

Ниже объяснены термины, связанные с реализацией распределённых транзакций. Сама реализация описана в отдельной статье DataShard: распределённые транзакции.

Детерминированные транзакции

Распределённые транзакции YDB вдохновлены исследовательской работой Building Deterministic Transaction Processing Systems without Deterministic Thread Scheduling Александра Томсона и Дэниела Дж. Абади из Йельского университета. В статье введено понятие **детерминированной транзакционной обработки** или **deterministic transaction processing**, которое позволяет эффективно обрабатывать распределённые транзакции. Оригинальная статья накладывала ограничения на виды операций, которые могут выполняться таким образом. Поскольку эти ограничения мешали реальным пользовательским сценариям, YDB развила свои алгоритмы для их выполнения, используя детерминированные транзакции в качестве этапов выполнения пользовательских транзакций с дополнительной оркестрацией и блокировками.

Оптимистичные блокировки

Как и во многих других системах управления базами данных, запросы YDB могут накладывать блокировки на определённые фрагменты данных, такие как строки таблиц, чтобы гарантировать, что параллельные изменения не приведут их в несогласованное состояние. Однако YDB проверяет эти блокировки не в начале транзакций, а при попытках их коммита (фиксации). Первый подход называется **пессимистичными блокировками** или **perssimistic locking** (например, используется в PostgreSQL), а второй — **оптимистичными блокировками** или **optimistic locking** (используется в YDB).

Инвалидация блокировок транзакций

Инвалидация блокировок транзакций (Transaction Lock Invalidation, **TLI**) — штатное поведение YDB при конфликте параллельных транзакций в рамках оптимистичных блокировок. Если одна транзакция (нарушитель) записывает данные и тем самым ломает блокировки другой транзакции (жертвы), YDB обнаруживает это при коммите жертвы и откатывает её с ошибкой `transaction locks invalidated`. Подробнее о диагностике TLI см. в Инвалидация блокировок транзакций.

Этап подготовки

Этап подготовки — это фаза транзакции, во время которой тело транзакции регистрируется на всех участвующих шардах.

Этап выполнения

Этап выполнения — это фаза транзакции, в ходе которой запланированная транзакция исполняется и генерируется ответ.

В некоторых случаях, вместо подготовки и выполнения, транзакция немедленно выполняется и генерируется ответ. Например, это происходит для транзакций, затрагивающих только один шард или для согласованных чтений из снимка данных (snapshot).

Грязные операции

В случае только для чтения транзакций, аналогично "read uncommitted" в других системах управления базами данных, может потребоваться чтение данных, которые ещё не были зафиксированы на диске. Это называется **грязными операциями** или **dirty operations**.

Набор чтения-записи

Набор чтения-записи, **RW-набор**, **read-write set** или **RW-set** — это набор данных, который будет участвовать в выполнении распределённой транзакции. Он объединяет данные набора чтения, которые будут считываться, и набор записи, для которого будут выполняться модификации.

Набор чтения

Набор чтения, **ReadSet данные**, **read set** или **ReadSet data** — это то, что пересылают участвующие шарды во время выполнения транзакции. В случае транзакций с данными он может содержать информацию о состоянии оптимистичных блокировок, готовности шарда к коммиту или решение отменить транзакцию.

Транзакционные прокси

Транзакционные прокси, **transaction proxy** или `TX_PROXY` — это сервис, который оркестрирует выполнение многих распределённых транзакций: последовательные фазы, выполнение фаз, планирование и агрегацию результатов. В случае прямой оркестрации другими акторами (например, QP data transactions), он используется для кэширования и выделения уникальных TxID.

Флаги транзакции

Флаги транзакции, **transaction flags** или **TxFlags** — это битовая маска флагов, которые каким-то образом изменяют выполнение транзакции.

Идентификатор транзакции

Идентификатор транзакции или **TxID** — это уникальный идентификатор, назначаемый каждой транзакции при её принятии YDB.

Идентификатор порядка транзакции

Идентификатор порядка транзакции или **transaction order id** — это уникальный идентификатор, назначаемый каждой транзакции во время планирования. Он состоит из PlanStep и Transaction ID.

Планируемый шаг

Планируемый шаг, **шаг**, **PlanStep** или **Step** — это логическое время, на которое запланировано выполнение набора транзакций.

Время медиатора

Во время выполнения распределённых транзакций, **время медиатора**, **медиаторное время** или **mediator time** — это логическое время, до которого (включительно) шард-участник должен знать весь план выполнения. Оно используется для продвижения времени при отсутствии транзакций на конкретном шарде, чтобы определить, может ли он читать из снимка (snapshot).

MiniKQL

MiniKQL — это язык, который позволяет выразить одну детерминированную транзакцию в системе. Это функциональный, строго типизированный язык. Концептуально язык описывает граф чтения из базы данных, выполнения вычислений над прочитанными данными и записи результатов в базу данных и/или в специальный документ, представляющий результат запроса (для показа пользователю). Транзакция MiniKQL должна явно задавать свой набор чтения (читаемые данные) и предполагать детерминированный выбор ветвей выполнения (например, отсутствует случайность).

MiniKQL — это язык низкого уровня. Конечные пользователи системы видят только запросы на языке YQL, который опирается на MiniKQL в своей реализации.

Query Processor

Query Processor или **QP** (ранее **KQP**) — это компонент YDB, отвечающий за оркестрацию выполнения пользовательских запросов и генерацию окончательного ответа.

Глобальная схема

Глобальная схема, **схема базы данных**, **global scheme**, **global schema** или **database schema** — это схема всех данных, хранящихся в базе данных. Она состоит из таблиц и других сущностей, таких как топики. Метаданные об этих сущностях называются глобальной схемой. Термин используется в противоположность **локальной схеме** или **local schema**, которая относится к схеме данных внутри таблетки. Пользователи YDB никогда не видят локальную схему и работают только с глобальной схемой.

KiKiMR

KiKiMR — это устаревшее название YDB, использовавшееся до того, как он стал продуктом с открытым исходным кодом (open source). Оно всё ещё может встречаться в исходном коде, старых статьях и видео и т.д.

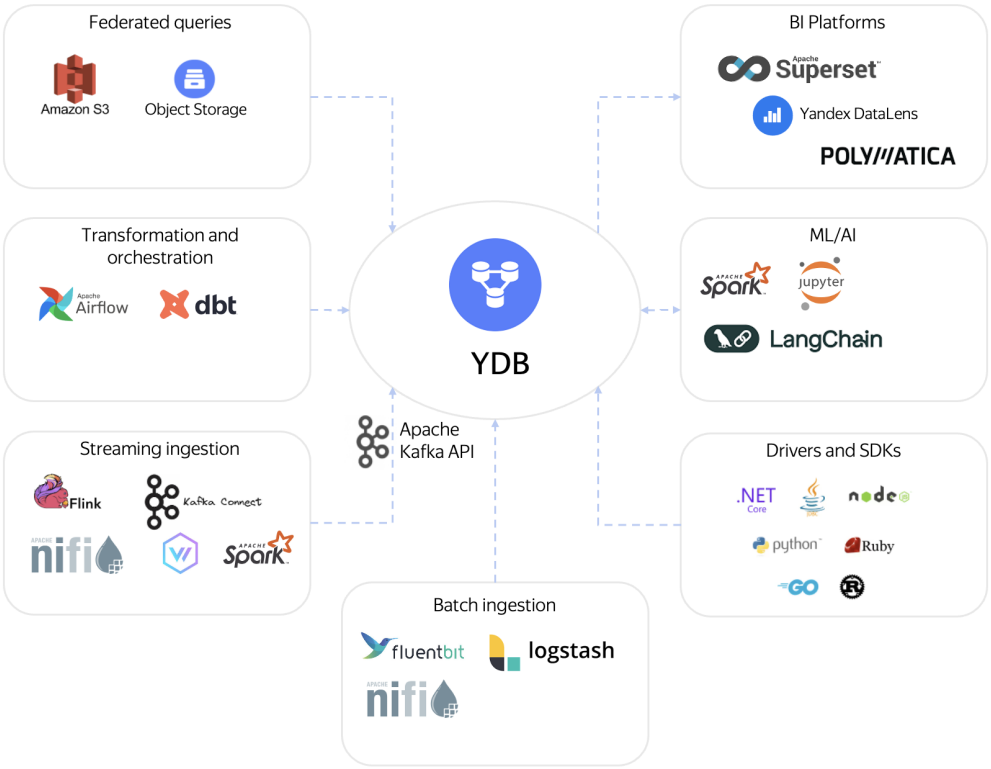
Recovery-режим таблетки

Recovery-режим — режим, в котором таблетка запускается и доступна через YDB UI, но **не отвечает на запросы**, **не выполняет никаких фоновых активностей** и **не читает данные из распределённого хранилища**, что позволяет выполнять операции восстановления. Остальные таблетки продолжают работать в штатном режиме, что позволяет кластеру продолжать функционировать, но будут недоступны операции и UI, связанные с таблеткой, переведённой в Recovery-режим.

Обзор

YDB — это распределённая отказоустойчивая база данных, которая предоставляет компоненты для построения корпоративного хранилища данных (КХД) на единой платформе.

Использование YDB позволяет консолидировать функциональность нескольких технологий (например, отдельных систем для потоковой обработки, хранения и аналитики) в одном решении. Вы можете использовать привычные инструменты и подходы, получая при этом свойства распределённой системы.



Ввод данных

Платформа спроектирована для приёма потоковых и пакетных данных в больших объёмах.

- Потоковая обработка: встроенная система топиков с поддержкой Kafka API для интеграции с существующими системами. Для сбора логов доступны плагины к Fluent Bit и Logstash.
- Пакетная загрузка: API BulkUpsert для быстрой загрузки массивов данных и коннектор к Apache Spark для интеграции с платформами обработки данных.
- Подключение через стандартные интерфейсы: JDBC-драйвер и нативные SDK.

Подробнее о вводе данных

Хранение данных

Основа хранилища — колоночные таблицы со встроенным сжатием, оптимизированные для аналитических нагрузок.

- Разделение хранения и вычислений: ключевое свойство YDB, позволяющее независимо масштабировать дисковое пространство и вычислительные мощности.
- Минимизация администрирования: фоновые процессы уплотнения (compaction) и удаления данных по TTL уменьшают количество ручных операций.

Подробнее о хранении данных

Выполнение запросов

YDB — это MPP-СУБД (Massively Parallel Processing) без выделенного мастер-узла. Узлы выполняют одинаковые роли, и система масштабируется горизонтально за счёт динамического добавления или удаления вычислительных ресурсов.

- Стоимостной оптимизатор (cost-based optimizer, CBO): выбирает план выполнения для запросов, анализируя статистику по данным.
- Механизм сброса на диск (spilling): позволяет выполнять запросы, промежуточные результаты которых не помещаются в оперативную память.
- Workload Manager: управляет распределением ресурсов между запросами, изолируя разные типы нагрузки.

Подробнее о выполнении запросов

Трансформация данных

Поддерживается трансформация данных с использованием стандартных подходов и инструментов.

- ELT на SQL: используйте `INSERT INTO ... SELECT` для построения витрин. Для управления сложными SQL-пайплайнами есть интеграция с dbt.
- ETL на Apache Spark: запускайте ETL-задания на Apache Spark с помощью параллельного коннектора.
- Оркестрация: автоматизируйте пайплайны с помощью Apache Airflow.

Подробнее о трансформации данных

Федеративные запросы

YDB позволяет выполнять запросы к данным в S3-совместимых хранилищах без их предварительной загрузки. Это упрощает работу с данными, хранящимися в озере данных (data lake).

Подробнее о федеративных запросах

Анализ и визуализация (BI и ML)

Для анализа данных можно использовать стандартные инструменты индустрии:

- BI-инструменты: Yandex DataLens, Apache Superset, Grafana, Polymatica.
- ML-инструменты: используйте Jupyter Notebook и Apache Spark для подготовки данных и обучения моделей машинного обучения.

Подробнее о BI; Подробнее о ML

Ввод данных

YDB спроектирована для приема как потоковых, так и пакетных данных. Отсутствие выделенных мастер-узлов позволяет осуществлять запись данных параллельно на все узлы базы данных, благодаря чему пропускная способность записи линейно масштабируется с ростом кластера. Выбор инструмента зависит от требований к задержке, гарантиям доставки и объему данных.

Потоковая загрузка (Real-time)

Для сценариев, требующих минимальной задержки — логи, метрики, потоки CDC.

- Топики с Kafka API: основной и рекомендуемый способ потоковой загрузки. Топики — это встроенный в YDB аналог Apache Kafka. Благодаря поддержке Kafka API вы можете использовать существующие клиенты и системы (Apache Flink, Spark Streaming, Kafka Connect) без изменений. Ключевое преимущество — возможность транзакционной записи из топика в таблицу, которая гарантирует семантику `exactly-once` на уровне базы данных;
- Плагины для Fluent Bit / Logstash: если вы используете Fluent Bit или Logstash для сбора логов, специализированные плагины позволяют писать данные напрямую в YDB, минуя промежуточные брокеры сообщений;
- Встроенный перенос данных (Transfer): сервис Transfer позволяет в потоковом режиме трансформировать и переносить данные из топиков в таблицы.

Пакетная загрузка (Batch)

Для загрузки больших объемов исторических данных, выгрузок из других систем или результатов batch-заданий.

- BulkUpsert — наиболее производительный способ для пакетной вставки. `BulkUpsert` — это специализированный API, оптимизированный для максимальной пропускной способности. Он требует меньше ресурсов по сравнению с транзакционными операциями, что позволяет загружать большие массивы данных с максимальной скоростью;
- Федеративные запросы к данным в S3 / Data Lake — YDB позволяет выполнять SQL-запросы напрямую к данным, хранящимся в объектных хранилищах, совместимых с Amazon S3, или в других внешних системах. Это удобный способ для загрузки данных без использования отдельных ETL-инструментов;
- Коннектор к Apache Spark сохраняет данные напрямую в таблицы YDB в многопотоковом режиме для максимально производительной записи;
- JDBC-драйвер и нативные SDK — с их помощью можно подключить любые приложения или пайплайны, включая Apache Spark, Apache NiFi и другие решения.

Хранение данных

Эффективное хранение данных — основа любого аналитического хранилища. YDB использует колоночный формат, архитектуру с разделением хранения и вычислений и автоматические процессы обслуживания для обеспечения высокой производительности и низкой стоимости владения.

Колоночные таблицы

Данные в колоночных таблицах хранятся по столбцам, а не по строкам. Этот подход является стандартом для OLAP-систем и даёт два ключевых преимущества:

1. Сокращение объёма чтения: при выполнении запроса (`SELECT column_a, column_b FROM ...`) с диска считываются данные только тех столбцов, которые участвуют в запросе;
2. Сжатие данных: колоночная организация хранения позволяет применять сжатие и кодирование данных индивидуально для каждой колонки, что позволяет сжать данные лучше, чем разнородные данные строки. По умолчанию используется алгоритм сжатия `LZ4` без специального кодирования, однако обе настройки могут быть заданы отдельно для каждой колонки. Подробнее в разделе Параметры запроса CREATE TABLE.

Архитектура с разделением хранения и вычислений

Разделение хранения и вычислений — это архитектурный принцип YDB. Узлы, отвечающие за хранение данных (storage nodes), и узлы, выполняющие запросы (dynamic nodes), разделены. Это позволяет:

- Независимо масштабировать ресурсы: если не хватает места на диске, вы добавляете узлы хранения. Если не хватает CPU для запросов — добавляете узлы вычислений. Это отличается от систем с жёстко связанными ресурсами хранения и вычислений;
- Быстро перераспределять нагрузку: для перераспределения вычислительной нагрузки между узлами не требуется физическое перемещение данных, переносятся только метаданные.

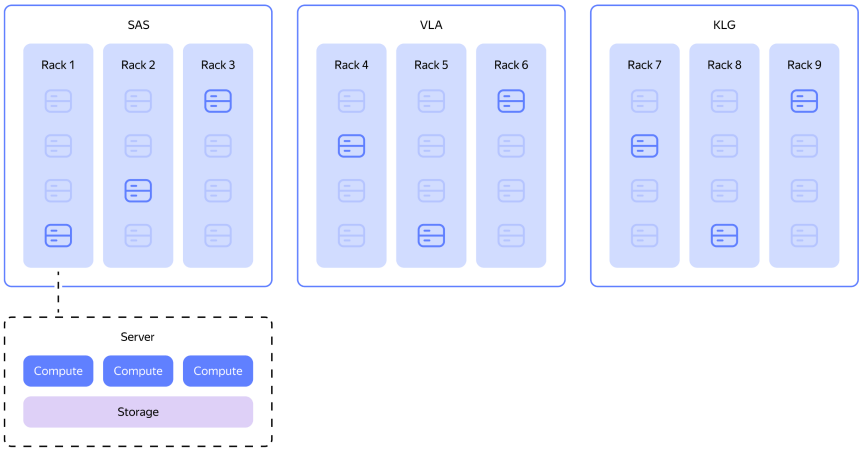
Автоматическая оптимизация хранения

YDB спроектирован так, чтобы минимизировать ручные операции по обслуживанию.

- Автоматическое уплотнение (compaction) данных: Данные хранятся в LSM-подобных структурах; процессы слияния и оптимизации данных происходят постоянно в фоновом режиме; вам не нужно запускать `VACUUM` или аналогичные команды.
- Автоматическое удаление данных: Для управления жизненным циклом данных используйте механизм удаления по TTL.

Встроенная отказоустойчивость

YDB изначально создавалась как отказоустойчивая система и поддерживает различные режимы размещения данных для защиты от сбоев оборудования, стоек или даже целых дата-центров.



Выполнение запросов

YDB — это распределённая MPP-база данных (Massively Parallel Processing), предназначенная для выполнения сложных аналитических запросов на больших объёмах данных. Каждый запрос автоматически распараллеливается по всем доступным вычислительным узлам кластера, что позволяет эффективно использовать вычислительные ресурсы.

YDB поддерживает ряд ключевых технологий, обеспечивающих высокую производительность и стабильность:

- Децентрализованная MPP-архитектура
- Стоимостной оптимизатор запросов
- Обработка данных, превышающих объем RAM
- Изоляция нагрузок и управление ресурсами

Децентрализованная MPP-архитектура

В отличие от MPP-систем с выделенным управляющим узлом (master node), архитектура YDB полностью децентрализована. Это обеспечивает два основных преимущества:

- Высокая отказоустойчивость: любой узел кластера может принять и координировать выполнение запроса. Отсутствует единая точка отказа (SPOF). Выход из строя части узлов не приводит к остановке кластера — нагрузка автоматически перераспределяется на оставшиеся узлы.
- Масштабируемость вычислений: вы можете добавлять или удалять вычислительные узлы без простоев, и система автоматически адаптируется, распределяя нагрузку с учётом нового состава кластера.

Стоимостной оптимизатор запросов

Перед выполнением запроса YDB использует стоимостной оптимизатор (Cost-Based Optimizer). Он анализирует текст запроса, метаданные и статистику по распределению данных в таблицах, чтобы построить физический план выполнения с наименьшей оценочной стоимостью.

Оптимизатор способен:

- Выбирать порядок соединения таблиц для запросов с десятками `JOIN`'ов;
- Подбирать алгоритмы распределенных `JOIN`'ов (например, Grace Join, Broadcast Join) в зависимости от размера таблиц;
- «Проталкивать» фильтры (`WHERE`) как можно ближе к источникам данных для уменьшения объема обрабатываемой информации на последующих стадиях.

Обработка данных, превышающих объем RAM

Аналитические запросы могут требовать для своей работы большие объемы оперативной памяти, особенно на операциях `JOIN` и `GROUP BY`. YDB спроектирована для работы с данными, которые могут не помещаться в RAM.

- Спиллинг (Spilling): Если промежуточные результаты выполнения запроса превышают лимит оперативной памяти, YDB автоматически выгружает их на локальный диск узла. Это предотвращает падение запроса с ошибкой «Out of Memory» и позволяет выполнять запросы над большими объемами данных.
- Распределенные алгоритмы JOIN: для соединения таблиц, превышающих память одного узла, используются распределенные алгоритмы, которые обрабатывают данные по частям на разных узлах.

Изоляция нагрузок и управление ресурсами

В корпоративном DWH часто работают разные команды с разными типами нагрузки. Чтобы эти нагрузки не мешали друг другу, в YDB встроен менеджер управления ресурсами.

- Workload Manager: Встроенный менеджер нагрузки позволяет создавать ресурсные пулы и с помощью классификаторов назначать запросы разных групп пользователей в разные пулы. Этот механизм решает проблему «шумных соседей», когда один ресурсоемкий запрос может замедлять работу системы для остальных пользователей.

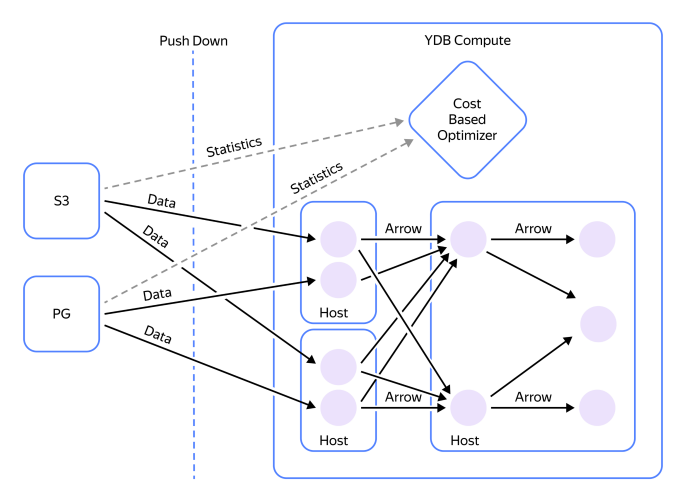
Федеративные запросы

Федеративные запросы — это возможность выполнять запросы к данным, хранящимся во внешних системах, без их предварительной загрузки (ETL) в YDB. Наиболее популярный сценарий — работа с данными в объектных хранилищах, совместимых с S3.

Как это работает

Вы можете создать во YDB внешнюю таблицу, которая ссылается на данные в S3. При выполнении запроса `SELECT` к такой таблице YDB инициирует параллельное чтение данных со всех вычислительных узлов. Каждый узел считывает и обрабатывает только ту часть данных, которая ему необходима.

- Поддерживаемые форматы: Parquet, CSV, JSON с различными алгоритмами сжатия.
- Оптимизация чтения: YDB использует механизмы оптимизации чтения данных из S3 (partition pruning) для Hive-размещения данных и для более сложных алгоритмов размещения.



Трансформация и подготовка данных (ETL/ELT)

Подготовка данных для анализа — ключевой этап в построении хранилища данных. YDB поддерживает все стандартные подходы к трансформации данных, позволяя выбрать инструмент, наиболее подходящий для конкретной задачи: от чистого SQL до сложных пайплайнов на Apache Spark.

ELT

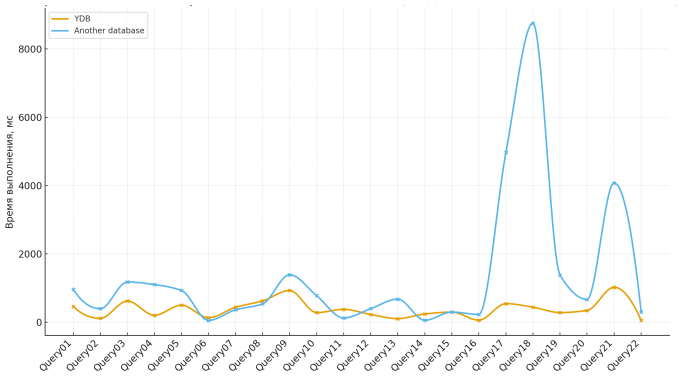
Трансформации данных с помощью SQL зачастую являются наиболее производительными, поскольку вся обработка происходит непосредственно в движке YDB, без перемещения данных во внешние системы и обратно. Логика описывается на SQL и выполняется распределённым MPP-движком, оптимизированным для аналитических операций.

Производительность в бенчмарке TPC-H

Производительность операций ELT напрямую зависит от скорости выполнения сложных аналитических запросов. Стандартным отраслевым тестом для оценки таких запросов является TPC-H.

Сравнение с другой распределённой аналитической СУБД на наборе запросов TPC-H показывает, что YDB демонстрирует более стабильную производительность, особенно при выполнении запросов, содержащих:

- Соединения (`JOIN`) большого числа таблиц, пяти и более;
- Вложенные подзапросы, используемые для фильтрации;
- Агрегации (`GROUP BY`) с последующей сложной фильтрацией по результатам.



Такая стабильность свидетельствует о высокой эффективности стоимостного оптимизатора запросов YDB при построении планов выполнения для комплексных SQL-шаблонов, типичных для реальных ELT-процессов. Для платформы хранилища данных (DWH) это означает предсказуемое время обновления данных и снижение рисков неконтролируемой деградации производительности в продакшн-среде.

Основные сценарии

- Построение витрин данных: используйте привычный синтаксис `INSERT INTO ... SELECT FROM ...` для создания агрегированных таблиц (витрин) из «сырых» данных;
- Объединение OLTP и OLAP данных: YDB позволяет в одном запросе объединять данные как из транзакционных (строковых), так и из аналитических (колоночных) таблиц. Это даёт возможность обогащать «холодные» аналитические данные актуальной информацией из OLTP-системы без необходимости их дублирования;
- Массовое обновление: для «слепой» записи больших объёмов данных без проверки существования можно использовать оператор `UPSERT INTO` .

Управление SQL-пайплайнами с помощью dbt

Для управления сложными SQL-пайплайнами используйте плагин к dbt. Плагин позволяет дата-инженерам описывать модели данных в виде `SELECT` -запросов, а dbt автоматически строит граф зависимостей между моделями и выполняет их в правильном порядке. Этот подход помогает реализовать принципы software engineering (тестирование, документирование, версионирование) при работе с SQL-кодом.

ETL

Сложные трансформации с помощью внешних фреймворков

Для задач, требующих сложной логики на языках программирования (Python, Scala, Java), интеграции с ML-пайплайнами или обработки больших объемов данных, удобно использовать внешние фреймворки для распределенной обработки.

Apache Spark — один из самых популярных инструментов для таких задач, для которого разработан специализированный коннектор к YDB. Если в вашей компании используются другие схожие решения (например, Apache Flink), их также можно задействовать для построения ETL-процессов за поддержки JDBC-драйвера.

Ключевое преимущество YDB при работе с такими системами — архитектура, позволяющая считывать данные параллельно. В YDB нет выделенного мастер-узла для экспорта, поэтому внешние инструменты могут читать информацию напрямую со всех узлов хранения. Это обеспечивает высокую скорость и линейную масштабируемость чтения.

Оркестрация пайплайнов

Для запуска пайплайнов по расписанию и управления зависимостями используются оркестраторы.

- Apache Airflow: для оркестрации данных в YDB поддерживается провайдер к Apache Airflow. С его помощью можно создавать DAG'и, которые запускают `dbt run` , выполняют YQL-скрипты или инициируют Spark-джобы.
- Встроенные механизмы: для некоторых задач не требуется внешний оркестратор. YDB может выполнять часть операций автоматически:
  - Удаление устаревших данных по TTL: автоматическая очистка партиций по истечении заданного времени.
  - Автоматический compaction: процессы слияния и оптимизации данных для LSM-дерева происходят в фоновом режиме, что исключает необходимость регулярного запуска команд вроде `VACUUM` .
- Другие оркестраторы: если в вашей компании принят другой инструмент (например, Dagster, Prefect) или собственный планировщик, вы можете использовать его для запуска тех же команд. Большинство оркестраторов позволяют выполнять shell-скрипты, что дает возможность вызывать YDB CLI, dbt run и другие утилиты.

Интеграция с другими ETL-инструментами через JDBC

YDB предоставляет JDBC-драйвер, что позволяет использовать широкий спектр существующих ETL-инструментов, таких как Apache NiFi и другие JDBC-совместимые системы.

BI-аналитика и визуализация данных

Интерактивность BI-дешбордов напрямую зависит от производительности базы данных, на которой они построены. YDB спроектирована как высокопроизводительная аналитическая платформа, выполняющая запросы за доли секунды, что позволяет аналитикам работать с данными в интерактивном режиме.

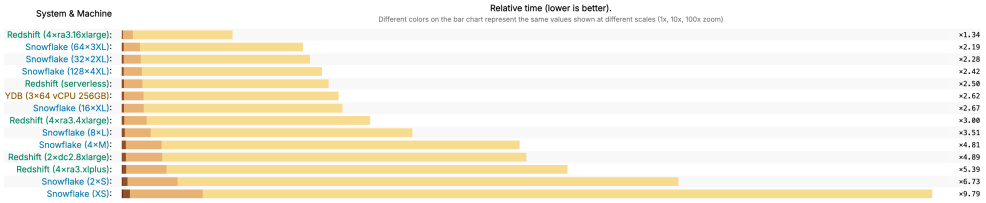
Это достигается за счёт ключевых архитектурных особенностей:

- Колоночное хранение: запросы считывают с диска только те столбцы, которые указаны в запросе, что сокращает объём операций ввода-вывода.
- MPP-архитектура: каждый запрос распараллеливается по всем доступным вычислительным узлам кластера, что позволяет задействовать все ресурсы для его выполнения.
- Децентрализованная архитектура: отсутствие единого мастер-узла позволяет эффективно обрабатывать множество параллельных запросов от пользователей BI-систем.

Производительность в независимых бенчмарках

Хотя синтетические тесты не всегда отражают реальную нагрузку, они служат хорошей отправной точкой для сравнения производительности. ClickBench — независимый бенчмарк для аналитических СУБД, разработанный создателями ClickHouse.

На наборе из 43 аналитических запросов YDB показывает конкурентоспособные результаты, опережая многие популярные open-source и облачные аналитические базы данных. Это подтверждает высокую производительность движка на типовых OLAP-запросах.



Интеграции с BI-платформами

YDB поддерживает следующие BI-платформы:

- Yandex DataLens;
- Apache Superset;
- Grafana;
- Polymatica.

Machine Learning

YDB служит эффективной платформой для хранения и обработки данных в ML-пайплайнах. Вы можете использовать привычные инструменты, такие как Jupyter Notebook и Apache Spark, на всех этапах жизненного цикла ML-модели.

Feature Engineering

Используйте YDB в качестве движка для подготовки признаков:

- SQL и dbt: Выполняйте сложные аналитические запросы для агрегации «сырых» данных и создания новых признаков. Материализуйте наборы признаков в строковые таблицы для быстрого доступа;
- Apache Spark: Для более сложных преобразований, требующих логики на Python или Scala, используйте коннектор к Apache Spark для чтения данных, их обработки и сохранения результатов обратно в YDB.

Обучение моделей

YDB может выступать в роли быстрого и масштабируемого источника данных для обучения моделей:

- Интеграция с Jupyter: Подключайтесь к YDB из Jupyter Notebook для ad-hoc анализа и прототипирования моделей;
- Распределённое обучение: коннектор к Apache Spark обеспечивает параллельное чтение данных со всех узлов кластера напрямую в Spark DataFrame. Это позволяет загружать обучающие выборки для моделей в PySpark MLlib, CatBoost, Scikit-learn и другие библиотеки.

Online Feature Store

Сочетание строковых (OLTP) и колоночных (OLAP) таблиц в YDB позволяет реализовать на одной платформе не только аналитическое хранилище, но и Online Feature Store.

- Используйте строковые (OLTP) таблицы для хранения признаков, требующих точечного чтения с низкой задержкой; это позволяет ML-моделям в реальном времени получать признаки для инференса.
- Используйте колоночные (OLAP) таблицы для хранения исторических данных и пакетного расчёта этих признаков.

Ключевые возможности для аналитики: быстрый справочник

Эта страница — карта документации по аналитическим возможностям YDB. Текст сгруппирован по этапам жизненного цикла данных, чтобы помочь быстро найти необходимую информацию для проектирования, разработки и эксплуатации аналитических решений.

Проектирование хранилища данных (Concepts & Design)

Основы организации данных, масштабирования и управления.

Основные концепции и типы данных

- Колоночные таблицы: архитектура хранения, оптимизированная для OLAP.
- Типы данных: полный справочник по поддерживаемым типам.

Масштабирование и производительность

- Проектирование ключей для максимальной производительности: как выбирать `PRIMARY KEY` и `PARTITION BY`.
- Партиционирование таблиц: механизм распределения данных по узлам.

Управление жизненным циклом данных

- TTL (Time-to-Live): автоматическое удаление устаревших данных по истечении срока.

Загрузка и выгрузка данных (Ingestion & Egress)

Инструменты и API для перемещения данных в YDB и из неё.

Потоковая загрузка (Streaming Ingestion)

- Topics (Kafka API): нативная работа с потоками данных через протокол Kafka.
- Transfer: управляемый сервис для переноса данных между топиками и таблицами.
- Коннектор Fluent Bit: прямая загрузка логов.

Пакетная загрузка (Batch Ingestion)

- Коннектор Apache Spark: чтение и запись данных для ETL/ELT-задач.
- BulkUpsert API: высокопроизводительная вставка больших объемов данных через SDK.

Взаимодействие с внешними системами

- Федеративные запросы: выполнение запросов к данным, находящимся во внешних системах (S3, ClickHouse, Postgres).
- Работа с S3 через внешние таблицы: чтение и запись данных в формате Parquet/CSV в Object Storage.

Обработка и трансформация данных (ETL/ELT)

Язык запросов и интеграция с инструментами оркестрации.

Язык запросов YQL

- Полный справочник по YQL: синтаксис, функции и операторы.
- Функции для работы с датой и временем: полный список и типовые сценарии.
- Функции для работы с JSON: извлечение данных из JSON-документов.

Инструменты для построения пайплайнов

- Интеграция с dbt (Data Build Tool): управление ELT-пайплайнами с помощью SQL.
- Интеграция с Apache Airflow: оркестрация сложных ETL/ELT-процессов.

Разработка и интеграция с приложениями (Development & SDKs)

Инструменты для разработчиков приложений.

- Обзор YDB SDK: нативные SDK для Go, Python, Java, C++, JavaScript.
- JDBC драйвер: стандартный способ подключения из Java-экосистемы.
- YDB CLI: инструмент командной строки для администрирования и выполнения запросов.

Анализ данных и визуализация (Analytics & BI)

Интеграция с инструментами для конечных пользователей.

BI-системы

- Apache Superset
- Grafana
- Yandex DataLens

Инструменты Data Science

- Jupyter Notebooks: выполнение YQL-запросов и анализ данных в интерактивном режиме.

Эксплуатация и управление производительностью (Operations & Performance)

Администрирование, мониторинг, безопасность и оптимизация.

Управление производительностью

- Анализ планов выполнения запросов (EXPLAIN): как понять план выполнения запроса и найти узкие места.
- Управление нагрузкой (Resource Pools): изоляция ресурсов CPU для разных команд или нагрузок.
- Стоимостной оптимизатор: обзор принципов работы планировщика запросов.

Мониторинг и диагностика

- YDB UI: веб-интерфейс для мониторинга состояния и диагностики кластера.
- Справочник по метрикам: полный список метрик для систем мониторинга.
- Готовые дашборды для Grafana: шаблоны для быстрой настройки мониторинга.

Безопасность и отказоустойчивость

- Аутентификация и авторизация: настройка доступа пользователей, в том числе через LDAP.

Архитектурные ограничения

- Известные ограничения системы: важный раздел для понимания особенностей и компромиссов архитектуры.

Потоковая обработка данных

Данный раздел описывает ключевые концепции потоковой обработки (stream processing) в YDB:

- Потоковые запросы — тип запросов для непрерывной обработки входящих событий. Описываются источники и приёмники данных, гарантии доставки, ограничения и управление запросами.
- Watermarks — механизм отслеживания прогресса обработки событий по времени.

Потоковые запросы

Потоковый запрос — это тип запроса, предназначенный для непрерывной обработки неограниченного потока данных (stream processing). В отличие от обычных запросов, потоковый запрос не завершается после получения результата, а работает постоянно, обрабатывая данные по мере их поступления.

Потоковая обработка данных — широко применяемый подход, реализованный в таких системах, как Apache Flink, Apache Kafka Streams, Amazon Kinesis Data Analytics. Типичные сценарии применения — мониторинг и алертинг, агрегация метрик по временным окнам, трансформация и фильтрация событий на лету, обогащение событий данными из справочников, поиск паттернов в последовательностях событий.

YDB реализует потоковую обработку как часть единой платформы для работы с данными. В дополнение к типовым сценариям использования потоковых запросов, интеграция в общую платформу YDB позволяет получать данные из топиков YDB, обрабатывать потоки изменений из строковых таблиц через CDC, а результаты обработки записывать в выходные топики или напрямую в таблицы. Это позволяет строить конвейеры обработки данных внутри YDB с минимальными временными задержками.

Отличия от обычных запросов

Обычные запросы работают с данными, которые уже сохранены в таблицах. Запрос выполняется, возвращает результат и завершается. Потоковый запрос создается и продолжает работу бесконечно до явной отмены пользователем. Данные непрерывно поступают в топик, «протекают» через запрос и записываются в приёмник — другой топик или таблицу.

Характеристика	Обычные запросы	Потоковые запросы
Данные	Конечный набор в таблицах	Бесконечный поток событий
Время жизни	Завершается после обработки	Работает непрерывно
Результат	Доступен после завершения	Обновляется по мере поступления данных
Восстановление при сбоях	Перезапуск вручную	Автоматическое восстановление из чекпоинта

Источники и приёмники данных

Потоковые запросы читают данные из топиков YDB и записывают результаты в топики или таблицы YDB. Данные в топик могут поступать из внешних систем (например, приложение записывает события через SDK), но сам потоковый запрос работает только с сущностями YDB. Прямое чтение из внешних систем, например из Apache Kafka, или запись в таблицы других баз данных, таких как PostgreSQL, не поддерживается.

Источники

Топики — основной источник потоковых данных. Запрос читает сообщения из одного или нескольких топиков и обрабатывает их по мере поступления. Данные в топик могут записывать:

- **Внешние приложения** — через YDB SDK или Kafka API. Например, сервис отправляет события телеметрии или логи в топик YDB, а потоковый запрос обрабатывает их.
- **CDC (Change Data Capture)** — потоки изменений из таблиц, реализуемые через встроенные топики. Позволяют реагировать на вставки, обновления и удаления записей в реальном времени. Подробнее: Change Data Capture (CDC).

При ручном изменении числа партиций в топике потоковый запрос будет автоматически перезапущен. После перезапуска будет происходить чтение всех партиций. Чтение из топиков с автопартиционированием не поддерживается.

Приёмники

Топики — для передачи результатов другим системам или следующим этапам обработки.

Таблицы — для материализации результатов. Данные сохраняются через UPSERT и доступны для обычных SQL-запросов.

Гарантии

В штатном режиме работы потоковые запросы обеспечивают гарантию доставки данных at-least-once — при сбоях запрос автоматически восстанавливается из чекпоинта и возобновляет чтение с сохранённых смещений.

При этом важно учитывать ограничения текущей реализации:

- **Сброс смещений при пересоздании запроса:** изменение текста запроса через DROP + CREATE приводит к удалению чекпоинта. Новый запрос начинает чтение с конца топика, пропуская события, поступившие в интервале между удалением старой версии и стартом новой.
- **Неполнота агрегатов из-за отсутствия watermarks:** из-за отсутствия механизма watermarks временные окна закрываются по системному времени (wall-clock). События, поступившие с задержкой (например, из-за медленных партиций или сетевых лагов), не попадают в агрегаты уже закрытых окон.

Подробное описание гарантий, аномалий и способов их минимизации — в разделе Гарантии доставки данных.

> **Примечание**
>
> Мы постоянно работаем над развитием механизмов потоковой обработки. В будущих версиях предоставляемые гарантии будут улучшены.

Ограничения

> **Важно**
>
> - Запрос должен содержать хотя бы одно чтение из топика, так как потоковая обработка требует наличия непрерывного входного потока данных.
> - Не поддерживается `JOIN` двух потоков (временное архитектурное ограничение).

Также не поддерживаются в текущей версии:

- Признак важного читателя для потребителей, используемых потоковыми запросами.

- Автопартиционирование (split/merge партиций) топиков, используемых потоковыми запросами. При увеличении числа партиций в топике, из которого читает работающий потоковый запрос, новые партиции не будут обрабатываться.
- Изменение текста работающего запроса. Чтобы изменить запрос, его нужно пересоздать — удалить с помощью DROP STREAMING QUERY и создать заново с помощью CREATE STREAMING QUERY.

Управление

Потоковые запросы создаются, изменяются и удаляются с помощью YQL-команд:

- CREATE STREAMING QUERY — создание;
- ALTER STREAMING QUERY — изменение и управление состоянием;
- DROP STREAMING QUERY — удаление.

Состояние запросов доступно в системной таблице `.sys/streaming_queries`.

Язык запросов

Потоковые запросы пишутся на YQL и поддерживают привычные SQL-конструкции: SELECT, WHERE, GROUP BY, JOIN. Для работы с временными окнами используется GROUP BY HOP.

Один потоковый запрос может читать несколько входных топиков, использовать конструкцию UNION ALL для объединения потоков данных и писать результат в несколько выходных топиков и/или таблиц (см. подробнее в статьях Форматы при записи данных и Запись в таблицы).

См. также

- Гарантии доставки данных — гарантии, аномалии при оконной агрегации и рекомендации;
- Быстрый старт: чтение и запись в топики — пошаговое руководство;
- Форматы данных при чтении/записи из топиков — форматы данных.

Watermarks

Watermark в потоковой обработке данных (stream processing) — это монотонно возрастающая нижняя оценка времён событий, которые ещё могут поступить в поток. Когда watermark достигает значения X, система объявляет, что все события с временем меньше X с высокой вероятностью уже получены.

YDB реализует механизм watermarks в потоковых запросах: они используются для корректного закрытия временных окон агрегации (HoppingWindow) и гарантируют, что результат окна выдаётся только тогда, когда система убедилась в полноте входных данных за этот период.

В потоковой обработке каждое событие имеет две временны́е метки: **время события** (event time) — момент, когда событие произошло в реальном мире, и **время обработки** (processing time) — момент, когда система получила событие. Из-за сетевых задержек, сбоев и неравномерной нагрузки эти два значения могут существенно расходиться. Именно поэтому системе нужен механизм watermarks: без него она не знает, когда можно считать прошедший временной диапазон достаточно полным, чтобы выдать результат.

Компромисс Точности и Задержки

В YDB этот компромисс регулируется явно: параметр `WATERMARK` в предложении `GROUP BY` задаёт выражение для вычисления watermark, в том числе величину отставания от времени последнего события. Это позволяет подобрать баланс между актуальностью результатов и полнотой учёта запоздавших событий.

Watermark не может одновременно учитывать сколь угодно большие задержки событий и продвигаться быстро: чем дольше система ждёт опоздавших событий, тем позже она выдаёт результаты. Это фундаментальный компромисс потоковой обработки.

События, поступившие после того, как watermark прошёл соответствующий временной диапазон, считаются опоздавшими и отбрасываются. Подробнее об обработке опоздавших событий и о настройке watermarks: Watermarks.

Архитектурный обзор YDB

Введение

YDB — это горизонтально масштабируемая, распределённая и отказоустойчивая универсальная система управления базами данных (СУБД), разработанная для обеспечения высокой производительности: например, типичный узел кластера способен обрабатывать десятки тысяч запросов в секунду. Система поддерживает георазпределённые (кросс-датацентровые) конфигурации — от небольших кластеров с несколькими узлами до кластеров из тысяч серверов, способных эффективно обрабатывать сотни петабайт данных.

Ключевые особенности и возможности YDB

- **Горизонтальное масштабирование и автоматическое партицирование**: данные и нагрузка распределяются динамически при росте объёма или интенсивности запросов.
- **Отказоустойчивость**: автоматическое восстановление при сбоях узлов, стоек и зон доступности.
- **Высокая доступность и сохранность данных**: обеспечивается с помощью автоматической синхронной репликации данных в пределах кластера.
- **Строгая консистентность и поддержка ACID-транзакций**: система обеспечивает распределённые транзакции с уровнем изоляции *serializable*. При необходимости уровень изоляции и строгость согласованности могут быть ослаблены для повышения производительности.
- **Язык запросов YQL** — SQL-диалект, оптимизированный для больших данных и сложных сценариев обработки.
- **Реляционная модель данных** с поддержкой строковых и колоночных таблиц, что делает возможным эффективное использование YDB как для транзакционных (OLTP), так и для аналитических (OLAP) нагрузок.
- **Иерархия объектов**: таблицы, топики и другие объекты организуются в иерархическое пространство имён, аналогично файловой системе.
- **Асинхронная репликация**: возможность настроить синхронизацию данных между базами YDB почти в реальном времени — как внутри одного кластера, так и между разными.
- **Потоковая обработка и распространение данных**:
  - **Топики** — хранение и потоковая доставка неструктурированных сообщений множеству подписчиков; поддерживают, в том числе, работу по протоколу Kafka.
  - **Change Data Capture (CDC)** — встроенная поддержка потокового чтения изменений данных из таблиц.
  - **Трансферы** — автоматическая доставка данных из топиков в таблицы.
- **Федеративные запросы** — выполнение запросов к внешним источникам данных (например, S3) в составе YQL-запросов без предварительной загрузки внешних данных в базу данных YDB.
- **Векторные индексы** — поддержка хранения и поиска по векторным представлениям данных для задач семантического поиска, сходства и ML-сценариев.
- **Наблюдаемость** — встроенная поддержка метрик, логов и дашбордов.
- **Безопасность и аудит** — шифрование данных (at-rest и in-transit), аудит операций, поддержка аутентификации и авторизации; подробнее — в разделе Безопасность.
- **Инструменты взаимодействия и интеграции** — доступны YDB CLI для администрирования и отладки, а также SDK для C++, C#, Go, Java, JavaScript, PHP, Python и Rust; поддерживается интеграция с внешними системами; подробнее — в разделах Интеграции YDB и Языки и API.
- **Открытая архитектура**: исходный код доступен под Apache License 2.0, взаимодействие с системой производится через открытый протокол gRPC, что позволяет реализовывать клиенты на любом языке программирования.

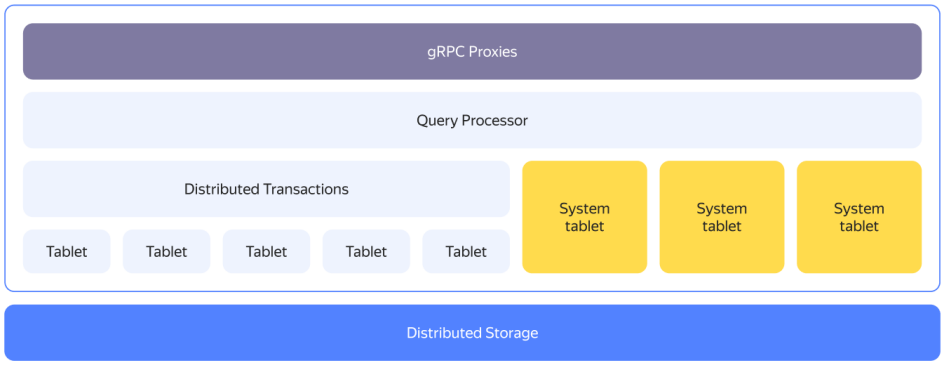
Основные сценарии применения

YDB — это универсальная платформа, подходящая для широкого спектра задач, где требуются масштабируемость, надёжность и гибкость. Основные сценарии:

- В распределённых системах, где требуется **строгая консистентность или поддержка транзакций**, включая изменение данных в нескольких строках и таблицах. YDB сочетает масштабируемость, характерную для NoSQL-систем, с возможностями реляционной СУБД по управлению согласованностью и целостностью данных.
- В системах, требующих хранения и обработки **очень большого объёма данных** с возможностью практически неограниченного горизонтального масштабирования (в эксплуатации находятся промышленные кластеры более чем из 5000 узлов, обрабатывающие нагрузку в миллионы запросов в секунду и хранящие петабайты данных).
- В высоконагруженных системах, где применяется **ручное шардирование** реляционных баз данных для распределения нагрузки. YDB упрощает архитектуру, устраняя необходимость в сложных шаблонах шардирования и ручной оркестрации.
- При разработке новых продуктов, для которых отсутствует надёжный прогноз нагрузки, или ожидается масштаб, превышающий возможности традиционных реляционных СУБД.
- В проектах, где требуется **гибкая платформа**, способная поддерживать разнообразные типы рабочих нагрузок и сценариев, включая транзакционные, потоковые и аналитические.

Как это работает?

Полное объяснение того, как работает YDB, получилось бы слишком объемным. Ниже вы можете ознакомиться с несколькими основными моментами, а затем продолжить изучение документации, чтобы узнать больше.

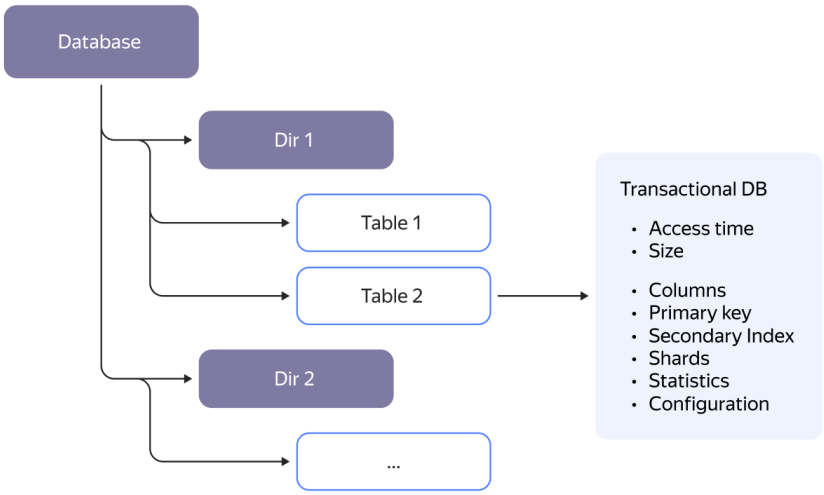


Кластеры YDB обычно работают с shared nothing архитектурой на обычном оборудовании. Уровни вычислений и хранения являются разнёсенными. Они могут работать как на отдельных наборах узлов, так и быть совмещёнными.

Один из ключевых элементов вычислительного слоя YDB называется *таблеткой*. Они являются логическими компонентами с состоянием, реализующими различные аспекты YDB.

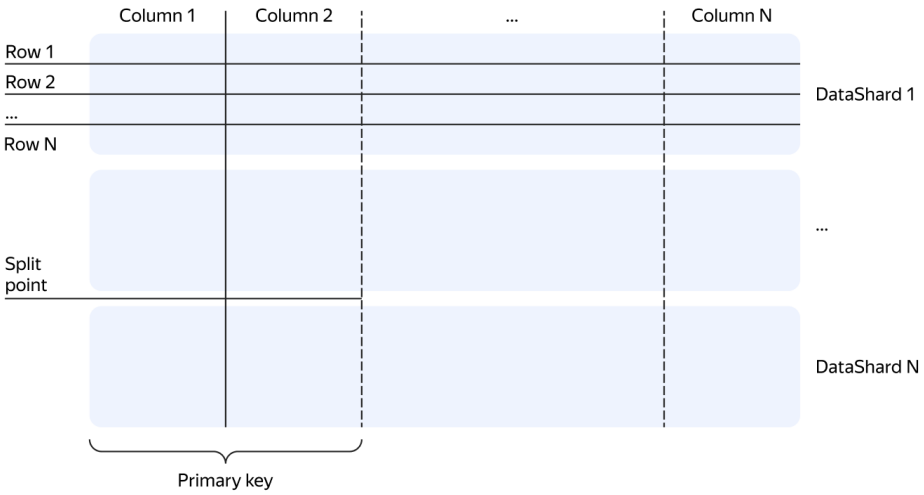
Более подробная общая архитектура YDB объясняется в разделе Общая схема YDB.

Иерархия



С точки зрения пользователя, всё внутри YDB организовано в иерархической структуре с использованием каталогов. Она может иметь произвольную глубину в зависимости от того, как вы решили организовать свои данные и проекты. Хотя YDB не имеет фиксированной глубины иерархии, как в других реализациях SQL, она все равно будет знакома, поскольку именно так выглядит любая виртуальная файловая система.

Таблица



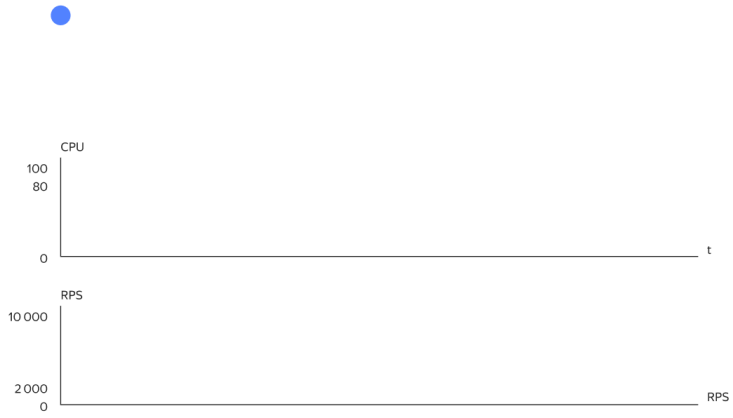
YDB предоставляет пользователям хорошо известную абстракцию — таблицы. В YDB существует два основных типа таблиц:

- Строковые таблицы предназначены для OLTP-нагрузок.
- Колоночные таблицы предназначены для OLAP-нагрузок.

Логически, с точки зрения пользователя, оба вида таблиц выглядят одинаково. Основное отличие между строковыми и колоночными таблицами заключается в способе хранения данных. В строковых таблицах значения всех колонок каждой строки располагаются рядом, а в колоночных таблицах — наоборот, каждая колонка хранится отдельно, и рядом оказываются ячейки, относящиеся к разным строкам.

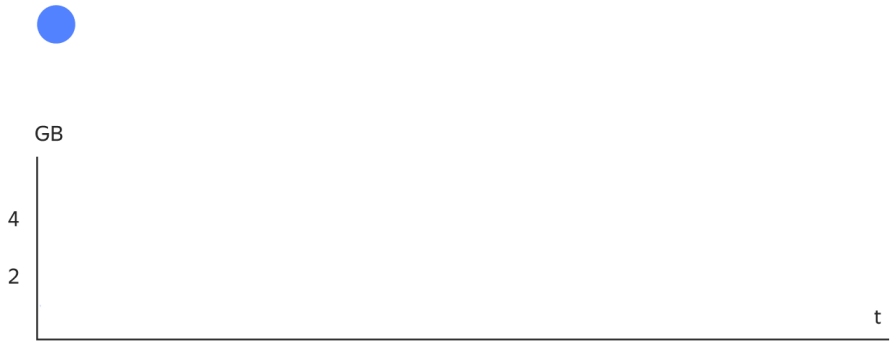
Независимо от типа таблицы, она должна содержать первичный ключ. В первичном ключе колоночных таблиц можно использовать только колонки с `NOT NULL`. Данные в таблицах физически сортируются по первичному ключу. Строковые таблицы автоматически партицируются по диапазонам первичных ключей в зависимости от объёма данных, а данные в колоночных таблицах партицируются не по первичному ключу, а по хешу от колонок партицирования. Каждый диапазон первичных ключей таблицы обрабатывается определённой таблеткой, называемой *data shard* для строчных таблиц и *column shard* — для колоночных.

Разделение по нагрузке



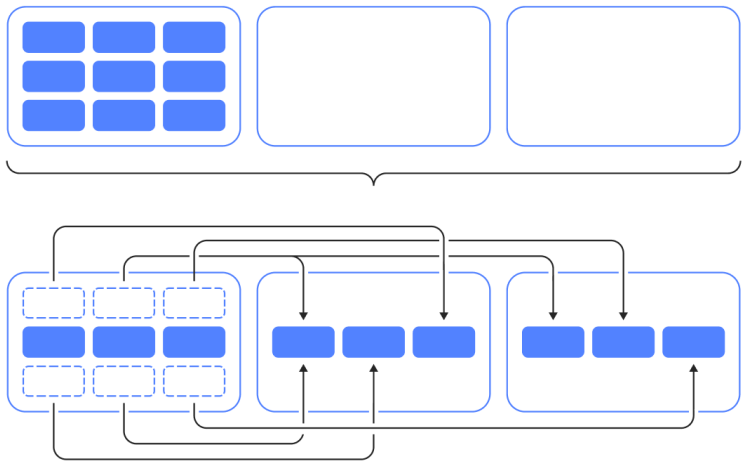
Data shard автоматически разделяются на большее количество при увеличении нагрузки. Они автоматически объединяются в нужное количество, когда пиковая нагрузка уходит.

Разделение по размеру



Data shard также автоматически разделяются при увеличении размера данных. Они автоматически сливаются обратно, если достаточное количество данных будет удалено.

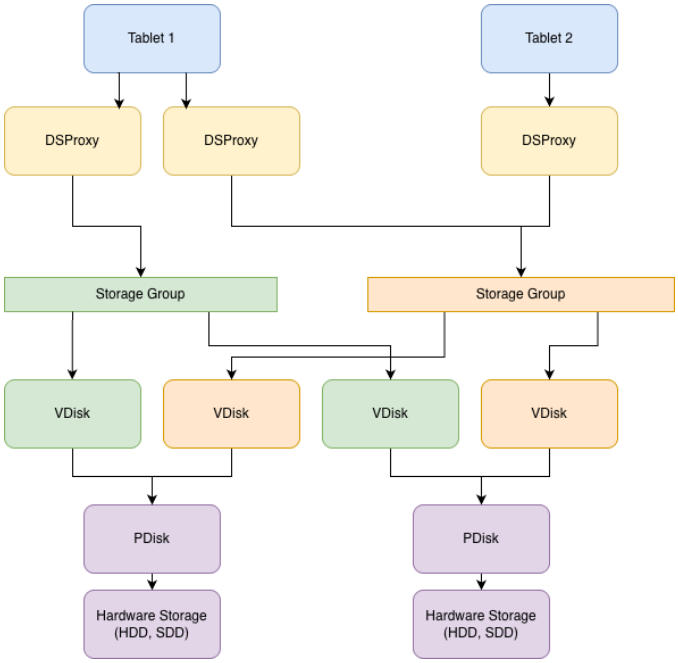
Автоматическая балансировка



YDB evenly distributes tablets among the nodes

YDB равномерно распределяет таблетки среди доступных узлов. Она перемещает тяжело загруженные таблетки с перегруженных узлов. Метрики CPU, памяти и сети отслеживаются для облегчения этого процесса.

Внутреннее устройство распределенного хранилища

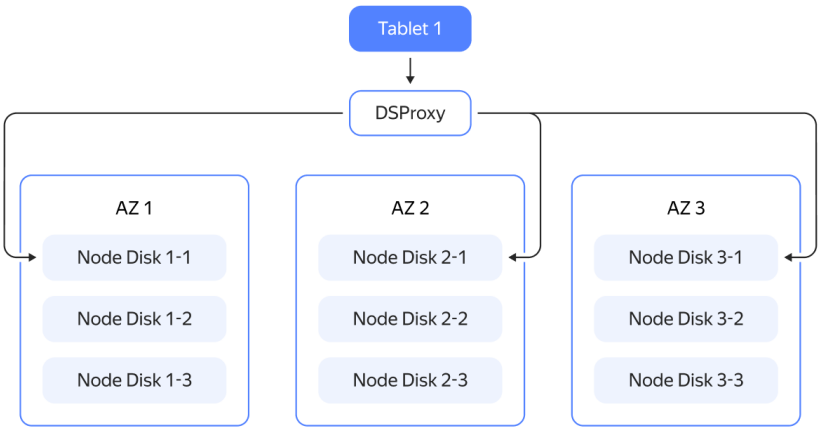


YDB не полагается на сторонние файловые системы. Вместо этого она работает непосредственно с дисковыми накопителями как с блочными устройствами. Поддерживаются различные типы дисков: NVMe, SSD и HDD — подробнее об этом можно узнать в системных требованиях. За работу с конкретным блочным устройством отвечает компонент PDisk, который взаимодействует непосредственно с физическим диском. Уровень абстракции выше PDisk называется VDisk.

Между VDisk и компонентами, непосредственно работающими с данными приложений, существует еще один слой абстракции — Storage Group. Storage Group объединяет несколько VDisk'ов между собой и обеспечивает необходимые режимы записи данных для отказоустойчивости.

Входной точкой в распределённое хранилище является DS-Proxy (distributed storage proxy). DS-Proxy скрывает распределённую природу хранилища и позволяет таблеткам записывать и читать данные, не заботясь о деталях взаимодействия с VDisk и Storage Group.

Прокси распределенного хранилища (DSProxy)



YDB writes data to 3 Availability Zones

Геораспределенная и отказоустойчивая конфигурация YDB обычно охватывает 3 датацентра или зоны доступности (Availability Zone - AZ). Когда YDB записывает данные на 3 зоны доступности, он не отправляет запросы на явно некорректные диски и продолжает работать без прерываний даже если одна зона доступности и диск в другой зоне доступности потеряны.

Управление ресурсами баз данных

По умолчанию выделяемые каждой базе данных вычислительные ресурсы используются всеми запросами к ней на равных условиях, без каких-либо ограничений. Если требуется разделить ресурсы внутри одной базы данных, то в YDB существуют пулы ресурсов и классификаторы пулов ресурсов. Один из типовых сценариев применения такой функциональности — это разделение контуров онлайн-вычислений и фоновой аналитики. Подробнее о ней можно узнать в статье Workload Manager — управление потреблением ресурсов.

Что дальше?

Если вас интересует более подробная информация о различных аспектах YDB, ознакомьтесь с соседними статьями в данном разделе документации. Если вы готовы к более практическому контенту, вы можете перейти к быстрому старту или курсу по YQL.

Архитектурный обзор YDB

Введение

YDB — это горизонтально масштабируемая, распределённая и отказоустойчивая универсальная система управления базами данных (СУБД), разработанная для обеспечения высокой производительности: например, типичный узел кластера способен обрабатывать десятки тысяч запросов в секунду. Система поддерживает георазпределённые (кросс-датацентровые) конфигурации — от небольших кластеров с несколькими узлами до кластеров из тысяч серверов, способных эффективно обрабатывать сотни петабайт данных.

Ключевые особенности и возможности YDB

- **Горизонтальное масштабирование и автоматическое партицирование**: данные и нагрузка распределяются динамически при росте объёма или интенсивности запросов.
- **Отказоустойчивость**: автоматическое восстановление при сбоях узлов, стоек и зон доступности.
- **Высокая доступность и сохранность данных**: обеспечивается с помощью автоматической синхронной репликации данных в пределах кластера.
- **Строгая консистентность и поддержка ACID-транзакций**: система обеспечивает распределённые транзакции с уровнем изоляции *serializable*. При необходимости уровень изоляции и строгость согласованности могут быть ослаблены для повышения производительности.
- **Язык запросов YQL** — SQL-диалект, оптимизированный для больших данных и сложных сценариев обработки.
- **Реляционная модель данных** с поддержкой строковых и колоночных таблиц, что делает возможным эффективное использование YDB как для транзакционных (OLTP), так и для аналитических (OLAP) нагрузок.
- **Иерархия объектов**: таблицы, топики и другие объекты организуются в иерархическое пространство имён, аналогично файловой системе.
- **Асинхронная репликация**: возможность настроить синхронизацию данных между базами YDB почти в реальном времени — как внутри одного кластера, так и между разными.
- **Потоковая обработка и распространение данных**:
  - **Топики** — хранение и потоковая доставка неструктурированных сообщений множеству подписчиков; поддерживают, в том числе, работу по протоколу Kafka.
  - **Change Data Capture (CDC)** — встроенная поддержка потокового чтения изменений данных из таблиц.
  - **Трансферы** — автоматическая доставка данных из топиков в таблицы.
- **Федеративные запросы** — выполнение запросов к внешним источникам данных (например, S3) в составе YQL-запросов без предварительной загрузки внешних данных в базу данных YDB.
- **Векторные индексы** — поддержка хранения и поиска по векторным представлениям данных для задач семантического поиска, сходства и ML-сценариев.
- **Наблюдаемость** — встроенная поддержка метрик, логов и дашбордов.
- **Безопасность и аудит** — шифрование данных (at-rest и in-transit), аудит операций, поддержка аутентификации и авторизации; подробнее — в разделе Безопасность.
- **Инструменты взаимодействия и интеграции** — доступны YDB CLI для администрирования и отладки, а также SDK для C++, C#, Go, Java, JavaScript, PHP, Python и Rust; поддерживается интеграция с внешними системами; подробнее — в разделах Интеграции YDB и Языки и API.
- **Открытая архитектура**: исходный код доступен под Apache License 2.0, взаимодействие с системой производится через открытый протокол gRPC, что позволяет реализовывать клиенты на любом языке программирования.

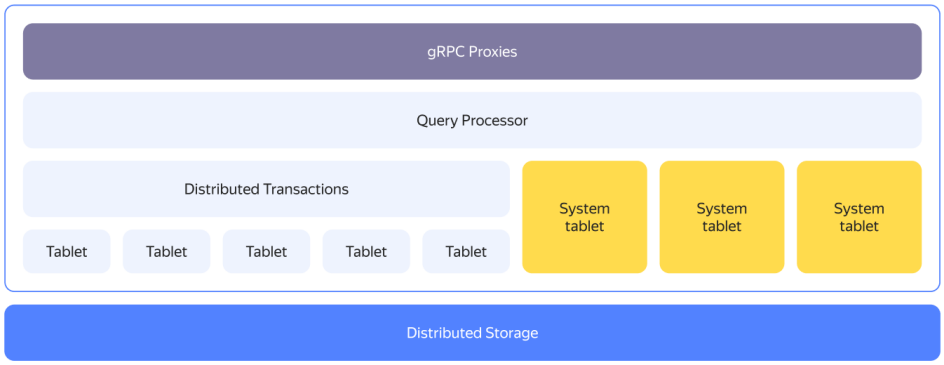
Основные сценарии применения

YDB — это универсальная платформа, подходящая для широкого спектра задач, где требуются масштабируемость, надёжность и гибкость. Основные сценарии:

- В распределённых системах, где требуется **строгая консистентность или поддержка транзакций**, включая изменение данных в нескольких строках и таблицах. YDB сочетает масштабируемость, характерную для NoSQL-систем, с возможностями реляционной СУБД по управлению согласованностью и целостностью данных.
- В системах, требующих хранения и обработки **очень большого объёма данных** с возможностью практически неограниченного горизонтального масштабирования (в эксплуатации находятся промышленные кластеры более чем из 5000 узлов, обрабатывающие нагрузку в миллионы запросов в секунду и хранящие петабайты данных).
- В высоконагруженных системах, где применяется **ручное шардирование** реляционных баз данных для распределения нагрузки. YDB упрощает архитектуру, устраняя необходимость в сложных шаблонах шардирования и ручной оркестрации.
- При разработке новых продуктов, для которых отсутствует надёжный прогноз нагрузки, или ожидается масштаб, превышающий возможности традиционных реляционных СУБД.
- В проектах, где требуется **гибкая платформа**, способная поддерживать разнообразные типы рабочих нагрузок и сценариев, включая транзакционные, потоковые и аналитические.

Как это работает?

Полное объяснение того, как работает YDB, получилось бы слишком объемным. Ниже вы можете ознакомиться с несколькими основными моментами, а затем продолжить изучение документации, чтобы узнать больше.

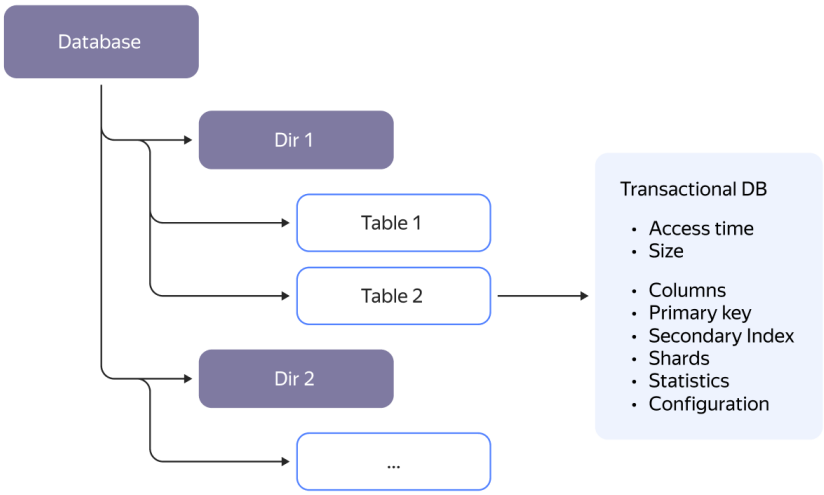


Кластеры YDB обычно работают с shared nothing архитектурой на обычном оборудовании. Уровни вычислений и хранения являются разнёсенными. Они могут работать как на отдельных наборах узлов, так и быть совмещёнными.

Один из ключевых элементов вычислительного слоя YDB называется *таблеткой*. Они являются логическими компонентами с состоянием, реализующими различные аспекты YDB.

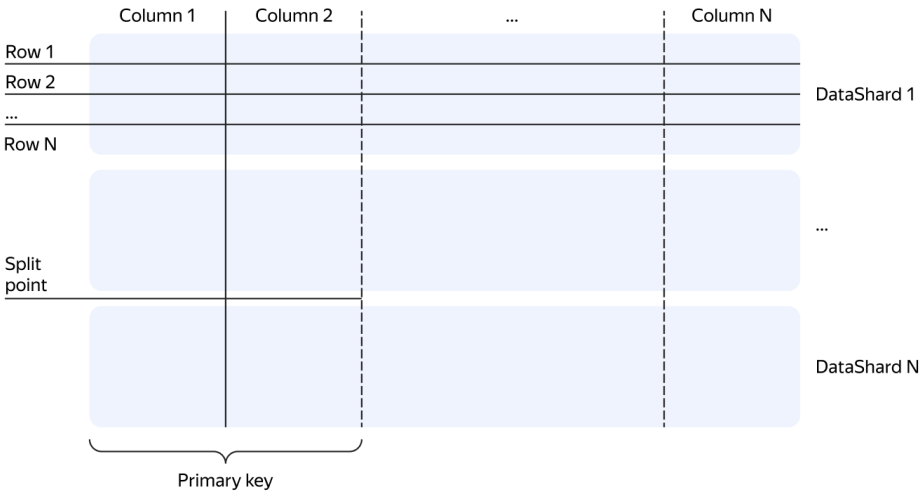
Более подробная общая архитектура YDB объясняется в разделе Общая схема YDB.

Иерархия



С точки зрения пользователя, всё внутри YDB организовано в иерархической структуре с использованием каталогов. Она может иметь произвольную глубину в зависимости от того, как вы решили организовать свои данные и проекты. Хотя YDB не имеет фиксированной глубины иерархии, как в других реализациях SQL, она все равно будет знакома, поскольку именно так выглядит любая виртуальная файловая система.

Таблица



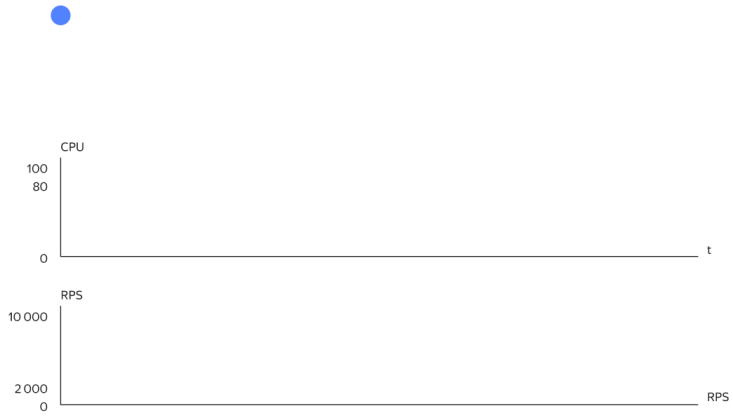
YDB предоставляет пользователям хорошо известную абстракцию — таблицы. В YDB существует два основных типа таблиц:

- Строковые таблицы предназначены для OLTP-нагрузок.
- Колоночные таблицы предназначены для OLAP-нагрузок.

Логически, с точки зрения пользователя, оба вида таблиц выглядят одинаково. Основное отличие между строковыми и колоночными таблицами заключается в способе хранения данных. В строковых таблицах значения всех колонок каждой строки располагаются рядом, а в колоночных таблицах — наоборот, каждая колонка хранится отдельно, и рядом оказываются ячейки, относящиеся к разным строкам.

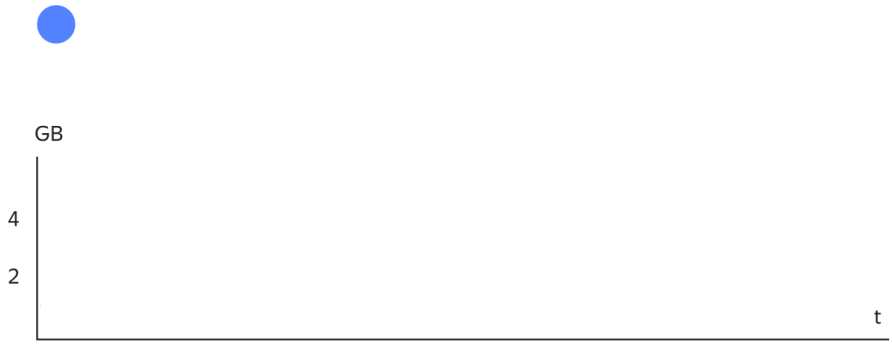
Независимо от типа таблицы, она должна содержать первичный ключ. В первичном ключе колоночных таблиц можно использовать только колонки с `NOT NULL`. Данные в таблицах физически сортируются по первичному ключу. Строковые таблицы автоматически партицируются по диапазонам первичных ключей в зависимости от объёма данных, а данные в колоночных таблицах партицируются не по первичному ключу, а по хешу от колонок партицирования. Каждый диапазон первичных ключей таблицы обрабатывается определённой таблеткой, называемой *data shard* для строчных таблиц и *column shard* — для колоночных.

Разделение по нагрузке



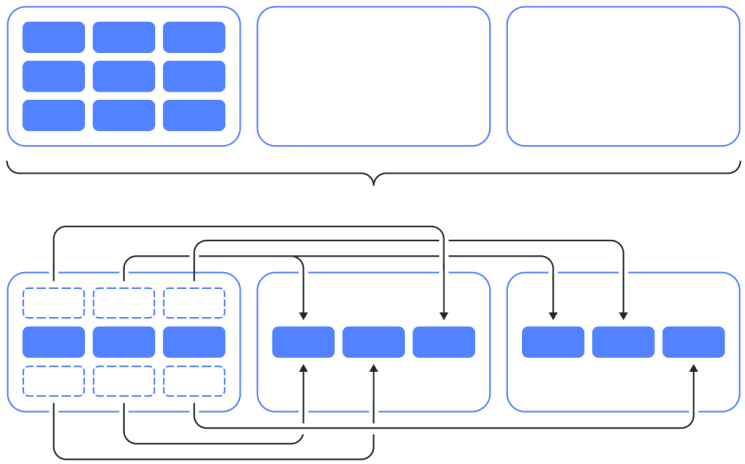
Data shard автоматически разделяются на большее количество при увеличении нагрузки. Они автоматически объединяются в нужное количество, когда пиковая нагрузка уходит.

Разделение по размеру



Data shard также автоматически разделяются при увеличении размера данных. Они автоматически сливаются обратно, если достаточное количество данных будет удалено.

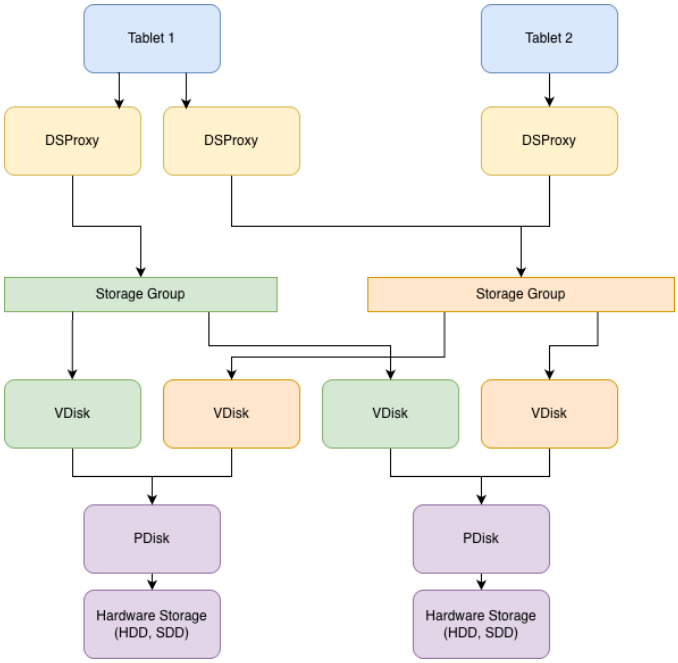
Автоматическая балансировка



YDB evenly distributes tablets among the nodes

YDB равномерно распределяет таблетки среди доступных узлов. Она перемещает тяжело загруженные таблетки с перегруженных узлов. Метрики CPU, памяти и сети отслеживаются для облегчения этого процесса.

Внутреннее устройство распределенного хранилища

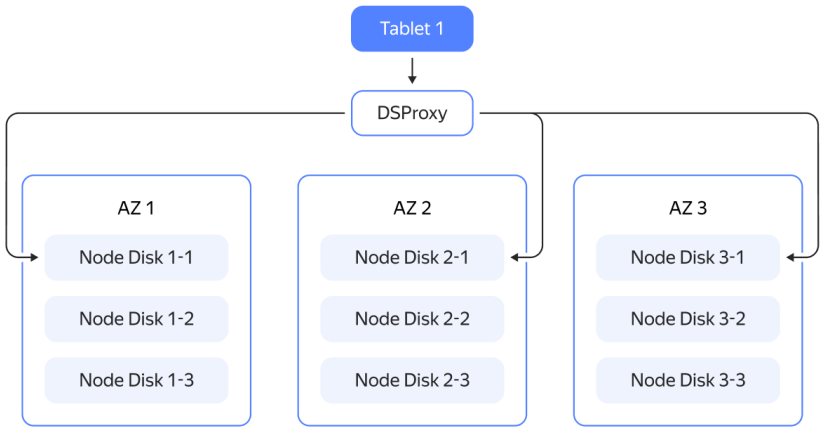


YDB не полагается на сторонние файловые системы. Вместо этого она работает непосредственно с дисковыми накопителями как с блочными устройствами. Поддерживаются различные типы дисков: NVMe, SSD и HDD — подробнее об этом можно узнать в системных требованиях. За работу с конкретным блочным устройством отвечает компонент PDisk, который взаимодействует непосредственно с физическим диском. Уровень абстракции выше PDisk называется VDisk.

Между VDisk и компонентами, непосредственно работающими с данными приложений, существует еще один слой абстракции — Storage Group. Storage Group объединяет несколько VDisk'ов между собой и обеспечивает необходимые режимы записи данных для отказоустойчивости.

Входной точкой в распределённое хранилище является DS-Proxy (distributed storage proxy). DS-Proxy скрывает распределённую природу хранилища и позволяет таблеткам записывать и читать данные, не заботясь о деталях взаимодействия с VDisk и Storage Group.

Прокси распределенного хранилища (DSProxy)



YDB writes data to 3 Availability Zones

Георасределенная и отказоустойчивая конфигурация YDB обычно охватывает 3 датацентра или зоны доступности (Availability Zone - AZ). Когда YDB записывает данные на 3 зоны доступности, он не отправляет запросы на явно некорректные диски и продолжает работать без прерываний даже если одна зона доступности и диск в другой зоне доступности потеряны.

Управление ресурсами баз данных

По умолчанию выделяемые каждой базе данных вычислительные ресурсы используются всеми запросами к ней на равных условиях, без каких-либо ограничений. Если требуется разделить ресурсы внутри одной базы данных, то в YDB существуют пулы ресурсов и классификаторы пулов ресурсов. Один из типовых сценариев применения такой функциональности — это разделение контуров онлайн-вычислений и фоновой аналитики. Подробнее о ней можно узнать в статье Workload Manager — управление потреблением ресурсов.

Что дальше?

Если вас интересует более подробная информация о различных аспектах YDB, ознакомьтесь с соседними статьями в данном разделе документации. Если вы готовы к более практическому контенту, вы можете перейти к быстрому старту или курсу по YQL.

Обзор сервисов распространения метаданных

Сервисы распространения метаданных — это три взаимосвязанные подсистемы кластера YDB, которые доставляют служебную информацию между узлами: **StateStorage**, **Board** и **SchemeBoard**. Все они построены на распределённом кворумном сервисе с детерминированным размещением реплик.

Эта статья даёт обзор назначения сервисов без погружения во внутренние идентификаторы и механизмы ядра. Подробное описание для разработчиков ядра — в разделе Подсистемы распространения метаданных. Инструкции по конфигурированию — в Конфигурирование подсистем распространения метаданных.

Зачем нужны сервисы распространения метаданных

Кластер YDB — распределённая система, в которой могут одновременно работать миллионы таблеток на тысячах узлов. Компонентам кластера нужно знать:

- где работает лидер конкретной таблетки и как к нему обратиться (**StateStorage**);
- какие узлы предоставляют сервисы, например точки подключения для клиентов (**Board**);
- какова актуальная схема базы данных (**SchemeBoard**).

Распространять эти данные с одного узла кластера плохо — это создает высокую нагрузку и проблемы при выходе из строя этого узла. Поэтому метаданные распределяются по множеству узлов кластера с помощью трёх специализированных подсистем.

StateStorage

StateStorage хранит актуальное состояние таблеток: кто сейчас лидер, поколение и шаг выбора лидера, список реплик. По id таблетки через этот сервис можно узнать текущий id актора, что позволяет взаимодействовать с ним через акторную систему.

> **Примечание**
>
> Данные в StateStorage **волатильны**: они хранятся в памяти реплик и восстанавливаются при перезапуске. Это не долговременное хранилище.

Board

Board — сервис публикации и подписки на метаданные в формате «путь → набор записей». Основное применение — хранение эндпоинтов баз данных: узлы публикуют адреса, клиенты и другие компоненты подписываются на изменения.

SchemeBoard

SchemeBoard распространяет метаданные схемы: таблицы, индексы, топики, права доступа. Узлы баз данных используют его как кеш схемы, чтобы не обращаться к SchemeShard при каждом запросе.

Сравнение сервисов

Характеристика	StateStorage	Board	SchemeBoard
Назначение	Состояние таблеток и лидерство	Публикация служебных метаданных	Распространение схемы
Тип данных	Состояние лидера таблетки	Пары путь → полезная нагрузка	Описания схемных объектов
Ключ	Идентификатор таблетки	Путь (строка)	Путь к объекту схемы
Основные потребители	Компоненты взаимодействия с таблетками	gRPC-прокси, клиенты	Узлы баз данных

Общий принцип

Все три сервиса используют один архитектурный подход: запись адресуется по ключу, набор реплик для ключа вычисляется детерминированно, операции выполняются по кворуму. Это обеспечивает масштабируемость и отказоустойчивость при роллинг-рестартах и выходе из строя стоек.

Детали (кольца реплик, кворум, смена конфигурации, размещение по доменам отказа) описаны в статье Подсистемы распространения метаданных.

Связанные материалы

- Подсистемы распространения метаданных — подробное описание для контрибьюторов ядра.
- Конфигурирование подсистем распространения метаданных.
- SelfHeal подсистем распространения метаданных.
- Режим bridge.
- Топология кластера.
- Глоссарий.

Соединение с БД

Для соединения с БД YDB из YDB CLI или приложения, использующего YDB SDK, необходимо указать эндпоинт и путь базы данных.

Эндпоинт

Эндпоинт (endpoint) - строка в формате `protocol://host:port`, предоставляемая владельцем кластера YDB для корректной маршрутизации запросов клиента к его базам данных через сетевую инфраструктуру и установки сетевого соединения. Для облачных баз данных endpoint показывается в консоли управления на странице требуемой БД, а также обычно может быть получен через CLI облачного поставщика. В корпоративных окружениях имена эндпоинтов YDB предоставляются командой администраторов или также могут быть получены в консоли управления внутренним облаком.

Примеры:

- `grpc://localhost:7135` — протокол обмена данными без шифрования (gRPC), сервер запущен на том же хосте что и клиент, принимает соединения на порту 7135.
- `grpcs://ydb.example.com` — протокол обмена данными с шифрованием (gRPCs), сервер запущен на хосте ydb.example.com в изолированной корпоративной интрасети, принимает соединения на порту YDB по умолчанию 2135.

Путь базы данных

Путь базы данных (`database`) - строка, определяющая где на кластере YDB находится база данных, к которой адресован запрос. Имеет структуру пути к файлу, с использованием символа `/` в качестве разделителя. Начинается всегда с символа `/`.

На кластере YDB может быть развернуто множество баз данных, и их пути определяются конфигурацией кластера. Как и эндпоинт, для облачных баз данных `database` показывается в консоли управления на странице требуемой БД, а также обычно может быть получен через CLI облачного поставщика.

При использовании облачных решений базы данных на кластере YDB создаются и размещаются в режиме самообслуживания, не требуя участия владельца и администраторов кластера.

> **Важно**
>
> Приложения не должны как-либо интерпретировать количество и значение каталогов в `database`, так как они определяются конфигурацией кластера YDB. Например, при работе с сервисом YDB в Yandex.Cloud `database` сейчас имеет структуру `region_name/cloud_id/database_id`, однако этот формат может быть изменен в будущем для новых БД.

Примеры:

- `/ru-central1/b1g8skpblkos03malf3s/etn01q5ko6sh271beftr` — база данных Yandex.Cloud с идентификатором `etn01q3ko8sh271beftr` в облаке `b1g8skpbljhs03malf3s`, развернутая в регионе `ru-central1`.
- `/local` — база данных по умолчанию при самостоятельном развертывании с использованием Docker.

Строка соединения

Строка соединения (connection string) — это строка в формате URL, указывающая эндпоинт и путь к базе данных, с использованием следующего синтаксиса:

```
<endpoint>?database=<database>
```

Примеры:

```
grpc://localhost:7135?database=/local
grpcs://ydb.serverless.yandexcloud.net:2135?database=/ru-central1/b1g8skpblkos03malf3s/etn01q5ko6sh271beftr
```

Использование строки соединения является альтернативой раздельному указанию эндпоинта и пути к базе данных и может применяться в тех инструментах, которые поддерживают данный метод.

Корневой сертификат для TLS

При использовании протокола с шифрованием (gRPC over TLS, или grpcs) продолжение сетевого соединения возможно только в том случае, если клиент уверен в том, что ему действительно отвечает подлинный сервер, с которым он пытается установить соединение, а не кто-то посередине, перехвативший запрос в сети. Это обеспечивается проверкой через цепочки доверия, для работы которой на клиенте должен быть установлен т.н. корневой сертификат.

В состав операционных систем, на которых запускается клиент, уже входит набор корневых сертификатов от основных мировых центров сертификации. Однако, владелец кластера YDB может использовать свой центр сертификации, не связанный ни с одним из мировых, что часто встречается в корпоративных окружениях, и почти всегда применяется при самостоятельном развертывании кластеров с поддержкой шифрования на соединениях. В этом случае владелец кластера должен каким-либо образом передать свой корневой сертификат для использования на стороне клиента. Такой сертификат может быть установлен в хранилище сертификатов операционной системы, где запускается клиент (вручную пользователем или корпоративной командой администраторов ОС), или встроен в состав самого клиента (как это сделано для Yandex.Cloud в YDB CLI и SDK).

Схемные объекты YDB

В YDB поддерживается следующий перечень схемных объектов:

- Директории
- Таблицы
- Представления
- Топики
- Узлы координации
- Секреты
- Внешние источники данных
- Внешние таблицы

Схемные объекты размещаются в пространстве имен кластера и именуются согласно общим для всех схемных объектов правилам именования.

См. также

- Концепции YDB
- Топология кластера YDB

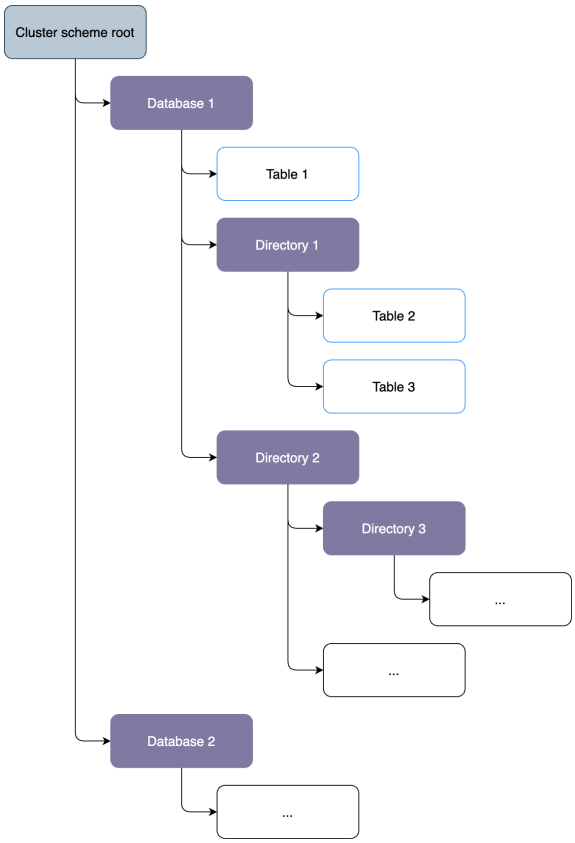
Пространство имён кластера и имена объектов

В YDB все схемные объекты организованы в иерархическое пространство имён — **схему кластера**. Эта схема определяет, как размещаются и именуются базы данных, директории и объекты, обеспечивая порядок и масштабируемость.

Схема кластера

Схема кластера — это иерархическая структура, в которой:

- **Корень схемы кластера** — верхнеуровневый элемент.
- На первом уровне находятся базы данных.
- Внутри базы данных можно создавать вложенные директории, формируя произвольную иерархию.
- Схемные объекты (таблицы, представления, индексы и др.) могут размещаться как в корне базы данных, так и внутри директорий.



Правила именования схемных объектов

Каждый схемный объект в YDB имеет имя. В YQL-выражениях имена схемных объектов указываются с помощью идентификаторов, заключённых в обратные кавычки (`) или без них. Для более подробной информации об идентификаторах, см. Ключевые слова и идентификаторы.

Имена схемных объектов в YDB должны соответствовать следующим требованиям:

- **Допустимые символы**:
  - латинские буквы (`A-Z`, `a-z`)
  - цифры (`0-9`);
  - специальные символы: `.`, `-` и `_`.
- **Максимальная длина**: 255 символов.
- **Ограничения на размещение**: объекты нельзя создавать в директориях, имена которых начинаются с точки (например, `.sys`, `.medatata`, `.sys_health`).

Директория

Для удобства организации поддерживается создание директорий по аналогии с файловой системой, то есть вся база состоит из дерева директорий, а схемные объекты, например, таблицы, находятся в листах этого дерева. В одной директории могут быть несколько поддиректорий и несколько схемных объектов. Имена схемных объектов внутри одной директории уникальны.

Таблица

Таблица — это реляционная таблица, содержащая набор связанных данных, состоящий из строк и столбцов. Таблицы описывают сущности. Например, статья в блоге может быть представлена таблицей `article` со столбцами: `id`, `date_create`, `title`, `author` и т.д. Строки содержат данные, а столбцы определяют типы данных. Например, столбец `id` не может быть пустым (`NOT NULL`) и должен содержать только уникальные целочисленные значения. Запись в YQL может выглядеть так:

```
CREATE TABLE article (
    id Int64 NOT NULL,
    date_create Date,
    author String,
    title String,
    PRIMARY KEY (id)
)
```

YDB поддерживает создание строковых и колоночных таблиц. Основное их отличие — в области применения и формате хранения данных на жестком диске. Для строковых таблиц данные хранятся последовательно в виде строк, а для колоночных — в виде столбцов. У каждого типа таблиц свое предназначение.

Правила наименования колонок (столбцов)

Имена колонок в YDB должны соответствовать следующим требованиям:

- Имя колонки может состоять из следующих символов:
  - прописные латинские буквы;
  - строчные латинские буквы;
  - цифры;
  - специальные символы: `-` и `_`.
- Длина имени колонки не должна превышать 255 символов.
- Имя колонки не должно начинаться с системного префикса `__ydb_`.

Строковые таблицы

Строковые таблицы хорошо подходят для транзакционных запросов, которые генерируют Online Transaction Processing (OLTP) системы, например, бэкенды сервисов погоды или интернет-магазинов. Строковые таблицы предоставляют эффективный доступ к большому количеству столбцов одновременно. Поиск по строковым таблицам работает очень быстро, за счет использования индексов.

Индекс — это структура данных, которая увеличивает скорость операций извлечения данных на основе одного или нескольких столбцов. Это аналог индекса в книге: вместо того чтобы просматривать каждую страницу книги, чтобы найти нужный раздел, вы можете обратиться к индексу в конце книги и быстро перейти к нужной странице.

Когда выполняется запрос на основе столбца (или столбцов), по которому (которым) создан индекс, СУБД может использовать этот индекс, чтобы быстро найти соответствующие строки, минуя полный перебор всех данных. Например, если у вас есть индекс на столбце "author", и вы ищете все статьи, написанные автором "Gray", СУБД использует индекс для быстрого нахождения всех строк с этой фамилией.

Создать строковую таблицу можно через web-интерфейс YDB, с помощью CLI или SDK. Вне зависимости от способа взаимодействия с YDB стоит помнить об общем правиле создания строковой таблицы: таблица должна иметь минимум одну ключевую колонку, при этом допускается создание таблицы, состоящей только из ключевых колонок.

По умолчанию при создании строковой таблицы все столбцы опциональны и могут иметь значения `NULL`, такое поведение можно изменить и задать условия `NOT NULL` для ключевых колонок, которые входят в состав первичного ключа. Первичные ключи уникальны, и строковые таблицы всегда упорядочены по ключу. Это означает, что точечное чтение по ключу, а также диапазонные запросы по ключу или префиксу ключа выполняются эффективно (фактически используя индекс). Допускается создание таблицы, состоящей только из ключевых столбцов. К выбору ключа нужно подходить аккуратно, поэтому рекомендуем ознакомиться со статьей: "Выбор первичного ключа для максимальной производительности".

Партиционирование строковой таблицы

Строковая таблица в БД может быть партиционирована по диапазонам значений первичного ключа. Каждая партиция таблицы отвечает за свой диапазон первичных ключей. Диапазоны ключей, обслуживаемых разными партициями, не пересекаются. Различные партиции таблицы могут обслуживаться разными серверами распределенной БД (в том числе расположенными в разных локациях), а также могут независимо друг от друга перемещаться между серверами для перебалансировки или поддержания работоспособности партиции при отказах серверов или сетевого оборудования.

При малом объеме данных или небольшой нагрузке таблица может состоять из одной партиции. При росте объема данных партиции или нагрузки на партицию, YDB автоматически разделяет её на две:

- Если объем данных превышает порог по размеру партиции, то разделение происходит по медианному значению первичного ключа.
- Если увеличилась нагрузка, то партиция сначала собирает сэмпл запрашиваемых ключей (читаемых, записываемых и удаляемых) и на основании этого сэмпла выбирает для разделения такой ключ, чтобы нагрузка распределилась поровну между новыми партициями. Таким образом в случае разделения по нагрузке новые партиции могут иметь существенно отличающийся размер.

Порог разделения партиции по размеру и включение/выключение автоматического разделения могут быть настроены индивидуально для каждой таблицы базы данных.

Независимо от значения параметра AUTO_PARTITIONING_PARTITION_SIZE_MB, YDB принудительно разделяет партиции, размер которых превышает параметр force_shard_split_data_size (по умолчанию 2 ГиБ). При этом YDB не ограничивает пользовательские настройки, а работает параллельно с ними. Например, если задать размер партиции 100 МиБ и ограничение на 10 партиций, то, когда у таблицы станет 10 партиций по 100 МиБ, она перестанет делиться, и размер партиций начнет увеличиваться. Когда размер какой-либо партиции превысит значение параметра `force_shard_split_data_size`, она разделится, и в таблице станет 11 партиций. Если суммарный размер двух соседних партиций уменьшится до 50% от значения параметра AUTO_PARTITIONING_PARTITION_SIZE_MB, они объединятся, и в таблице снова станет 10 партиций.

Помимо автоматического разделения, предоставляется возможность создать пустую таблицу с заданным количеством партиций. При этом можно вручную задать точные границы разделения ключей по партициям или указать равномерное

распределение на заданное количество партиций. В последнем случае границы будут созданы по первой компоненте первичного ключа. Равномерное распределение можно указать для таблиц, у которых первая компонента первичного ключа является целым числом с типом данных `Uint64` или `Uint32`.

Параметры партиционирования относятся только к самой таблице, но не к вторичным индексам, построенным на её данных. Каждый индекс обслуживается своим набором партиций, и решения о разделении или объединении его партиций принимаются независимо, на основании настроек по умолчанию. В будущем эти настройки могут стать доступными пользователям, аналогично настройкам основной таблицы.

Характерное время операции разделения или объединения составляет около 500 мс. В течение этого времени данные, вовлечённые в операцию, становятся кратковременно недоступны для чтения и записи. Специализированные методы-обёртки в YDB SDK автоматически выполняют повторные попытки при обнаружении, что партиция находится в состоянии разделения или объединения, не поднимая эту информацию до уровня приложения. Важно отметить, что если система по каким-либо причинам перегружена (например, из-за нехватки CPU или пропускной способности выделенных дисковых ресурсов), операции разделения и объединения могут занимать больше времени.

В схеме данных определяются следующие параметры партиционирования таблицы:

AUTO_PARTITIONING_BY_SIZE

- Тип: `Enum` (`ENABLED` , `DISABLED`).
- Значение по умолчанию: `ENABLED` .

Режим автоматического партиционирования по размеру партиции. Если размер партиции превысил значение, заданное параметром AUTO_PARTITIONING_PARTITION_SIZE_MB, то она встает в очередь на разделение (split). Если суммарный размер двух или более соседних партиций меньше 50% от значения параметра AUTO_PARTITIONING_PARTITION_SIZE_MB, то они встают в очередь на объединение (merge).

AUTO_PARTITIONING_BY_LOAD

- Тип: `Enum` (`ENABLED` , `DISABLED`).
- Значение по умолчанию: `DISABLED` .

Режим автоматического партиционирования по нагрузке. Если в течение нескольких десятков секунд партиция потребляет более 50% CPU, то она ставится в очередь на разделение (split). Если в течение часа суммарная нагрузка на две или более соседних партиции составляет менее 35% одного ядра CPU, то они ставятся в очередь на объединение (merge).

При принятии решения о разделении партиции по нагрузке или об объединении нескольких партиций по нагрузке, YDB рассматривает CPU нагрузку и на лидера данной партиции, и на все её реплики.

Выполнение операций разделения или объединения само по себе утилизирует CPU, и занимает время. Поэтому, при работе с плавающей нагрузкой рекомендуется вместе с включением данного режима устанавливать отличное от 1 значение параметра минимального количества партиций AUTO_PARTITIONING_MIN_PARTITIONS_COUNT, чтобы спады нагрузки не приводили к снижению количества партиций ниже необходимого, и не было потребности их заново делить при появлении нагрузки.

При выборе минимального количества партиций имеет смысл руководствоваться соображениями, что одна партиция таблицы может находиться только на одном сервере и использовать не более 1 ядра CPU для операций изменения данных. Исходя из этого, для таблицы на которой может ожидаться высокая нагрузка, можно указывать минимальное количество партиций не менее количества узлов (серверов), а лучше порядка количества ядер CPU, выделенных базе.

AUTO_PARTITIONING_PARTITION_SIZE_MB

- Тип: `Uint64` .
- Значение по умолчанию: `2000 MB` (`2 ГБ`).

Желаемый порог размера партиции в мегабайтах. Рекомендуемые значения варьируются от `10 MB` до `2000 MB` . Если этот порог превышается, партиция может быть разделена.
Указанное значение служит лишь рекомендацией для разделения. Возможны ситуации, когда разделение не произойдёт, даже если указанный размер превышен.
Эта настройка применяется, когда включён режим `AUTO_PARTITIONING_BY_SIZE` .

AUTO_PARTITIONING_MIN_PARTITIONS_COUNT

- Тип: `Uint64` .
- Значение по умолчанию: `1` .

Объединение партиций (merge) производится только в том случае, если их фактическое количество превышает заданное этим параметром значение. При использовании режима автоматического партиционирования по нагрузке рекомендуется устанавливать отличное от 1 значение данного параметра, чтобы периодический спад нагрузки не приводил к снижению количества партиций ниже необходимого.

AUTO_PARTITIONING_MAX_PARTITIONS_COUNT

- Тип: `Uint64` .
- Значение по умолчанию: `50` .

Разделение партиций (split) производится только в том случае, если их количество не превышает заданное этим параметром значение. При любых включенных режимах автоматического партиционирования рекомендуется устанавливать осмысленное значение этого параметра, а также отслеживать приближение фактического количества партиций к нему, иначе рано или поздно партиции перестанут разделяться при росте данных или нагрузке, что приведет к сбою.

UNIFORM_PARTITIONS

- Тип: `Uint64` .
- Значение по умолчанию: не применимо.

Количество партиций для равномерного начального разделения таблицы. Первая колонка первичного ключа должна иметь тип `Uint64` или `Uint32` . Созданная таблица сразу будет разделена на указанное количество партиций.

При включенном автоматическом партиционировании необходимо также не забыть установить корректное значение параметра AUTO_PARTITIONING_MIN_PARTITIONS_COUNT, чтобы все партиции не объединились в одну сразу после создания таблицы.

PARTITION_AT_KEYS

- Тип: `Expression`.
- Значение по умолчанию: не применимо.

Граничные значения ключей для начального разделения на партиции. Представляется списком граничных значений, разделенных запятыми и обрамленными в скобки. Каждое граничное значение может быть либо набором значений ключевых колонок (также разделенных запятыми и обрамленными в скобки), либо единичным значением, если указываются только значения первой ключевой колонки. Примеры: `(100, 1000)`, `((100, "abc"), (1000, "cde"))`.

При включенном автоматическом партиционировании необходимо также не забыть установить корректное значение параметра AUTO_PARTITIONING_MIN_PARTITIONS_COUNT, чтобы все партиции не объединились в одну сразу после создания таблицы.

Чтение с реплик

При выполнении запросов в YDB фактическое выполнение запроса к каждой партиции осуществляется в единой точке, обслуживающей протокол распределенных транзакций. Но благодаря хранению данных на разделяемом хранилище возможен запуск одной или нескольких реплик партиции, без выделения дополнительного места на сторадже — данные уже хранятся реплицированно и возможно обслуживание более одного читателя (но писатель при этом все еще в каждый момент строго один).

Применение чтения с реплик дает следующие возможности:

- Обслуживать клиентов, критичных к минимальным задержкам, недостижимым иными способами в мульти-ДЦ кластере. Достигается за счет приближения точки выполнения запроса к точке задания запроса, что отсекает задержку на меж-ДЦ пересылки. В результате, сохраняя все гарантии мульти-ДЦ кластера по надежности хранения, отвечать на точечные читающие запросы за единицы миллисекунд.
- Обслуживать читающие запросы с реплик без влияния на модифицирующие запросы, выполняющиеся на партиции. Это может быть полезно как для изоляции разных сценариев, так и для увеличения пропускной способности партиции.
- Продолжать обслуживание при переездах лидера партиции (как штатной при балансировке, так и при сбоях). Позволяет переживать процессы в кластере без влияния на читающих клиентов.
- В целом повышать предел производительности чтения партиций, если множество читающих запросов попадают на одни и те же ключи.

В схеме данных таблицы можно указать необходимость запуска реплик для чтения для каждой партиции таблицы. Обращения к репликам для чтения типично происходят не покидая сети датацентра, что позволяет обеспечить время ответа в единицы миллисекунд:

Имя параметра	Описание	Тип	Допустимые значения	Возможность изменения	Возможность сброса
`READ_REPLICAS_SETTINGS`	`PER_AZ` означает использование указанного количества реплик в каждом AZ, `ANY_AZ` — во всех AZ суммарно.	String	`"PER_AZ:<count>"`, `"ANY_AZ:<count>"` где `<count>` — число реплик. Чтобы удалить реплики, установите значение 0. Например: "PER_AZ:0".	Да	Нет

Внутреннее состояние каждой реплики восстанавливается в точности по состоянию лидера и полностью консистентно.

Кроме состояния данных в сторадже на реплики отправляется и поток обновлений с лидера. Обновления отправляются в реальном времени, сразу после коммита в лог. Но отправляются асинхронно, что приводит к некоторой задержке применения обновлений на репликах относительно их коммита на лидере (типично в десятки миллисекунд, но может увеличиваться в случае проблем связности в кластере). В связи с этим, чтение с реплик обеспечивается только в режиме транзакций `StaleReadOnly()`.

Если реплик несколько, то отставание их от лидера может различаться, т.е. хотя каждая реплика каждой из партиции сохраняет внутреннюю консистентность, между разными партициями могут наблюдаться артефакты. Код приложения должен быть к этому готов. По этой же причине на данный момент невозможно и выполнение с реплик кроссшардовых транзакций.

Автоматическое удаление устаревших данных (TTL)

YDB поддерживает функцию автоматического фонового удаления устаревших данных. В схеме данных таблицы может быть определена колонка подходящего типа, значение которой у всех строк будет сравниваться в фоновом режиме с текущим временем. Строки, для которых текущее время стало больше, чем значение в колонке с учетом заданной задержки, будут удалены.

Имя параметра	Тип	Допустимые значения	Возможность изменения	Возможность сброса
`TTL`	Expression	`Interval("<literal>") ON <column> [AS <unit>]` или `Interval("literal1") action1, ..., Interval("literal1") action1 ON <column> [AS <unit>]`	Да	Да

Синтаксис значения TTL описан в статье Time to Live (TTL) и вытеснение данных во внешнее хранилище. Подробнее об удалении устаревших данных читайте в разделе Time to Live (TTL).

Переименование

YDB позволяет переименовать существующую таблицу, переместить ее в другую директорию этой же базы данных, а также заменить одну таблицу другой, при этом данные заменяемой таблицы будут удалены. При выполнении операций изменяются только метаданные таблицы, например ее путь и имя. Данные таблицы не переносятся и не перезаписываются.

Операции выполняются изолированно, внешний процесс видит только два состояния таблицы: до и после выполнения операции. Это важно, например, при замене таблицы — данные заменяемой таблицы удаляются в той же транзакции, в которой заменяющая таблица переименовывается. Во время выполнения операции замены возможны ошибки запросов к заменяемой таблице с retryable-статусами.

Скорость выполнения переименования определяется типом дата-транзакций, которые выполняются в данный момент на таблице, и не зависит от количества данных в таблице.

- Переименование таблицы в YQL

- Переименование таблицы через CLI

Локальные блум-индексы

Использование фильтра Блума позволяет эффективнее определять отсутствие ключей в таблице при множественных точечных запросах по первичному ключу, снижая количество необходимых операций ввода-вывода с диска, ценой увеличения объема потребляемой памяти.

Рекомендуемый способ управления фильтрами Блума в строковых таблицах — через локальные Блум-индексы (`LOCAL USING bloom_filter`), создаваемые с помощью ALTER TABLE ... ADD INDEX и удаляемые с помощью ALTER TABLE ... DROP INDEX. Подробнее см. Блум-индексы.

Важно

Имя параметра	Тип	Допустимые значения	Возможность изменения	Возможность сброса
`KEY_BLOOM_FILTER`	Enum	`ENABLED`, `DISABLED`	Да	Нет

Настройка `KEY_BLOOM_FILTER` устарела (deprecated). Она включает фильтр Блума по всему первичному ключу, а при установке `DISABLED` очищает все фильтры Блума на таблице.

Группы колонок

YDB позволяет группировать колонки в таблице для оптимизации их хранения и использования. Механизм групп колонок позволяет увеличить производительность операций неполного чтения строк путем разделения хранения колонок таблицы на несколько групп. Наиболее часто используемый сценарий — организация хранения редко используемых атрибутов в отдельной группе колонок. Тогда можно включить сжатие данных и/или хранить их на более медленных накопителях.

У каждой группы колонок есть собственное имя, уникальное в рамках таблицы. Состав групп колонок устанавливается при создании таблицы, а также может быть изменен позднее. Удаление групп колонок из существующей таблицы не предусмотрено.

Группа колонок может содержать произвольное число колонок своей таблицы. Каждая колонка таблицы принадлежит одной, и только одной группе колонок (группы колонок не пересекаются).

Для каждой таблицы существует основная группа колонок с именем `default`, в которую входят все колонки, для которых не указана явным образом другая группа колонок. Колонки первичного ключа всегда принадлежат основной группе колонок и не могут быть перемещены в другую группу. Также, если нет других групп колонок, то изменение атрибутов основной группы колонок позволяет задавать атрибуты для всей таблицы.

Для группы колонок устанавливаются атрибуты, влияющие на хранение и кэширование данных:

- используемый тип устройств хранения данных (SSD или HDD, доступность типов зависит от конфигурации кластера YDB);
- режим сжатия данных (без сжатия или сжатие алгоритмом LZ4);
- режим кэширования.

Атрибуты группы колонок устанавливаются при создании таблицы, а также могут быть изменены впоследствии. Изменения атрибутов хранения не применяются немедленно к хранимым данным, вместо этого они применяются при последующем фоновом LSM compaction. Изменения атрибутов кэширования применяются сразу.

Доступ к данным, хранящимся в полях основной группы колонок, является более быстрым и требует меньше ресурсов, чем доступ к данным той же самой строки таблицы, хранящимся в полях дополнительно созданных групп колонок. Поиск по первичному ключу всегда осуществляется в основной группе колонок. При обращении к полям в других группах колонок помимо поиска по первичному ключу необходимы дополнительные поисковые операции для определения конкретной позиции хранения этих полей. Изменение режима кэширования не влияет на необходимость дополнительных операций для неосновных колонок, поэтому при одинаковых режимах кэширования для основной группы колонок доступ тоже будет быстрее.

Таким образом, вынос части колонок таблицы в отдельную группу колонок позволяет ускорить чтение наиболее важных и часто используемых колонок (входящих в состав основной группы колонок) ценой некоторого замедления доступа к остальным колонкам. Кроме того, на уровне групп колонок осуществляется управление параметрами хранения данных — выбор типа устройств хранения и режима сжатия, а также параметров кэширования.

Режимы кэширования

Для групп колонок можно задавать режимы кэширования. Режимы кэширования в YDB позволяют управлять стратегией кэширования данных таблиц в оперативной памяти. Это особенно важно для оптимизации времени ответа при чтении небольших часто используемых таблиц. Поддержаны два режима кэширования:

- `regular` (значение по умолчанию);
- `in_memory`.

В режиме кэширования `regular` данные групп колонок загружаются в общий кэш только при первом обращении к ним. Если группа колонок использует режим `in_memory`, её данные автоматически предзагружаются в кэш при запуске системы.

Когда суммарный объём данных в кэше превышает установленный лимит, сначала освобождается память, занятая страницами с режимом `regular`. Если этого недостаточно, вытесняются страницы с режимом `in_memory`. Таким образом, режим `in_memory` даёт приоритетное хранение в оперативной памяти, но размер кэша остаётся ограниченным конфигурацией — память не будет использоваться неограниченно.

Режим кэширования `in_memory` может быть полезен в различных сценариях работы. Один из частых сценариев — хранение небольших справочных таблиц (“lookup-таблиц”), которые полностью помещаются в оперативную память. В этом случае использование режима `in_memory` позволяет минимизировать задержки при доступе к данным, так как таблица всегда будет находиться в памяти, даже если обращения к ней происходят нерегулярно.

Однако это не единственный способ применения. Например, в некоторых случаях может быть целесообразно хранить в памяти все данные конкретной таблицы, независимо от её назначения, если бизнес-логика приложения требует постоянного быстрого доступа к полной версии таблицы.

Используя режим `in_memory`, помните, что выделенная на такие таблицы память уменьшает объём, доступный для кэша обычных таблиц, что может негативно сказаться на их производительности (увеличить latency).

Следует отметить, что режим кэширования `in_memory` влияет только на операции чтения. Операции записи при любом режиме пишутся на диск, тем самым гарантии сохранности данных остаются без изменений. Это отличает режимы кэширования YDB от некоторых других in-memory СУБД, которые могут полностью содержать таблицы в памяти ценой снижения гарантий на сохранность данных.

Пользовательские атрибуты

Пользовательские атрибуты позволяют добавлять произвольную информацию к метаданным таблицы. Эта информация не интерпретируется сервером, однако может быть интерпретирована клиентом БД (человеком или, чаще, программой).

Атрибуты задаются в виде пары ключ-значение. Ключом и значением атрибута может быть только строка или тип, который может быть представлен строкой (например, используя base64-кодирование).

Ключи и значения пользовательских атрибутов имеют следующие ограничения:

- длина ключа — 1–100 байт;
- длина значения — 1–4096 байт;
- максимальный общий размер атрибутов (сумма длин всех ключей и значений) — 10240 байт.

О том, как добавить, изменить, получить текущие значения или удалить атрибуты читайте в статье Пользовательские атрибуты таблиц.

Колоночные таблицы

Колоночные таблицы YDB хранят данные каждого столбца отдельно (независимо) от других столбцов. Такой принцип хранения данных оптимизирован для работы с Online Analytical Processing-нагрузками (OLAP), так как при выполнении запроса считываются только те столбцы, которые непосредственно участвуют в запросе. Еще один плюс такого подхода — высокая степень сжатия данных, так как в столбцах зачастую хранятся повторяющиеся или близкие данные. Минусом является то, что выполнение операций над строками становится более затратным.

На данный момент основной сценарий использования колоночных таблиц YDB — запись данных с возрастающим первичным ключом (например, время события), анализ этих данных и удаление устаревших данных по TTL. Оптимальным способом добавления данных в колоночные таблицы YDB является пакетная запись, выполняемая блоками в единицы МБ. Вставка пакетов данных производится атомарно: данные будут записаны или во все партиции, или ни в одну.

В большинстве случаев работа с колоночными таблицами YDB аналогична работе со строковыми, но есть и отличия:

- В качестве первичного ключа можно использовать только `NOT NULL` колонки.
- Данные партицируются не по первичному ключу, а по Hash от колонок партицирования.
- Поддерживается возможность задавать сжатие индивидуально для каждой колонки при создании колоночной таблицы или изменить её позднее.
- Поддерживается возможность задавать кодирование данных индивидуально для каждой колонки при создании колоночной таблицы или изменить её позднее.

Повторим создание таблицы `article`, на этот раз в колоночной форме, с помощью следующей YQL-команды:

```
CREATE TABLE article_column_table (
    id Int64 NOT NULL,
    author String,
    title String,
    PRIMARY KEY (id)
)
WITH (STORE = COLUMN);
```

Локальные блум-индексы

В колоночных и строковых таблицах можно задавать локальные Блум-индексы по колонкам: `LOCAL USING bloom_filter` или `LOCAL USING bloom_ngram_filter`. Индексы создаются при создании таблицы или добавляются через ALTER TABLE ADD INDEX. Подробнее: локальные индексы, Блум-индексы.

Локальный min_max-индекс

В колоночных таблицах можно задавать локальный min_max-индекс по колонкам с помощью `LOCAL USING min_max`. Индекс задается при создании таблицы или добавляется через ALTER TABLE ADD INDEX. Подробнее: локальные индексы, min_max-индекс.

Партицирование колоночной таблицы

В отличие от строковых таблиц YDB, колоночные таблицы партицируют данные не по первичным ключам, а по специально выделенным ключам — ключам партицирования. Ключи партицирования являются подмножеством первичных ключей таблицы.

В отличие от партицирования данных в строковых таблицах YDB, партицирование данных для колоночных таблиц выполняется не по значениям ключей, а по hash-значениям от ключей, что позволяет равномерно распределить данные во все существующие партиции. Такое партицирование позволяет избежать хотспотов при вставке и ускоряет аналитические запросы, обрабатывающие (считывающие) большие объемы данных.

Выбор ключей партицирования существенно влияет на производительность колоночных таблиц. Подробнее смотрите: "Выбор первичного ключа для максимальной производительности колоночных таблиц".

Для управления партицированием данных используется дополнительный параметр партиционирования: AUTO_PARTITIONING_MIN_PARTITIONS_COUNT. Остальные параметры партицирования для колоночных таблиц игнорируются.

AUTO_PARTITIONING_MIN_PARTITIONS_COUNT определяет минимальное физическое количество партиций для хранения данных.

Тип: `Uint64`.

Значение по умолчанию: 64.

С учетом, что остальные параметры партицирования игнорируются, это же значение определяет и верхнее число партиций.

Ограничения колоночных таблиц

В настоящий момент реализована не вся функциональность колоночных таблиц. Сейчас не поддерживается:

- Чтение с реплик.
- Глобальные вторичные индексы.
- Векторные и полнотекстовые индексы.
- Фильтр Блума для первичного ключа (`KEY_BLOOM_FILTER`; см. фильтр Блума для строковых таблиц).
- Change Data Capture.
- Пользовательские атрибуты таблиц.

Представления (VIEW)

Представление — это способ сохранить запрос и обращаться к его результатам как к настоящей таблице. Само представление не хранит данных, кроме текста запроса. Запрос, хранящийся в представлении, выполняется при каждом `SELECT` из него, генерируя возвращаемый результат. Любые изменения в таблицах, на которые ссылается представление, немедленно отражаются в результатах чтения из него.

Представления обычно используются для:

- сокрытия сложного запроса за понятным интерфейсом таблицы;
- точечного управления доступом к данным таблицы;
- реализации стабильного интерфейса к таблице, чья схема может меняться.

Важно

В настоящий момент представления в YDB невозможно использовать для предоставления частичного доступа к таблице другим пользователям. Запрос, хранимый в представлении, выполняется от имени пользователя представления, поэтому для чтения из представления пользователю также нужны и права на чтение из подлежащих таблиц. Больше информации по этой теме можно найти на странице `CREATE VIEW` в описании опции `security_invoker`.

Инвалидация представления

Если удалить одну из таблиц, на которые ссылается представление, то оно станет невалидным. Запросы на чтения из него будут возвращать ошибку точно так же как если бы вы попытались выполнить записанный в представление запрос непосредственно. Зависимости между представлениями и таблицами на данный момент не отслеживаются. Чтобы вернуть такое представление обратно в валидное состояние, достаточно создать (или восстановить) таблицу с тем же именем, которое было у той таблицы, на которое ссылается представление в своём запросе. В этом смысле чтение из представления аналогично чтению из подзапроса. Единственное различие между чтением из подзапроса и чтением из представления заключается в том, что контекст исполнения запроса, сохранённого в представлении, отличается от контекста объемлющего запроса, читающего из него.

Время выполнения запросов, читающих из представления

Выполнение запроса можно разбить на 2 стадии:

1. компиляция,
2. исполнение скомпилированного кода.

При компиляции запросов с участием представлений все упоминания представлений разворачиваются в текст запросов, хранящихся в них. Это означает, что время исполнения скомпилированного кода (стадия №2 выполнения запроса) для запросов с участием представлений и прямых запросов должно всегда совпадать, потому что исполняемый код в обоих случаях генерируется одинаковый.

Что касается времени компиляции (стадия №1 выполнения запроса) запросов с участием представлений против времени компиляции аналогичных запросов, но записанных непосредственно, то оно отличается в большую сторону, но незначительно. Это происходит потому, что чтение из представления:

```
SELECT * FROM a_view;
```

проходит те же преобразования, что и аналогичный запрос, читающий из подзапроса:

```
SELECT * FROM (SELECT * FROM underlying_table);
```

но с дополнительным расходом времени за загрузку содержимого `a_view` из базы данных и подстановку запроса на место представления.

Важно отметить, что результаты компиляции в YDB кэшируются и поэтому при исполнении одного и того же запроса несколько раз подряд в рамках одной сессии:

```
-- выполняется несколько раз в течение одной сессии
SELECT * FROM hot_view;
```

вы не заметите разницы в общем времени выполнения запроса, записанного с участием представлений, по сравнению с аналогичным запросом, записанным непосредственно.

Транзакционность переопределения представления

Важно

Планы выполнения запросов, содержащих представления, в настоящее время кэшируются. Это может привести к использованию старого плана запроса в течение короткого времени после того, как данное представление было изменено. Эта проблема будет исправлена в будущих версиях YDB. Ниже приведено более подробное описание проблемы.

Кэш результатов компиляции

YDB кэширует результаты компиляции запросов на стороне сервера для большей скорости их выполнения. Для лёгких запросов вида `SELECT 1;` компиляция может занимать значительное время по сравнению со временем выполнения запроса.

Запись в кэше ищется по ключу, основа которого - текст запроса (но в нём есть и другие поля, например, ID пользователя).

Кеш обновляется автоматически, чтобы оставаться актуальным, но в случае с представлениями это обновление не происходит в той же транзакции, в которой пользователь изменяет определение представления, а выполняется с некоторой задержкой.

Описание проблемы

Представьте себе следующую ситуацию.

Алёна постоянно выполняет следующий запрос:

```
-- сессия Алёны
SELECT * FROM some_view_which_is_going_to_be_redefined;
```

В то же время Борис переопределяет представление, из которого читает Алёна, следующими командами:

```
-- сессия Бориса
DROP VIEW some_view_which_is_going_to_be_redefined;
CREATE VIEW some_view_which_is_going_to_be_redefined ...;
```

Текст запроса Алёны не меняется, а значит его компиляция будет выполнена один раз, а все последующие запросы будут брать готовый результат компиляции из кэша. Борис переопределяет представление, поменяв запрос хранящийся в нём. Теоретически, запись с результатами компиляции запроса Алёны должна быть удалена из кэша тут же, в одной транзакции с изменениями Бориса. Но на практике это не так. Запрос Алёны будет перекомпилирован после того, как информация об изменившемся определении представления дойдёт до тех нод, на которых закэширован результат компиляции запроса Алёны. Это означает, что в течении некоторого времени, запросы Алёны могут возвращать результат, не соответствующий актуальному определению представления. Подобное поведение YDB нежелательно и будет исправлено в следующих релизах.

См. также

- CREATE VIEW
- ALTER VIEW
- DROP VIEW

Топик

Топик в YDB — это сущность для хранения неструктурированных сообщений, предназначенная для их доставки множеству подписчиков. Фактически, топик — это именованный набор сообщений.

Приложение-писатель записывает сообщения в топик. Приложения-читатели независимы друг от друга, они получают, "считывают", сообщения из топика в том порядке, в котором они были записаны. С помощью топика реализуется архитектурный шаблон издатель-подписчик.

Топик YDB обладает следующими свойствами:

- Гарантиями at-least-once при чтении сообщений подписчиками.
- Гарантиями exactly-once при публикации сообщений (дедупликация сообщений).
- Гарантиями последовательной обработки сообщений (FIFO), опубликованных с одинаковым идентификатором источника.
- Масштабирование пропускной способности передачи сообщений, публикуемых с разными идентификаторами последовательности.

Сообщения

Данные передаются в виде потоков сообщений. Сообщение — это минимальная неделимая единица пользовательской информации. Сообщения состоят из тела и атрибутов, а также из дополнительных системных свойств. Содержимое сообщений представляет собой набор байт, которое никак не интерпретируется YDB.

Сообщения могут содержать пользовательские атрибуты в формате "ключ-значение". Они возвращаются вместе с телом сообщения при чтении. Пользовательские атрибуты позволяют читателю принять решение о необходимости обработки сообщения, не распаковывая тело сообщения.Атрибуты сообщения задаются при инициализации сессии записи. Это означает, что все сообщения, записанные в рамках одной сессии записи, будут обладать одинаковыми атрибутами при чтении.

Партиционирование

Для горизонтального масштабирования топик разделяется на отдельные элементы, *партиции*, являющиеся единицами параллелизма. Каждая партиция имеет ограниченную пропускную способность, рекомендуемая скорость записи до 1 МБ/с.

> **Примечание**
>
> В настоящий момент уменьшение количества партиций в топике поддерживается только при помощи удаления и повторного создания топика с меньшим количеством партиций.

Партиции бывают двух видов:

- **Активные.** Все партиции по умолчанию, в них возможны как запись, так и чтение.
- **Неактивные.** Из них можно только читать. Неактивные партиции появляются после разделения партиции при включенном автопартиционировании. Неактивные партиции удаляются автоматически, когда все сообщения из такой партиции удалены по истечении времени хранения.

Смещение (offset)

Все сообщения внутри партиции имеют уникальный порядковый номер, называемый `смещением` (offset). Смещение монотонно возрастает при записи новых сообщений.

Автопартиционирование

Количество партиций топика и их пропускная способность задаются при создании топика и определяют общую пропускную способность топика на запись. Если максимальная требуемая скорость записи в топик неизвестна при его создании или будет меняться со временем, можно воспользоваться автопартиционированием для динамического масштабирования топика. Если на топике включено автопартиционирование вверх, то количество партиций в таком топике автоматически увеличивается при возрастании скорости записи (подробнее см. Режимы автопартиционирования).

Гарантии

1. SDK и сервер обеспечивают гарантии записи exactly-once в случае разделения партиций. Это означает, что любое сообщение будет записано либо в родительскую партицию, либо в одну из дочерних. Сообщение не может быть записано одновременно и в родительскую, и в дочернюю партиции. Более того, сообщение не может быть записано в одну партицию несколько раз.
2. SDK и сервер обеспечивают порядок чтения. Сначала данные будут вычитаны из родительских партиций, и только затем — из дочерних.
3. Таким образом, гарантии exactly-once записи и порядка чтения продолжают выполняться для конкретного идентификатора источника (producer-id).

Режимы автопартиционирования

Для любого топика возможны следующие режимы автопартиционирования.

Выключено (DISABLED)

Автопартиционирование отключено. В этом случае число партиций остаётся неизменным, и автоматическое масштабирование не происходит.

Исходное количество партиции указывается при создании топика. При ручном изменении числа партиций в этом режиме добавляются новые партиции. Все ранее существовавшие партиции остаются активными.

Увеличение (UP)

На топике включено автопартиционирование «вверх», то есть при увеличении скорости записи количество партиций увеличивается. При уменьшении скорости записи количество партиций остаётся неизменным.

Алгоритм увеличения числа партиций: если в течение заданного времени скорость записи в какую-то партицию превышает указанный порог (в % от максимальной скорости записи в партицию), эта партиция делится на две. Исходная партиция становится неактивной, и из неё можно только читать данные. Когда истечёт срок хранения сообщений в этой партиции и все сообщения будут удалены, сама партиция также будет удалена. Две новые дочерние партиции становятся активными, и в них возможны как чтение, так и запись.

Пауза (PAUSED)

Автопартиционирование на топике приостановлено. Автоматическое увеличение числа партиций не происходит. При необходимости можно снова включить режим увеличения числа партиций.

Примеры YQL-запросов для перевода топика в различные режимы автопартиционирования можно посмотреть здесь.

> **Примечание**
>
> Режимы автопартиционирования DOWN и UP_AND_DOWN находятся в разработке и в текущей версии не поддержаны.
> Режим UP_AND_DOWN на текущий момент реализован как UP.

Ограничения

При использовании автопартиционирования действуют следующие ограничения:

1. Если на топике включено автопартиционирование, его нельзя отключить, можно только приостановить.
2. Если на топике включено автопартиционирование, то запись или чтение в такой топик по протоколу Kafka API невозможны.
3. Автопартиционирование не может быть включено на топике с режимом хранения по месту.

Источники

Идентификатор источника, `producer_id` — это способ упорядочить набор сообщений. Порядок записанных сообщений сохраняется в пределах `producer_id`.

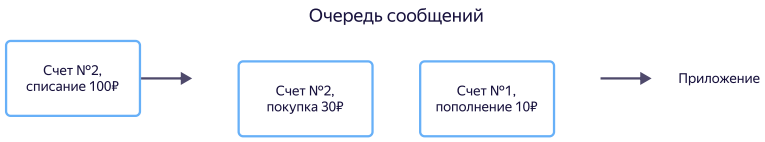
При первом использовании идентификатор источника `producer_id` привязывается к партиции топика по алгоритму round-robin и все сообщения с этим `producer_id` попадают в одну и ту же партицию. Привязка удаляется при отсутствии новых сообщений, использующих этот идентификатор источника, в течение 14 дней.

> **Важно**
>
> Рекомендуемое максимальное количество `producer_id` — до 100 тысяч на одну партицию за последние 14 дней. Для ориентира — это примерно 5 новых `producer_id` в минуту круглосуточно.

Когда важен порядок обработки сообщений

Рассмотрим финансовое приложение, задача которого вычислять остаток на счете пользователя и разрешать либо запрещать списание средств.

Для решения подобных задач можно использовать очередь сообщений. При пополнении счета, снятии средств или совершении покупки в очередь записывается сообщение с идентификатором счета, суммой и типом операции. Приложение обрабатывает поступающие сообщения и вычисляет баланс.

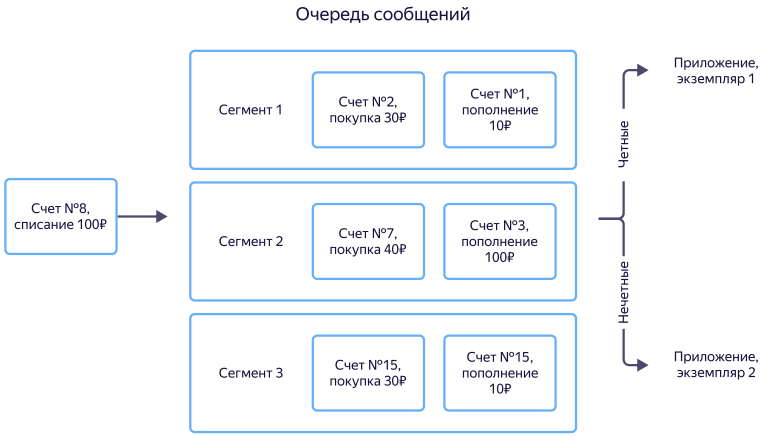


Для правильного вычисления баланса важен порядок обработки сообщений. Если пользователь сначала пополняет счет, а затем совершает покупку, то и сообщения с информацией об этих операциях должны быть обработаны приложением в такой же последовательности. Иначе может произойти ошибка бизнес-логики, и, например, приложение отклонит покупку из-за нехватки средств. В очередях сообщений есть механизмы гарантированного порядка доставки, но они не могут обеспечить порядок сообщений внутри одной очереди на произвольных объемах данных.

Когда сообщения из потока читают несколько экземпляров приложения, сообщение о пополнении счета может получить один, а о списании — другой. В этом случае не существует экземпляра, который гарантированно содержит верную информацию о балансе. Для решения этой проблемы можно сохранять данные в СУБД, обмениваться информацией между экземплярами приложения, строить распределенный кеш и пр.

В YDB можно так записывать данные, чтобы сообщения от одного источника приходили в один и тот же экземпляр приложения. Для этого сообщения от каждого источника пишутся со своими уникальными идентификаторами источника (`producer_id`), а для защиты от дублей используется порядковый номер сообщения от источника (`seqno`). В YDB сообщения с одинаковым `producer_id` попадают в одну и ту же партицию. При чтении из топика каждый из экземпляров читателя обслуживает своё подмножество партиций, благодаря чему задача синхронизации между экземплярами не возникает. Например, с помощью этого подхода можно сделать так, чтобы сообщения о транзакциях по одному счёту всегда попадали в одну и ту же партицию и обрабатывались экземпляром приложения, который связан с этой партицией.

Ниже приведен пример, когда все транзакции по счетам с четными идентификаторами передаются в первый экземпляр приложения, а с нечетными — во второй.



Если порядок обработки не важен

Для некоторых задач порядок обработки сообщений не критичен. Например, иногда важно просто доставить данные, а упорядочение выполнит система хранения.

Для таких случаев можно использовать упрощенный режим записи, называемый "записью без дедупликации". В этом режиме не нужно указывать идентификаторы источника сообщений (`producer_id`) и порядковые номера сообщений — `sequence number` . Запись без дедупликации работает быстрее и потребляет меньше ресурсов на сервере, но упорядочение и дедупликация сообщений на сервере не происходит. Это значит, что если отправить одно и то же сообщение повторно (например, при падении и последующем перезапуске пишущего процесса), оно может быть записано больше одного раза.

Порядковые номера сообщений

Все сообщения от одного источника имеют порядковый номер, `sequence number`, используемый для дедупликации. Порядковый номер сообщения должен монотонно возрастать в рамках пары `топик` , `источник` . При получении сервером сообщения с порядковым номером, меньшим или равным максимальному записанному по паре `топик` , `источник` , сообщение будет пропущено как дубликат. При этом допускается наличие пропусков в последовательности порядковых номеров сообщений. Порядковые номера сообщений должны быть уникальны только в пределах пары `топик` , `источник` .

Не используются, если выбран режим записи без дедупликации.

Примеры порядковых номеров сообщений

Тип	Пример	Описание
Файл	Смещение передаваемых данных от начала в файле	Нельзя удалять строки из начала файла, так как это приведет или пропуску части данных, как к дублям, либо к потере части данных.
Таблица базы данных	Автоинкрементный идентификатор записи	

Время хранения сообщений

Для каждого топика определено время хранения сообщений. После истечения времени хранения сообщения автоматически удаляются.
Исключение составляют данные, которые ещё не были подтверждены "важным" читателем — они будут храниться до тех пор, пока читатель их не обработает.
При наличии читателя с явно указанным временем доступности, время хранения необработанных читателем сообщений увеличивается до указанного значения.

Сжатие данных

При передаче приложение-писатель указывает, что сообщение может быть сжато одним из поддерживаемых кодеков. Название кодека передается при записи и сохраняется вместе с сообщением, а также возвращается на чтении. Сжатие сообщений происходит по каждому сообщению в отдельности, сжатие пакета сообщений не поддерживается. Операции сжатия-разжатия данных производятся на стороне приложений-читателей и -писателей.

Список поддерживаемых кодеков явно указывается в каждом топике. При попытке записи данных в топик с неподдерживаемым кодеком, это приведет к ошибке записи.

Кодек	Описание
`raw`	Без сжатия.
`gzip`	Сжатие алгоритмом gzip.
`lzop`	Сжатие алгоритмом lzop.
`zstd`	Сжатие алгоритмом zstd.

Читатель

Читатель — это именованная сущность для чтения данных из топика. Читатель содержит позиции чтения, подтвержденные читателем по каждому топику, читаемого от его имени.

Сессия чтения — это подключение клиента к топику для получения сообщений от имени читателя. Сессии чтения работают через Topic API. Один читатель может установить несколько сессий чтения: в этом случае партиции топика распределяются между этими сессиями. Для работы с сессиями чтения рекомендуется использовать YDB SDK (см. Работа с топиками).

В YDB читатели могут быть следующих типов:

- **потоковый** — это тип читателя, при котором партиция эксклюзивно назначается одному потребителю, и все сообщения из этой партиции обрабатываются этим потребителем.
- **разделяемый (общий)** — это тип читателя, который позволяет нескольким потребителям совместно обрабатывать сообщения из одного топика без эксклюзивного назначения партиций каждому из них.

Потоковый читатель

Потоковый читатель предназначен для потоковой обработки данных из топика. В этом режиме несколько экземпляров приложения-потребителя параллельно читают сообщения и обрабатывают их по мере поступления. Типичные сценарии — сбор и анализ логов приложения, построение агрегатов в реальном времени, репликация событий во внешние системы.

Читать из топика с использованием потокового читателя можно по Topic API и протоколу Kafka.

В потоковом режиме каждая партиция топика в каждый момент времени назначается не более чем одному потребителю. Все сообщения из назначенной партиции обрабатывает этот потребитель.

За распределение партиций между потребителями отвечает сервер YDB. При подключении и отключении потребителей сервер перераспределяет партиции и стремится распределить их между активными потребителями как можно равномернее.

Количество партиций определяет максимальную степень параллелизма: одновременно читать из топика могут не больше потребителей, чем партиций. Если потребителей больше, чем партиций, часть экземпляров остаётся без назначенных партиций и простаивает до появления свободных партиций или уменьшения числа потребителей.

Разделяемый (общий) читатель

Разделяемый (общий) читатель в YDB — это модель чтения из топика, когда за читателем закрепляется одно или несколько сообщений из топика, а не партиция топика целиком. Это позволяет одну партицию обрабатывать одновременно большим количеством потребителей.

Типичный пример использования — обмен сообщениями между микросервисами. Читать из топика с использованием разделяемого (общего) читателя можно по протоколу Amazon SQS.

Сообщение за потребителем закрепляется на некоторое время. Если за это время потребитель не сообщил об окончании обработки сообщения и не продлил его время обработки, то это значит, что потребитель не смог обработать сообщение, и сообщение становится доступным для чтения и будет вычитано еще раз. Время обработки настраивается в параметрах читателя, но может быть задано и для каждого запроса на чтение сообщений.

К сообщениям, которые не могут быть обработаны, применяется Dead letter policy. Dead letter policy может быть настроена одним из следующих способов:

- **none** — сообщение после неуспешной обработки возвращается в очередь и после будет отдано для повторного чтения.
- **move** — сообщение после нескольких попыток обработки перемещается в DLQ, в качестве которого может быть указан любой другой топик базы данных.
- **delete** — сообщение после нескольких попыток обработки удаляется.

По умолчанию Dead letter policy выключена (none). Количество попыток обработки сообщения до его перемещения в DLQ (move) или его удаления (delete) настраивается на читателе. Если не удалось переместить сообщение в DLQ, то оно возвращается обратно в очередь (как при выключенной Dead letter policy), а при следующей неудачной попытке обработать сообщение будет снова выполнена попытка переместить сообщение в DLQ. Например, если удалить топик, указанный в качестве DLQ, то сообщения после неудачной обработки будут возвращаться в очередь до тех пор, пока не будет создан топик, указанный в качестве DLQ, либо не будет указан другой существующий топик.

В зависимости от конфигурации разделяемого (общего) читателя чтение из топика может осуществляться как с сохранением порядка сообщений, так и без. Порядок сообщений сохраняется среди сообщений с одинаковым message-group-id, который может быть задан для каждого сообщения при его записи. Если разделяемый (общий) читатель сконфигурирован на чтение с сохранением порядка сообщений, то таким читателем в один момент времени потребителям на обработку выдаётся не более одного сообщения из одной message-group-id.

При помещении сообщения в очередь можно указать задержку перед тем, как сообщение будет отдано на обработку в первый раз. Порядок сообщений определяется временем добавления сообщения в топик — если чтение осуществляется с сохранением порядка сообщений, то отложенное сообщение приостановит выдачу на обработку всех последующих сообщений с таким же message-group-id.

См. также Разделяемый читатель топика: устройство и ограничения.

Позиция чтения

Позиция чтения — это сохраненное смещение читателя по каждой партиции топика. Позиция чтения сохраняется читателем после отправки подтверждения прочитанных данных. При установке новой сессии чтения сообщения поступают читателю начиная с сохраненной позиции чтения. Это позволяет пользователям не хранить позицию чтения на своей стороне.

Ограничение чтения из одной партиции

Параметр `partition_max_in_flight_bytes` можно задать при создании сессии чтения.

Этот параметр задает верхнюю границу in-flight - объема данных по одной партиции, которые уже выданы сервером в ответах чтения, но еще не подтверждены коммитом смещения.

- `0` — отдельный лимит in-flight по партиции не применяется;
- положительное значение — при достижении лимита сервер временно прекращает отдавать новые порции по этой партиции до тех пор, пока клиент не выполнит коммит и не уменьшит in-flight.

Слишком маленькое значение при медленной обработке или редких коммитах может вызывать паузы чтения по партиции. В таком случае увеличьте лимит или выполняйте коммиты чаще.

Важный читатель

Читатель может обладать признаком "важный". Наличие этого признака означает, что сообщения в топике не будут удаляться до тех пор, пока читатель не прочитает и не подтвердит сообщения. Этот признак можно устанавливать для самых критичных читателей, которые должны обработать все данные даже при длительном простое.

Важно

Так как длительный простой важного читателя может привести к использованию всего свободного места хранения данных непрочитанными сообщениями, необходимо следить за отставанием чтения важных читателей

Время доступности сообщений для читателя

Для читателя может быть задан временной период доступности, в течение которого ему будут доступны сообщения из топика, которые он ещё не обработал.

Эта опция позволяет продлить время хранения сообщений в топике с времени хранения вплоть до указанного времени доступности, если читатель не подтверждает обработку.
В отличие от признака "важного" читателя, параметр ограничивает максимальный возраст сообщений, которые будут храниться в топике.
Если опция не задана или все читатели подтверждают обработку без большой задержки (меньше времени хранения топика), то данные будут удалены по обычным правилам.

Протоколы для работы с топиками

Для работы с топиками используется YDB SDK (см. Работа с топиками).

Также ограниченно поддержаны протоколы Kafka API версии 3.4.0 (см. Kafka API) и Amazon SQS API (см. Amazon SQS API).

С одним топиком работа может вестись одновременно по нескольким протоколам. Например, запись может осуществляться с использованием Topic API, а чтение — с использованием Amazon SQS API, и наоборот.

Транзакции с участием топиков

YDB поддерживает работу с топиками в рамках транзакций.

Транзакционное чтение из топика

Данные в топиках не изменяются при чтении из топика. Поэтому при чтении в транзакции из топика непосредственно транзакционной операцией является только изменение смещения (offset). При транзакционном чтении через SDK не происходит коммита смещений. Отложенный коммит смещений происходит автоматически при коммите транзакции, SDK скрывает это от пользователя.

Транзакционная запись в топик

При транзакционной записи в топик данные до коммита сохраняются вне партиции, а затем публикуются (становятся видимыми) в момент коммита транзакции. При этом данные будут добавлены в конец партиции в последовательные смещения (offset). Видимости собственных изменений в топиках в транзакциях с участием топиков не предусмотрено.

Ограничения при работе с топиками в транзакции

Транзакции не накладывают дополнительных ограничений на работу с топиками. Внутри транзакции можно записывать большие объёмы данных в топик, писать в несколько партиций и читать несколькими консьюмерами.

Тем не менее рекомендуется выбирать режим работы с транзакциями, учитывая особенности транзакционной работы с топиками: данные публикуются в момент коммита транзакции. То есть если транзакция длительная, данные станут видимыми только спустя значительное время.

YQL-запросы к топикам

Для работы с топиками можно использовать YQL-запросы с привычными конструкциями: SELECT для чтения сообщений из топика и INSERT для записи сообщений в топик.

Узел координации

Узел координации (coordination node) — это объект в YDB, который позволяет клиентским приложениям распределённо координировать свои действия. Типовые сценарии использования узлов координации:

- Распределённые семафоры и мьютексы.
- Обнаружение сервисов (service discovery).
- Выбор лидера (leader election).
- Очереди задач (task queue).
- Публикация небольших объёмов данных с возможностью получать оповещения об изменениях.
- Эфеменрные блокировки произвольных заранее неизвестных сущностей.

Семафоры

Узлы координации позволяют создавать внутри себя семафоры и управлять ими. Типовые операции с семафорами:

- Создание (create);
- Захват (acquire);
- Освобождение (release);
- Получение содержимого (describe);
- Удаление (delete).

К семафору может быть приложен счётчик, который ограничивает количество одновременных его захватов, а также небольшой объём произвольных данных.

YDB поддерживает два типа семафоров: персистентные и эфемерные. Персистентный семафор перед захватом необходимо создать, он будет существовать либо до момента его явного удаления, либо до удаления узла координации, в котором он был создан. Эфемерные семафоры автоматически создаются в момент их первого захвата и удаляются при последнем освобождении, что удобно использовать, например, в сценариях распределённой блокировки.

> **Примечание**
>
> Семафоры в YDB **не** являются рекурсивными. Таким образом, захват и освобождение семафора являются идемпотентными операциями.

Использование

Работа с узлами координации и семафорами осуществляется через отдельные методы в YDB SDK.

Аналогичные системы

С помощью узлов координации YDB можно решать задачи, которые традиционно выполняются с помощью таких систем, как Apache Zookeeper, etcd, Consul и других. Если в проекте используется YDB для хранения данных, а также одна из этих сторонних систем для координации, то переход на узлы координации YDB может сократить количество систем, которые необходимо эксплуатировать и поддерживать.

Секреты

Когда YDB обращается ко внешним системам — например, для получения данных из объектных хранилищ или внешних СУБД через федеративные запросы или для асинхронной репликации или трансфера данных в другую базу YDB, — возникает задача аутентификации во внешних системах. Для её решения аутентификационные данные, требующие особого обращения, такие как пароли и токены, можно хранить в специальных схемных объектах — секретах.

Секреты позволяют YDB предоставлять необходимые аутентификационные данные для взаимодействия с внешними системами, при этом значение секрета нельзя просмотреть, а можно только создать или обновить. Использование секретов обеспечивает дополнительную безопасность и изоляцию доступа при работе с аутентификационными данными.

Синтаксис

Для управления секретами используются следующие операторы YQL:

- CREATE SECRET — создание секрета.
- ALTER SECRET — изменение существующего секрета.
- DROP SECRET — удаление секрета.

Использование

Примеры использования секретов и работа с ними есть в следующих разделах:

- Настройка времени жизни строк (TTL) таблицы
- Чтение из бакетов S3 через внешние источники данных

Управление доступом

Секреты являются объектами схемы, поэтому права на них выдаются с помощью команды `GRANT`, а отзываются – с помощью команды `REVOKE`. Для использования секрета в запросе, например, при создании внешнего источника данных или трансфера данных, необходимо право `SELECT ROW`.

Внешние источники данных

Внешний источник (external data source) - это объект в YDB, описывающий параметры подключения к внешнему источнику данных. Например, в случае ClickHouse внешний источник описывает сетевой адрес, логин и пароль для аутентификации в кластере ClickHouse, а в случае S3 (Object Storage) описывает реквизиты доступа и путь к бакету.

В следующем примере приведен пример создания внешнего источника, ведущего на кластер ClickHouse:

```
CREATE EXTERNAL DATA SOURCE test_data_source WITH (
  SOURCE_TYPE="ClickHouse",
  LOCATION="192.168.1.1:8123",
  DATABASE_NAME="default",
  AUTH_METHOD="BASIC",
  USE_TLS="TRUE",
  LOGIN="login",
  PASSWORD_SECRET_PATH="test_password_path",
  PROTOCOL="NATIVE"
);
```

После создания внешнего источника данных можно выполнять чтение данных из созданного объекта `EXTERNAL DATA SOURCE`. Пример ниже иллюстрирует чтение данных из таблицы `test_table` из базы данных `default` в кластере ClickHouse:

```
SELECT * FROM test_data_source.test_table;
```

С помощью внешних источников данных можно выполнять федеративные запросы для задач межсистемной аналитики данных.

В качестве источников данных можно использовать:

- S3

Экспериментальные источники данных

- ClickHouse
- Greenplum
- Microsoft SQL Server
- MySQL
- PostgreSQL
- YDB (федеративные запросы между базами YDB)

Внешние таблицы

Часть внешних источников, например, системы управления базами данных, хранят данные в схематизированном виде, а часть, как S3 (Yandex Object Storage), в виде отдельных файлов. Для работы с файловыми источниками данных необходимо знать как правила расположения файлов, так и форматы самих хранимых данных.

Для описания хранимых данных в таких источниках используется специальная сущность - внешние таблицы, `EXTERNAL TABLE`. Внешние таблицы позволяют задать схему данных у хранимых файлов, а также описать схему расположения файлов внутри источника.

Запись в YQL может выглядеть так:

```
CREATE EXTERNAL TABLE s3_test_data (
  key Utf8 NOT NULL,
  value Utf8 NOT NULL
) WITH (
  DATA_SOURCE="bucket",
  LOCATION="folder",
  FORMAT="csv_with_names",
  COMPRESSION="gzip"
);
```

Во внешние таблицы можно вставлять данные, так же, как и в обычные. Например, для записи данных во внешнюю таблицу достаточно выполнить следующий запрос:

```
INSERT INTO s3_test_data
SELECT * FROM Table
```

Подробнее про работу с внешними таблицами, описывающими бакеты S3 (Object Storage) описано в разделе Чтение из бакетов S3 через внешние таблицы.

Коллекция резервных копий

Коллекция резервных копий организует полные и инкрементальные резервные копии выбранных строковых таблиц в управляемые цепочки. Она позволяет **восстановить данные к состоянию самой поздней резервной копии в цепочке**, обеспечивая защиту от случайной потери данных — ошибочных удалений или изменений.

> **Примечание**
>
> Практические инструкции по созданию и управлению коллекциями резервных копий см. в разделе Коллекции резервных копий документа «Резервное копирование и восстановление».

Обзор

Коллекции резервных копий решают типичные задачи резервного копирования для производственных нагрузок:

- **Эффективность хранения**: Инкрементальные резервные копии захватывают только изменения с момента предыдущей резервной копии, значительно сокращая требования к хранилищу по сравнению с множеством полных резервных копий.
- **Согласованное восстановление**: Все таблицы в коллекции создаются из одного глобального снимка, обеспечивая ссылочную целостность между таблицами при восстановлении.
- **Восстановление по цепочке**: Восстановление к состоянию **самой поздней** резервной копии в актуальной цепочке — см. `RESTORE`.

Для сравнения с другими методами резервного копирования (export/import, dump/restore) см. Концепции резервного копирования.

Ключевые концепции

Эти термины необходимы для понимания коллекций резервных копий. Подробные определения см. в глоссарии.

- **Полная резервная копия**: Полный снимок всех данных в коллекции на определённый момент времени. Служит основой для последующих инкрементальных резервных копий.
- **Инкрементальная резервная копия**: Захватывает только изменения (вставки, обновления, удаления) с момента предыдущей резервной копии. Требует всей цепочки резервных копий для восстановления.
- **Цепочка резервных копий**: Упорядоченная последовательность, начинающаяся с полной резервной копии, за которой следует ноль или более инкрементальных резервных копий.

Ограничения

Перед использованием коллекций резервных копий учтите эти ограничения:

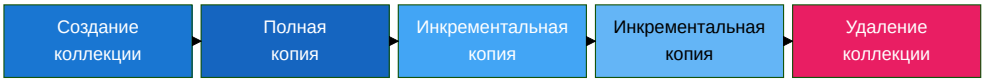
- **Только строковые таблицы**: Колоночные таблицы не поддерживаются.
- **Одна коллекция на таблицу**: Таблица может принадлежать только одной коллекции резервных копий. Чтобы включить таблицу в другую коллекцию, выполните `DROP BACKUP COLLECTION` для текущей коллекции и создайте новую с нужным составом таблиц.
- **Неизменяемый состав**: После создания список таблиц в коллекции нельзя изменить. Для добавления новых таблиц создайте новую коллекцию со всеми нужными таблицами.
- **Нет частичного восстановления**: Нельзя восстановить отдельные таблицы из коллекции; восстанавливается вся коллекция целиком.
- **Внешнее планирование**: YDB не предоставляет встроенного планирования резервного копирования. Используйте внешние инструменты, такие как cron, для автоматизации.

Архитектура

Коллекции резервных копий используют механизм копирования при записи в сочетании с потоками изменений для эффективных инкрементальных резервных копий. В этом разделе объясняется, как компоненты работают вместе.

Как работают коллекции резервных копий

Следующая диаграмма иллюстрирует процесс резервного копирования:



Создание коллекции определяет, какие таблицы включить, и создаёт схемный объект. Это быстрая операция только с метаданными.

Полная резервная копия создаёт согласованный снимок всех таблиц в коллекции. Ключевые характеристики:

- Использует **глобальный снимок**, который обеспечивает согласованность всех таблиц в коллекции.
- Создаёт **потоки изменений** для каждой таблицы для отслеживания последующих модификаций.
- Использует **копирование при записи**: Резервная копия создаётся быстро путём ссылки на существующие данные; фактическое копирование данных происходит только при модификации исходных данных.

Инкрементальная резервная копия захватывает все изменения с момента предыдущей резервной копии:

- Использует **распределённую транзакцию** для чтения потоков изменений из всех таблиц в согласованной точке, обеспечивая ссылочную целостность всей коллекции.
- Считывает накопленные изменения из потоков изменений, созданных при выполнении полной резервной копии и накапливающих изменения **с момента предыдущей резервной копии (полной или инкрементальной)**.
- Записывает все модификации: вставки, обновления и удаления (в том числе в виде записей об удалении).
- Сохраняет данные потоков изменений в таблицы инкрементальной резервной копии.

> **Важно**
>
> Изменения схемы (ALTER TABLE) таблиц в коллекции резервных копий не отслеживаются инкрементальными резервными копиями. Если вам нужно изменить схему таблицы с резервной копией, создайте новую полную резервную копию после изменения схемы, чтобы цепочка резервных копий отражала новую структуру.

Примечание

Потоки изменений, созданные при полной резервной копии, автоматически удаляются при удалении коллекции резервных копий. Их нельзя удалить вручную или использовать для других целей, пока коллекция существует.

Хранение

Коллекции резервных копий хранятся в кластере YDB в выделенной структуре каталогов:

```
/Root/database/.backups/collections/
├── my_collection/
|   ├── 20250821141425Z_full/
|   |   ├── table_1/
|   |   └── table_2/
|   └── 20250821151519Z_incremental/
|       ├── table_1/
|       └── table_2/
```

Примечание

Каталог `.backups` создаётся автоматически при создании первой коллекции резервных копий и управляется системой. После его создания вы можете управлять таблицами резервных копий внутри него (например, при экспорте или импорте резервных копий).

Кластерное хранилище

По умолчанию резервные копии хранятся в кластере. Резервные копии в кластере предназначены для восстановления после **логических ошибок**, таких как случайный `DROP TABLE`, `TRUNCATE TABLE` или ошибочные изменения данных. Преимущества:

- Быстрые операции резервного копирования и восстановления.
- Интегрированные механизмы безопасности.
- Не требуется внешняя инфраструктура.

Важно

Резервные копии в кластере находятся в той же зоне отказа, что и защищаемые данные. Если кластер выйдет из строя с превышением его отказоустойчивости (например, полная потеря кластера или катастрофические события в дата-центре), и данные, и резервные копии могут быть потеряны. Для защиты от таких сценариев используйте внешнее хранилище.

Внешнее хранилище

Для **аварийного восстановления** и защиты от отказов всего кластера регулярно экспортируйте коллекции резервных копий во внешнее хранилище (S3-совместимое хранилище или файловую систему) с помощью операций export/import.

Для экспорта резервных копий во внешнее хранилище используйте YDB CLI:

- `ydb export s3` для S3-совместимого хранилища.
- `ydb export nfs` для NFS на хостах кластера.
- `ydb tools dump` для файловой системы на локальном компьютере.

Каждую резервную копию в цепочке необходимо экспортировать отдельно. Сохраняйте порядок цепочки при экспорте/импорте для успешного восстановления.

Фоновые операции

Все операции резервного копирования и восстановления выполняются асинхронно, позволяя продолжать обычные операции с базой данных. Отслеживайте прогресс с помощью `ydb operation list incbackup`.

Восстановление из резервных копий

Восстановление возвращает данные к состоянию самой поздней резервной копии в цепочке, находящейся в кластере. Чтобы восстановиться к более раннему состоянию, импортируйте из внешнего хранилища только нужный префикс цепочки — см. Импорт и восстановление.

Процесс восстановления

1. **Импорт из внешнего хранилища** (при необходимости): Если резервные копии были экспортированы, импортируйте полную резервную копию и все инкрементальные резервные копии до нужной точки восстановления.
2. **Выполнение восстановления**: Выполните `RESTORE collection_name` для восстановления всех таблиц из коллекции резервных копий. Система применяет полную резервную копию и все инкрементальные резервные копии последовательно для достижения последней точки резервного копирования.

Важно

Операция восстановления завершится с ошибкой, если хотя бы одна из восстанавливаемых таблиц уже существует по тому же пути. Переименуйте или удалите конфликтующие таблицы перед восстановлением.

Все таблицы в коллекции восстанавливаются к одной и той же точке во времени: данные из разных таблиц будут взаимно согласованы.

Примечание

Во время операции восстановления целевые таблицы недоступны для модификаций. Частично восстановленные данные могут быть видимы для читающих нагрузок. Планируйте восстановление на окно обслуживания или отключайте доступ приложений к затронутым таблицам до завершения операции.

См. также

- Концепции резервного копирования: Обзор всех подходов к резервному копированию в YDB
- Резервное копирование и восстановление: коллекции резервных копий: пошаговые сценарии и команды CLI
- Рецепты и примеры: Типовые сценарии и примеры
- Справочник YQL:
  - CREATE BACKUP COLLECTION
  - BACKUP
  - RESTORE
  - DROP BACKUP COLLECTION

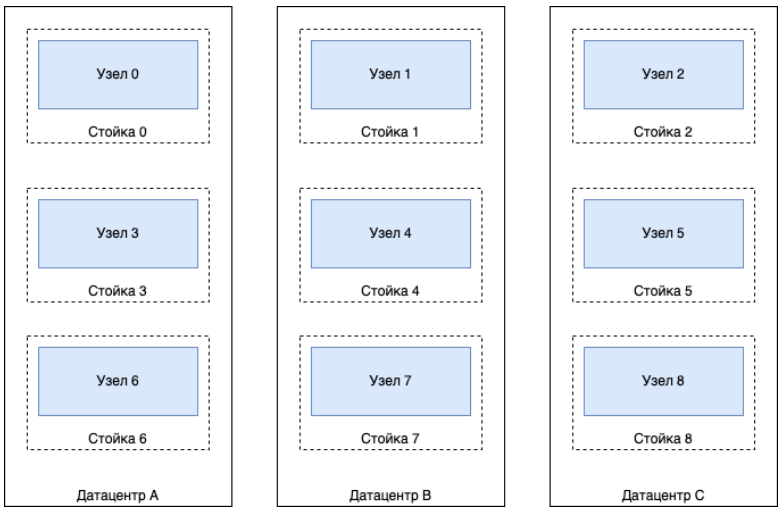
Топология кластера YDB

Кластер YDB состоит из узлов хранения и узлов баз данных. Работоспособность обоих типов узлов важна для обеспечения доступности баз данных YDB: узлы баз данных реализуют логику управления данными, а узлы хранения обеспечивают их сохранность. При этом наибольшее влияние на отказоустойчивость кластера и его способность обеспечивать надёжное хранение данных оказывает подсистема распределённого хранилища, состоящая из набора узлов хранения. При развёртывании кластера необходимо выбрать режим работы распределённого хранилища в соответствии с ожидаемой нагрузкой и требованиями к доступности баз данных. Режим работы нельзя изменить после первоначальной настройки кластера, что делает его выбор одним из ключевых решений при планировании нового развёртывания YDB.

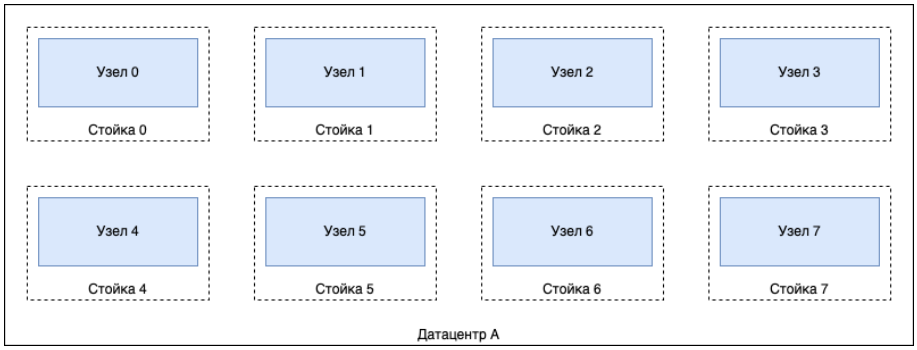
Режимы работы кластера

Топология кластера строится в соответствии с режимом работы распределённого хранилища, который должен быть выбран на основе требований к уровню отказоустойчивости. Модель отказа, используемая в YDB, основана на концепциях домена отказа и области отказа. YDB предоставляет следующие режимы работы распределённого хранилища:

- `mirror-3-dc`. Данные реплицируются между 3 областями отказа (обычно зонами доступности или дата-центрами) с использованием как минимум 3 доменов отказа (обычно серверных стоек) в каждой области отказа. Кластер YDB остаётся доступным при выходе из строя любой области отказа; кроме того, дополнительно может выйти из строя ещё один домен отказа в любой из 2 работоспособных областей отказа без прекращения работы кластера. Этот режим рекомендуется для кластеров с высокими требованиями к отказоустойчивости, размещённых в трёх и более дата-центрах.



- `block-4-2`. Избыточность по схеме стирающего кода. На каждые 4 блока исходных данных формируются 2 дополнительных блока с кодами избыточности. Узлы хранения размещаются как минимум в 8 доменах отказа (обычно серверных стойках). Кластер YDB остаётся полностью доступным при недоступности двух любых доменов отказа, продолжая записывать все 6 частей данных в оставшихся доменах. Этот режим рекомендуется для кластеров, размещённых в одном дата-центре или зоне доступности.



- `none`. Избыточность отсутствует. Любой отказ приводит к временной недоступности или потере всех или части хранимых данных. Этот режим рекомендуется использовать только для разработки приложений или проведения функциональных тестов.

> **Примечание**
>
> Под выходом из строя сервера подразумевается как полная, так и частичная его недоступность, например выход из строя одного диска на сервере.

Отказоустойчивые режимы работы распределённого хранилища требуют значительного количества оборудования для обеспечения максимальных гарантий доступности, поддерживаемых YDB. Однако в некоторых сценариях использования подобные затраты на оборудование могут быть слишком высокими на начальном этапе. В связи с этим YDB предлагает вариации данных режимов, которые требуют меньше оборудования, но при этом обеспечивают определённый уровень отказоустойчивости. Требования и гарантии всех режимов работы и их вариаций представлены в таблице ниже, а последствия выбора конкретного режима обсуждаются далее в статье.

Режим	Множитель объёма хранения	Минимальное количество узлов	Домен отказа	Область отказа	Кол-во дата-центров	Кол-во серверных стоек
`mirror-3-dc`, переживает отказ дата-центра и ещё 1 стойки в оставшихся дата-центрах	3	9 (12 рекомендовано[1])	Стойка	Дата-центр	3	3 в каждом дата-центре

`mirror-3-dc` *(упрощённый)*, переживает отказ дата-центра и ещё 1 сервера в оставшихся дата-центрах	3	12	½ стойки	Дата-центр	3	6
`mirror-3-dc` *(3 узла)*, переживает отказ 1 узла или 1 дата-центра	3	3	Сервер	Дата-центр	3	Не важно
`block-4-2`, переживает отказ 2 стоек	1.5	8 (10 рекомендовано[1])	Стойка	Дата-центр	1	8
`block-4-2` *(упрощённый)*, переживает отказ 1 стойки	1.5	10	½ стойки	Дата-центр	1	5
`block-4-2` *(упрощённый отказоустойчивый)*, переживает отказ 1 узла	1.5	4	Сервер	Дата-центр	1	Не важно
`none`, избыточность отсутствует	1	1	Узел	Узел	1	1

Примечание

Приведённый выше множитель объёма хранения относится только к фактору обеспечения отказоустойчивости. Для планирования размера хранилища необходимо учитывать и другие влияющие на него факторы, такие как фрагментация и гранулярность слотов.

О том, как задать топологию кластера YDB, читайте в разделе Конфигурация Blob Storage.

Режим bridge

Режим bridge — это особый режим работы кластера, существенно отличающийся от перечисленных ранее режимов распределённого хранилища. В режиме bridge узлы кластера разделяются на несколько pile (обычно соответствующих дата-центрам), в каждом из которых данные хранятся с использованием одного из описанных выше режимов распределённого хранилища, а между pile организуется синхронная репликация.

Важно понимать, что pile не являются независимыми кластерами YDB, а представляют собой части единого кластера со сложной топологией.

В режиме bridge предусмотрено явное управление остановкой и возобновлением репликации. Кластер YDB становится недоступным при отказе любого из pile до тех пор, пока не будет выполнена команда остановки репликации в этот pile. После её выполнения кластер восстанавливает работоспособность. Таким образом, кластер остаётся доступным до тех пор, пока не откажет последний pile.

Возобновление репликации в pile после её отключения может занять значительное время, поскольку YDB выполняет синхронизацию хранилища в этом pile с остальными, реплицируя недостающие данные. В процессе синхронизации кластер остаётся доступным.

Режим bridge рекомендуется использовать для кластеров, размещённых в двух дата-центрах, а также для систем с высокими требованиями к отказоустойчивости — например, когда необходимо сохранять доступность при отказе трёх дата-центров из четырёх.

При использовании режима bridge в каждом pile должно быть достаточно узлов, доменов и областей отказа для корректной работы выбранного режима хранения. При этом результирующий множитель объёма хранения будет равен сумме множителей объёмов хранения всех pile.

Время отклика кластера в режиме bridge для большинства операций ограничено временем отклика самого медленного pile.

Упрощённые конфигурации

В случаях, когда невозможно использовать рекомендованное количество оборудования, можно разделить серверы одной стойки на 2 фиктивных домена отказа. В такой конфигурации отказ одной стойки будет означать отказ не одного, а сразу двух доменов. При использовании таких упрощённых конфигураций YDB сохраняет работоспособность при отказе сразу двух доменов. Минимальное количество стоек в кластере для режима `block-4-2` составляет 5, а для `mirror-3-dc` — по 2 в каждом дата-центре (т.е. суммарно 6 стоек).

Есть 2 варианта минимальной отказоустойчивой конфигурации кластера YDB:

- Вариант режима работы `mirror-3-dc` с 3 узлами, который требует всего три сервера с тремя дисками на каждом. В этой конфигурации каждый сервер выступает как домен отказа, так и как область отказа. Таким образом, кластер может выдержать отказ только одного сервера. Каждый сервер должен находиться в своём независимом дата-центре для обеспечения должного уровня отказоустойчивости.
- Вариант режима работы `block-4-2` с 4 узлами, который требует 4 сервера с 2 или более дисками на каждом. В этой конфигурации диски каждого сервера делятся на 2 домена отказа с помощью атрибута `disk_scope`, что в сумме даёт 8 доменов отказа, необходимых для работы режима `block-4-2`. Такой кластер сохраняет работоспособность при отказе одного сервера.

Кластеры YDB, настроенные с использованием одного из этих подходов, могут использоваться в production окружениях, если в них не требуются повышенные гарантии отказоустойчивости.

Доступный объём ресурсов и производительность

Система может работать с доменами отказа любого размера, однако при малом количестве доменов и неодинаковом количестве дисков в разных доменах количество групп хранения, которые можно будет создать, окажется ограничено. В таких условиях часть оборудования в слишком больших доменах отказа может оказаться недоиспользованной. В случае полной утилизации оборудования значительный перекос в размерах доменов может сделать невозможной реконфигурацию.

Например, в кластере с режимом отказоустойчивости `block-4-2` имеется 15 стоек. В первой из них расположено 20 серверов, а в остальных 14 стойках — по 10 серверов. Для полноценной утилизации всех 20 серверов из первой стойки YDB будет создавать группы так, что в каждой из них будет участвовать 1 диск из этого самого большого домена отказа. В

результате, при выходе из строя оборудования в любом другом домене отказа нагрузка не сможет быть распределена на оборудование в первой стойке.

YDB может объединять в группу хранения диски разных производителей, с различной ёмкостью и скоростью. Результирующие характеристики всей группы будут ограничены наихудшими характеристиками оборудования, которое в неё входит. Обычно лучших результатов удаётся добиться при использовании однотипного оборудования.

> **Примечание**
>
> При создании больших кластеров следует учитывать, что оборудование из одной партии с большей вероятностью может иметь одинаковый дефект и отказать одновременно.

Таким образом, в качестве оптимальных аппаратных конфигураций кластеров YDB для промышленного применения рекомендуются следующие:

- **Кластер в одной зоне доступности**: использует режим отказоустойчивости `block-4-2` и состоит из 9 или более стоек с равным количеством одинаковых серверов в каждой.
- **Кластер в трёх зонах доступности**: использует режим отказоустойчивости `mirror-3-dc` и расположен в трёх дата-центрах с четырьмя или более стойками в каждом. Стойки укомплектованы равным количеством одинаковых серверов.

Обеспечение доступности баз данных

База данных в кластере YDB доступна, если её хранилище и вычислительные ресурсы находятся в рабочем состоянии:

- Все выделенные базе данных группы хранения должны быть доступны, что означает соблюдение допустимого уровня отказов для каждой группы.
- Вычислительные ресурсы доступных в данный момент узлов базы данных (в первую очередь объём оперативной памяти) должны быть достаточны для запуска всех таблеток, управляющих пользовательскими объектами, такими как таблицы или топики, и для обработки пользовательских сессий.

Чтобы база данных могла пережить отказ одного из дата-центров в кластере, использующем режим работы `mirror-3-dc`, должны быть выполнены следующие условия:

- Узлы хранения должны обеспечивать как минимум двойной запас пропускной способности ввода-вывода и дисковой ёмкости по сравнению с необходимыми для работы в обычных условиях. В худшем случае нагрузка на сохранившиеся узлы при длительном отключении одного из дата-центров может утроиться, но лишь на ограниченный период времени — до завершения восстановления ставших недоступными дисков в оставшихся дата-центрах.
- Узлы базы данных должны быть равномерно распределены между всеми тремя дата-центрами и содержать достаточно ресурсов для обработки полной нагрузки при сохранении двух из трёх дата-центров. Это означает наличие как минимум 35% запаса по ресурсам CPU и оперативной памяти в обычных условиях, то есть при отсутствии каких-либо отказов. Если узлы базы данных обычно нагружены более чем на 65%, следует рассмотреть возможность добавления дополнительных узлов либо увеличения вычислительной мощности каждого узла.

Дополнительная информация

- Документация по администрированию кластеров
- Конфигурация Blob Storage
- Примеры конфигурационных файлов кластера
- Распределённое хранилище YDB

[1] Использование меньшего количества узлов приведёт к ограничению способности кластера к автоматическому восстановлению.

Выполнение запросов

YDB предоставляет единый интерфейс для выполнения запросов на языке YQL и единый распределённый движок их выполнения. Один и тот же синтаксис и одни и те же механизмы позволяют обращаться к данным разной природы: от строк в таблицах базы данных до неограниченных потоков событий в топиках и данных во внешних системах.

С точки зрения пользователя запросы выполняются к трём типам сущностей:

- таблицы базы данных — транзакционные (OLTP) и аналитические (OLAP) запросы к данным, хранящимся в YDB;
- внешние источники данных — федеративные запросы к данным, расположенным за пределами YDB;
- топики — запросы к неограниченным потокам данных: потоковые запросы и бесконечные запросы к топикам.

Общий порядок обработки запроса, а также базовые понятия — сессии, транзакции, повторные выполнения, язык запросов и результирующие наборы — описаны в разделе Процесс выполнения запросов.

Запросы к таблицам базы данных

Основной сценарий — выполнение запросов к таблицам, хранящимся в YDB. Единый интерфейс способен эффективно обрабатывать широкий спектр нагрузок — от высоконагруженных транзакционных OLTP-запросов до сложных аналитических OLAP-запросов.

Запросы к внешним источникам данных (федеративные запросы)

Федеративные запросы позволяют обращаться к данным, расположенным во внешних системах, без переноса этих данных внутрь YDB. С помощью YQL-запросов можно читать данные из внешних СУБД и объектных хранилищ (S3), а также объединять их с данными в таблицах YDB.

Подробнее см. в разделе Федеративные запросы.

Запросы к топикам

Запросы к топикам позволяют обрабатывать данные из очередей сообщений. Есть два варианта таких запросов.

Потоковые запросы

Потоковые запросы — основной способ потоковой обработки данных в production. Они оформляются как постоянные объекты схемы (`STREAMING QUERY`), читают сообщения из топиков по мере их поступления, записывают результаты в выходные топики или таблицы и автоматически восстанавливаются после сбоев из чекпоинтов.

Подробнее см. в разделе Потоковые запросы и в описании CREATE STREAMING QUERY.

Запросы к топикам в табличном режиме

Прочитать данные из топика можно и обычным `SELECT`. В отличие от потоковых запросов, он не создаёт постоянного объекта схемы и не восстанавливается после сбоев, поэтому предназначен в первую очередь для отладки и проверки данных в топике.

Подробнее см. в разделе YQL-запросы к топикам.

Выполнение запросов

YDB предоставляет единый интерфейс для выполнения запросов на языке YQL и единый распределённый движок их выполнения. Один и тот же синтаксис и одни и те же механизмы позволяют обращаться к данным разной природы: от строк в таблицах базы данных до неограниченных потоков событий в топиках и данных во внешних системах.

С точки зрения пользователя запросы выполняются к трём типам сущностей:

- таблицы базы данных — транзакционные (OLTP) и аналитические (OLAP) запросы к данным, хранящимся в YDB;
- внешние источники данных — федеративные запросы к данным, расположенным за пределами YDB;
- топики — запросы к неограниченным потокам данных: потоковые запросы и бесконечные запросы к топикам.

Общий порядок обработки запроса, а также базовые понятия — сессии, транзакции, повторные выполнения, язык запросов и результирующие наборы — описаны в разделе Процесс выполнения запросов.

Запросы к таблицам базы данных

Основной сценарий — выполнение запросов к таблицам, хранящимся в YDB. Единый интерфейс способен эффективно обрабатывать широкий спектр нагрузок — от высоконагруженных транзакционных OLTP-запросов до сложных аналитических OLAP-запросов.

Запросы к внешним источникам данных (федеративные запросы)

Федеративные запросы позволяют обращаться к данным, расположенным во внешних системах, без переноса этих данных внутрь YDB. С помощью YQL-запросов можно читать данные из внешних СУБД и объектных хранилищ (S3), а также объединять их с данными в таблицах YDB.

Подробнее см. в разделе Федеративные запросы.

Запросы к топикам

Запросы к топикам позволяют обрабатывать данные из очередей сообщений. Есть два варианта таких запросов.

Потоковые запросы

Потоковые запросы — основной способ потоковой обработки данных в production. Они оформляются как постоянные объекты схемы (`STREAMING QUERY`), читают сообщения из топиков по мере их поступления, записывают результаты в выходные топики или таблицы и автоматически восстанавливаются после сбоев из чекпоинтов.

Подробнее см. в разделе Потоковые запросы и в описании CREATE STREAMING QUERY.

Запросы к топикам в табличном режиме

Прочитать данные из топика можно и обычным `SELECT` . В отличие от потоковых запросов, он не создаёт постоянного объекта схемы и не восстанавливается после сбоев, поэтому предназначен в первую очередь для отладки и проверки данных в топике.

Подробнее см. в разделе YQL-запросы к топикам.

Процесс выполнения запросов

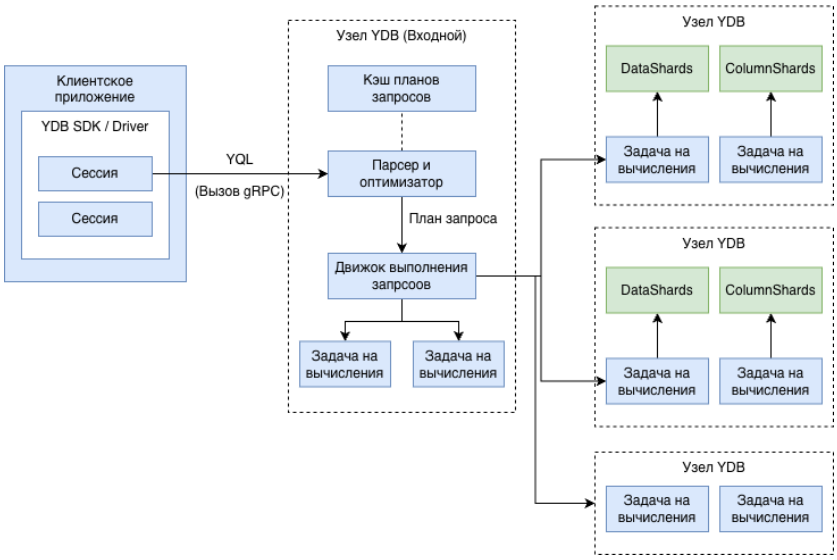
Эта статья описывает процесс выполнения запросов в YDB. Она предназначена для ознакомления пользователей с возможностями и ограничениями движка выполнения запросов YDB, включая ключевые особенности, такие как поддерживаемый язык запросов и общий порядок их выполнения. Также здесь вводится базовая терминология и концепции, которые используются в других разделах документации.

YDB предоставляет единый интерфейс для выполнения запросов, способный эффективно обрабатывать широкий спектр нагрузок — от высоконагруженных транзакционных OLTP-запросов до сложных аналитических OLAP-запросов. Такой подход позволяет приложениям выполнять транзакционные и аналитические запросы прозрачно, без необходимости использовать разные API для разных типов нагрузки.

Для выполнения запросов используется распределённый движок, спроектированный с учётом масштабируемости и эффективности в больших распределённых средах. При запуске запроса YDB автоматически распределяет работу между несколькими узлами, максимально учитывая локальность данных — обрабатывает данные там, где они хранятся. Это снижает избыточные сетевые пересылки. Кроме того, применяется вычислительный pushdown (вынос фильтрации и вычислений ближе к слою хранения), что дополнительно ускоряет обработку. Благодаря этим техникам YDB эффективно справляется со сложными запросами и большими нагрузками на уровне кластера.

Общий порядок работы

Далее описан пошаговый процесс обработки SQL-запросов в YDB. Знание этого процесса помогает лучше понять архитектуру и внутреннее устройство YDB.



1. **Подключение к базе данных**
   Приложение использует один из официальных SDK YDB для подключения к базе данных. SDK автоматически управляет пулом сессий — логических подключений, необходимых для выполнения запросов. Каждая сессия физически связана с одним из узлов кластера. Когда требуется выполнить запрос, SDK предоставляет готовую сессию из пула, избавляя разработчика от необходимости ручного управления соединениями.
2. **Начало транзакции и отправка запроса**
   Используя активную сессию, приложение может начать транзакцию и сформировать запрос на языке YQL согласно своей бизнес-логике, после чего отправить его в кластер YDB.
3. **Парсинг и поиск в кеше планов**
   На серверной стороне узел YDB, получивший запрос, сначала проверяет его корректность (парсинг и анализ). Затем система проверяет наличие готового физического плана выполнения в кеше запросов. Если план найден, он используется повторно.
4. **Оптимизация и подготовка плана**
   Если подходящего плана нет, оптимизатор запросов формирует новый физический план, определяющий наиболее эффективный способ выполнения запроса в распределённой системе. Подробнее о принципах оптимизации запросов и видах планов см. в статье Оптимизация запросов в YDB.
5. **Распределённое выполнение запроса**
   Согласно подготовленному физическому плану YDB начинает распределённое выполнение запроса: обработка делится между несколькими узлами, каждый из которых отвечает за свою часть расчетов или обращения к данным согласно полученному плану. Такой параллелизм обеспечивает высокую скорость и масштабируемость выполнения даже для объёмных выборок.
6. **Потоковая передача результатов клиенту**
   Если запрос возвращает результат (`SELECT` и т. п.), он поступает в приложение в виде одного или нескольких результирующих наборов, которые представляют собой строго типизированные таблицы. Данные передаются потоково (частями), это позволяет обрабатывать результаты сразу по мере поступления и эффективно работать с большими выборками, не загружая всю выборку в память.
7. **Продолжение или завершение транзакции**
   После получения результатов приложение может либо продолжить транзакцию, отправив дополнительные запросы в её контексте, либо завершить её фиксацией изменений (commit).

Более подробное описание перечисленных этапов и связанных с ними понятий приведено в отдельных разделах ниже.

Сессии

Сессия в YDB — это логическое "соединение" с базой данных, которое хранит контекст, необходимый для выполнения запросов и управления транзакциями. Внутри сессии поддерживается состояние транзакций и другая рабочая информация, что позволяет выполнять связанные друг с другом запросы как часть одной транзакции. Большинство операций с запросами выполняется в контексте активной сессии.

Сессии являются долгоживущими объектами. Одна из их важных задач — эффективное распределение нагрузки: за счёт распределения сессий и связанных с ними запросов по разным узлам кластера YDB достигает высокой доступности и масштабируемости.

На практике создавать, переиспользовать и удалять сессии вручную не требуется. Все официальные SDK для YDB предоставляют встроенный пул сессий: SDK сам управляет жизненным циклом сессий, создаёт их по мере необходимости, повторно использует и возвращает обратно в пул — всё это прозрачно для пользователя и не требует дополнительной логики в приложении.

Для принудительного завершения сессии, например с долгим запросом, используйте `KILL SESSION`. Найти её идентификатор можно в системном представлении `.sys/query_sessions`.

Транзакции

Каждый запрос в YDB выполняется в контексте транзакции, что обеспечивает согласованность и надёжное хранение данных. Транзакциями можно управлять явно (через отдельные вызовы в SDK) или указывать соответствующие флаги во время выполнения запроса.

YDB также поддерживает интерактивные транзакции, которые дают возможность выполнять несколько запросов в рамках одной транзакции, позволяя при этом вашему приложению выполнять пользовательскую логику между этими запросами. Это позволяет строить сложные рабочие процессы, в которых требуется рассматривать несколько связанных операций как единую атомарную единицу.

Для получения подробной информации о транзакциях и доступных режимах транзакций в YDB смотрите статью Транзакции.

Повторные выполнения (Retries)

В YDB применяется механизм оптимистичных блокировок для управления транзакциями. Это означает, что транзакция может быть прервана во время выполнения, если выявлен конфликт и система не может гарантировать нужный уровень изоляции — например, если две транзакции одновременно изменяют одни и те же данные. Помимо конфликтов, в распределённой среде возможны временная недоступность отдельных узлов из-за сетевых сбоев, отказов оборудования или технических работ, что также может привести к необходимости повторного выполнения транзакции.

Повторное выполнение всегда следует реализовывать на уровне всей транзакции, а не отдельного запроса. В интерактивных транзакциях последовательность выполнения и промежуточные результаты отдельных запросов могут влиять на последующие действия. Поэтому, если запрос завершился ошибкой из-за конфликта или временной ошибки, необходимо повторять всю транзакцию с самого начала, чтобы обеспечить корректность и консистентность данных.

Все официальные SDK для YDB предоставляют встроенные механизмы управления транзакциями и повторным выполнением, упрощающие написание приложений. Используя стандартные механизмы повторного выполнения транзакций из SDK, вы автоматически получаете корректную реализацию логики повторного выполнения без необходимости реализовывать её вручную. Подробнее о механизмах повторного выполнения для различных SDK см. Обработка ошибок.

Язык запросов

Запросы для YDB пишутся на YQL — SQL-диалекте, специально адаптированном для распределённых масштабируемых баз данных. Хотя YQL не полностью совместим с ANSI SQL, он во многом сохраняет знакомый синтаксис и принципы SQL, что облегчает обучение и переход для опытных пользователей SQL. Полная справка по языку приведена в документации по YQL.

Большинство операций с данными в YDB осуществляется именно через YQL — он является основным инструментом для работы с данными и администрирования базы. Владение YQL позволяет использовать все возможности распределённой архитектуры YDB и реализовывать сложную бизнес-логику непосредственно в запросах.

YQL поддерживает все основные конструкции SQL, в числе которых:

- Язык манипулирования данными (DML): `SELECT`, `INSERT`, `REPLACE`, `UPDATE`, `DELETE`, `UPSERT`.
- Язык определения данных (DDL): `CREATE`, `ALTER`, `DROP` для таблиц, индексов и других объектов схемы.
- Соединения — все стандартные виды соединений, а также специальные типы соединений (например, `LEFT SEMI`, `RIGHT SEMI`, `ANY`).
- Агрегации — группировка (`GROUP BY`) и оконные функции.
- Именованные выражения для структурирования текста запроса.
- Большое количество встроенных функций для обработки разных типов данных и решения сложных задач прямо в запросе.
- Прагмы (pragma) и подсказки (hints) для управления планом выполнения.

Результирующие наборы

Результатом выполнения запроса в YDB может быть один или несколько результирующих наборов (result sets). Результирующий набор похож на таблицу: он содержит строки со строгой типизацией данных в каждом столбце. Строгая типизация результатов обеспечивает предсказуемость и согласованность формата выдачи.

Результирующие наборы могут содержать произвольно большой объём данных, поэтому для их эффективной передачи YDB использует потоковую выдачу (streaming) — результат возвращается на клиент порциями (chunks). Это позволяет сразу начинать обработку данных без ожидания получения всей выборки и минимизирует использование оперативной памяти клиентским приложением.

Ограничения

При работе с запросами в YDB важно учитывать ряд ограничений:

- **Отсутствие схемных транзакций**
  YDB не поддерживает схемные транзакции, поэтому DDL-операции (создание или изменение таблиц) нельзя объединять с DML-запросами (вставка, изменение или удаление данных) в одной транзакции или одном запросе.
- **Большие обновления и оптимистичные блокировки**
  YDB применяет механизм оптимистичных блокировок. При попытке выполнить очень большие `UPDATE` или `DELETE` в рамках одной транзакции вероятность конфликтов по блокировкам сильно возрастает, делая такие операции непрактичными для реального использования. Для больших изменений рекомендуется использовать `BATCH UPDATE` / `BATCH DELETE`.
- **Ограничения на размер транзакции**
  Объём данных, записываемых в одной транзакции, ограничен. Подробности см. в разделе Ограничения при выполнении запросов.

Полный список ограничений системы приведён в Ограничения базы данных.

Оптимизация запросов в YDB

В YDB используются два типа оптимизаторов запросов: оптимизатор, основанный на правилах, и стоимостной оптимизатор. Стоимостной оптимизатор применяется для сложных запросов, как правило, аналитических (OLAP), а оптимизация по правилам работает на всех запросах.

План запроса представляет собой граф операций, таких как чтение данных из источника, фильтрация потока данных по предикату или выполнение более сложных операций, таких как JOIN и GROUP BY. Оптимизаторы в YDB принимают на вход начальный план запроса и преобразуют его в более эффективный план, эквивалентный начальному с точки зрения возвращаемого результата.

Оптимизатор запросов основанный на правилах

Значительная часть оптимизаций в YDB применима практически для любых планов запросов, что устраняет необходимость в анализе альтернативных планов и их стоимости. Оптимизатор, основанный на правилах состоит из набора эвристических правил, которые применяются всегда, когда это возможно. Например, практически в любом запросе полезно отфильтровывать неиспользуемые данные как можно раньше в плане исполнения. У каждого правила оптимизатора есть условия срабатывания и логика переписывания плана. Правила применяются итеративно до тех пор, пока существуют правила, применимые к текущему плану.

Стоимостной оптимизатор запросов

Для более сложных оптимизаций, таких как выбор оптимального порядка и алгоритмов JOIN-ов, используется стоимостной оптимизатор. Для каждого запроса стоимостной оптимизатор рассматривает большое количество альтернативных планов выполнения и выбирает из них лучший на основе оценки стоимости каждого варианта. На текущий момент этот оптимизатор работает только с планами, где есть операции JOIN. Он выбирает наилучший порядок исполнения этих операций, а также выбирает самую эффективную реализацию для каждого Join в плане.

Стоимостной оптимизатор состоит из трех основных компонент:

- Модуль перебора порядка соединений;
- Модуль оценки стоимости плана;
- Модуль статистики, на которую опирается оценка стоимости.

Перебор порядка соединений

Текущий стоимостной оптимизатор в YDB перебирает все полезные варианты соединения таблиц с помощью оператора Join, для которых определены условия соединений. Берется исходный план запроса, из которого строится граф соединений. В зависимости от исходного запроса могут получиться разные топологии этого графа, которые существенно влияют на объем множества альтернативных планов, которое требуется сравнить.

Например, вот типичный пример популярной топологии «звезда», где основная таблица фактов соединяется с многочисленными таблицами измерений:

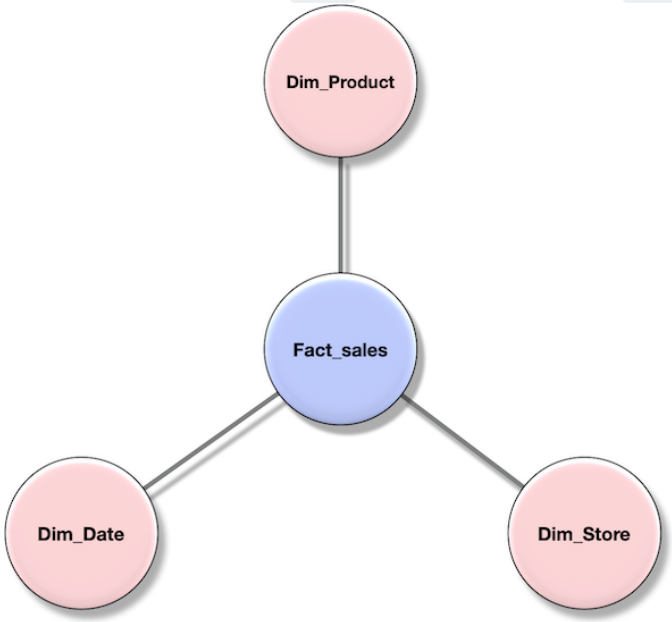
```
SELECT
    P.Brand,
    S.Country AS Countries,
    SUM(F.Units_Sold)

FROM Fact_Sales F
INNER JOIN Dim_Date D    ON (F.Date_Id = D.Id)
INNER JOIN Dim_Store S   ON (F.Store_Id = S.Id)
INNER JOIN Dim_Product P ON (F.Product_Id = P.Id)

WHERE D.Year = 1997 AND  P.Product_Category = 'tv'

GROUP BY
    P.Brand,
    S.Country
```

В графе этого запроса все таблицы `Dim...` соединяются с таблицей фактов `Fact_Sales`:



К типичным топологиям также относятся «цепочка» и «клика». «Цепочка» - это топология, где таблицы соединены друг с другом последовательно и каждая таблица участвует не более, чем в одном соединении. «Клика» — полностью связанный

граф, где каждая таблица соединяется с другой.

На практике в OLAP запросах часто встречается топология, представляющая из себя комбинацию «звезы» и «цепочки», сложные топологии вроде «клики» встречаются очень редко.

Топология сильно влияет на количество альтернативных планов, которые требуется рассмотреть оптимизатору. Следовательно, стоимостной оптимизатор ограничивает количество соединений, которые сравниваются полным перебором, в зависимости от топологии исходного плана. Возможности точной оптимизации в YDB приведены в следующей таблице:

Топология	Кол-во поддерживаемых соединений
Цепочка	110
Звезда	18
Клика	15

YDB использует модификацию алгоритма DPHyp для перебора порядка соединений. Это самый современный алгоритм динамического программирования для оптимизации запросов. Посредством построения гиперграфов запросов он избегает перебора лишних альтернатив и позволяет оптимизировать планы с операторами `JOIN`, сложными предикатами, а также с операторами `GROUP BY` и `ORDER BY`.

Оценка стоимости планов

Сравнение сложности планов основано на стоимостной функции, которая оценивает ресурсоёмкость каждой операции плана. Основные параметры стоимостной функции — это прогнозы размера входных данных для каждого оператора и размера его результата. Эти прогнозы выполняются на основе статистики, собранной по таблицам YDB, а также анализа самого плана.

Статистики для стоимостного оптимизатора

Стоимостной оптимизатор использует статистики только по таблице целиком, сбор статистик по отдельным колонкам находится в разработке. YDB собирает и поддерживает статистку в актуальном состоянии в фоновом режиме. Вы можете принудительно запустить сбор статистики, используя команду ANALYZE.

Текущий набор статистик по таблицам:

- Количество записей таблицы;
- Размер таблицы в байтах.

Текущие уровни стоимостной оптимизации

В YDB можно выставить уровень стоимостной оптимизации через прагму CostBasedOptimizationLevel

В частности, чтобы принудительно включить стоимостной оптимизатор для текущего запроса, необходимо использовать прагму `PRAGMA ydb.CostBasedOptimization = "on";`. Если требуется отключить стоимостной оптимизатор, значение прагмы следует установить в `off`.

Многоверсионное управление конкурентным доступом (MVCC)

Эта статья описывает, как многоверсионное управление конкурентным доступом (MVCC, Multi-Version Concurrency Control) используется в YDB.

Транзакции в YDB

По умолчанию транзакции YDB выполняются с уровнем изоляции "serializable" (сериализуемость). С некоторой степенью упрощения сериализуемость транзакций означает, что они могут быть с тем же самым результатом выполнены в виде строгой последовательности операций без пересечений во времени. С технической точки зрения порядок выполнения транзакций может быть произвольным, однако YDB дополнительно гарантирует отсутствие устаревших чтений: все изменения данных, закоммиченные до начала транзакции, будут видимы в этой транзакции. YDB поддерживает интерактивное выполнение транзакций (участки логики приложения могут выполняться между отдельными запросами в рамках транзакции), и использует оптимистичные блокировки для обнаружения конфликтов в работе конкурентно выполняемых модифицирующих транзакций. Если две транзакции пересекаются во времени и конфликтуют (например, обе транзакции читают запись по одному и тому же ключу, а затем пытаются изменить прочитанную запись), то одна из двух транзакций будет успешно подтверждена, а другая будет прервана, и должна быть повторена позднее.

YDB - распределенная СУБД, в которой таблицы делятся на шарды на основе диапазонов значений первичного ключа. Каждый шард может содержать до 2 Гигабайт пользовательских даных. Управление данными шарда таблицы реализуется специальным программным компонентом - таблеткой DataShard, экземпляр которого запускается на одном из вычислительных узлов кластера YDB. Таблетка DataShard представляет собой отказоустойчивую реплицируемую машину состояний (RSM, replicated state machine), использующую надежное общекластерное хранилище (blob storage) для данных и журнала изменений. Таблетки DataShard могут быстро перемещаться между вычислительными узлами, и реализуют низкоуровневые программные интерфейсы (API) для доступа к данным и поддержки распределенных транзакций.

Реализация распределенных транзакций в YDB основана на идеях Calvin, в котором распределение детерминированных транзакций между несколькими участниками осуществляется с использованием предопределенного порядка выполнения транзакций. Каждый участник получает на обработку подмножество общего потока транзакций, отбирая транзакции, затрагивающие конкретного участника. Поскольку каждый участник получает детерминированные транзакции в одном и том же относительном порядке, и принимает одинаковое решение - подтвердить или отменить её выполнение, вся база данных в конечном итоге достигает согласованного детерминированного состояния, несмотря на то, что разные участники могут обрабатывать свои подмножества потока транзакций с разными скоростями. Важно понимать, что транзакции YDB всегда работают с согласованным состоянием данных на момент своего выполнения.

У детерминированных распределенных транзакций есть ограничение: необходимо заранее знать состав участников. Однако YDB использует их только как "строительные блоки", разбивая пользовательские транзакции на фазы, и гарантируя корректность через блокировки. В современном YDB выполнение в виде распределенной транзакции требуется только для итоговой операции подтверждения изменений в конце пользовательской транзакции. Все остальные фазы выполнения YDB трансформирует в простые операции, затрагивающие только один шард, не нарушая при этом гарантии уровня изоляции "serializable".

Наличие предопределенного порядка выполнения транзакций может стать узким местом для параллельного выполнения операций. Например, при наличии в последовательности медленной транзакции, содержащей ожидание внешнего условия или доступности данных, более поздние транзакции в той же последовательности также оказываются ожидающими. Тем самым возникает потребность поддержать возможность изменения порядка выполнения не конфликтующих друг с другом транзакций без нарушения видимых пользователю гарантий.

Опыт практической эксплуатации ранних версий YDB показал, что для обеспечения высокой производительности важно обеспечить параллельное выполнение не только для полностью независимых, но и для частично конфликтующих транзакций. Например, выполнение широкого (многострочного) чтения данных одной транзакцией ранее могло блокировать операции записи данных в тот же диапазон ключей в других транзакциях, фактически останавливая их выполнение на период чтения данных с диска первой транзакцией. Реализация MVCC в YDB позволила снять это ограничение.

Что такое MVCC

Многоверсионное управление конкурентным доступом (MVCC, Multi-Version Concurrency Control) -- это способ повысить параллелизм обращений к базе данных за счет хранения нескольких версий данных одной и той же строки в том виде, в каком они были в разные моменты времени. Это позволяет читателям продолжать чтение из снимка состояния данных, не блокируя писателей. Базы данных, реализующие MVCC, не перезаписывают существующие строки, вместо этого они создают модифицированные копии изменяемых строк с добавлением версионной информации. Старые версии строк удаляются процессами сборки мусора в тот момент, когда они оказываются не нужны ни одному из активных или потенциальных читателей.

Простым и прямолинейным вариантом добавления поддержки MVCC в отсортированное хранилище пар ключ-значение является хранение нескольких значений для каждого ключа. Это может быть обеспечено добавлением суффикса с данными о версии к значению ключа записи и пропуском неактуальных для текущей транзакции или снимка версий при чтении данных. Многие базы данных в том или ином виде используют этот подход для реализации MVCC. Стоит помнить, что описанный прямолинейный подход ведет не только к раздуванию объема хранимых данных (поскольку требуется хранить множество копий одних и тех же данных, а поиск и удаление старых версий записей - ресурсоёмкая задача). Что важнее, производительность запросов может снижаться из-за необходимости сканирования и пропуска не подходящих под запрос версий обрабатываемых записей.

Почему поддержка MVCC необходима в YDB

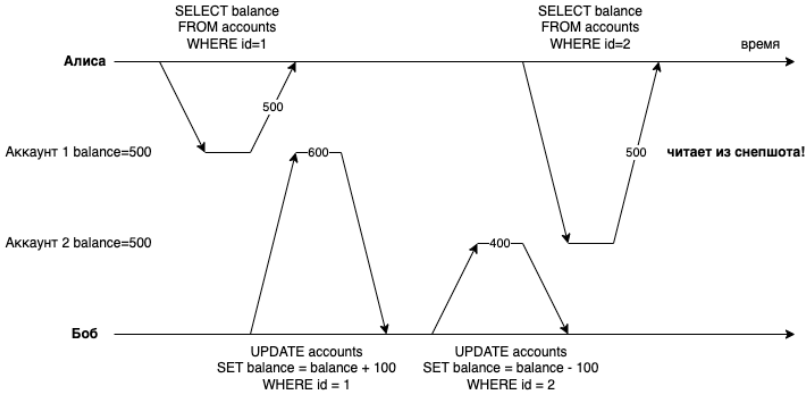
Шарды таблиц YDB хранят данные в отсортированном хранилище пар ключ-значение, реализованном как оптимизированное для записи LSM-дерево. Ранние версии YDB не использовали MVCC. Последовательность выполнения транзакций является предопределенной (что обеспечивается Координаторами -- аналогами компонентов "sequencer" в оригинальной публикации Calvin), но YDB активно меняет последовательность выполнения операций на каждом участнике, что допустимо при соблюдении внешних гарантий и отсутствии изменений конечного результата обработки. При отсутствии поддержки MVCC возможности по изменению порядка выполнения транзакций ограничены конфликтами между операциями чтения и записи данных: например, операция записи не может быть начата до завершения операции широкого чтения. Применение MVCC позволяет выполнять операции записи, не дожидаясь завершения конфликтующих операций чтения, и только операции чтения должны дождаться завершения предшествующих им операций записи. Таким образом, MVCC упрощает механизм изменения последовательности выполнения операций и повышает общую пропускную способность.

Время	Оператор	Без MVCC	MVCC	Описание
v1000:123	`UPSERT`	☑️	☑️	Выполнен
v1010:124	`SELECT`	⏳	⏳	Чтение данных
v1020:126	`UPSERT`	❌	☑️	UPSERT без MVCC ждет завершения SELECT

v1030:125	SELECT	❌	⏳	SELECT ждет завершения UPSERT перед чтением данных

Отсутствие поддержки MVCC заставляло ранние версии YDB применять дополнительные меры для обеспечения согласованности данных. Интерактивная транзакция может состоять из нескольких операций чтения, каждая из которых выполняется в свой момент времени. Чтобы исключить изменение уже прочитанных данных конфликтующими транзакциями в период времени между выполненной операцией чтения и моментом подтверждения текущей транзакции, YDB выполняла дополнительные проверки с использованием оптимистических блокировок. Использование этого механизма достаточно часто приводило к принудительной отмене и вынужденному повтору читающих транзакций на стороне приложения из-за ошибок сериализации, что неудобно для пользователей и снижает производительность.

Реализация MVCC с использованием глобальных номеров версий (доступных для использования при выполнении детерминированных распределенных транзакций) позволяет YDB выполнять операции чтения с использованием глобально согласованного снимка данных. Использование снимка для чтения данных позволяет избежать конфликтов между операциями чтения и записи, и полностью исключить возможность ошибок сериализации для операций чтения. Это улучшает пропускную способность баз данных в нагрузках с интенсивным чтением. MVCC также позволяет обеспечить повторяемость чтения в транзакциях, когда несколько операций чтения данных из одной и той же таблицы в транзакции всегда возвращают одинаковый (глобально согласованный) результат.

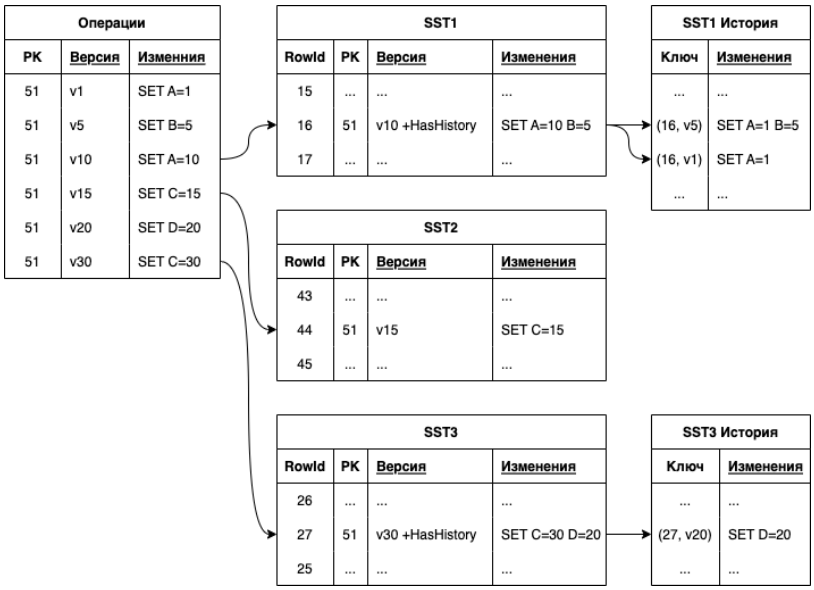


Организация хранения данных MVCC

Таблетка DataShard хранит один шард таблицы в виде оптимизированного для записи LSM-дерева, в котором для каждого первичного ключа хранится список операций изменения значений колонок строки. Актуальное состояние строки формируется каждый раз при обращении путем объединения индивидуальных операций изменения значений. Фоновая компактификация аналогичным образом объединяет изменения значений на нескольких уровнях LSM-дерева, и записывает агрегированное состояние каждой строки.

При добавлении поддержки MVCC в YDB одной из целей было исключить деградацию производительности для существующих нагрузок, типичным примером которых являются запросы актуального состояния записей в таблице с указанием диапазона значений первичного ключа. Наивный метод реализации MVCC с добавлением версионного суффикса к первичному ключу не рассматривался разработчиками, поскольку он неизбежно ведет к снижению производительности запросов. Вместо этого YDB всегда хранит в основной SST-структуре (sorted string table, часть LSM-дерева) актуальные версии строк. При наличии нескольких версий строки ее наиболее актуальная версия помечается специальным флагом наличия "истории изменения". Более старые версии помеченных таким образом строк хранятся в дополнительной SST-структуре, при этом версионные данные для каждой строки отсортированы в порядке убывания номера версии.

При чтении данных из снимка YDB проверяет, не является ли актуальная версия строки в основной SST-структуре "слишком свежей", и при необходимости выполняет двоичный поиск в SST-структуре с историческими данными. Изменения, содержащиеся в найденной версии строки, соответствующей текущему снимку, используется для восстановления актуального состояния данных строки. Поскольку уровни LSM-дерева примерно соответствуют времени их создания, поиск версий строки в SST-структуре для текущего уровня можно прекратить, как только удается обнаружить подходящую версию. Для построения полного состояния строки на момент снимка необходимо найти в каждом из более старых уровней LSM-дерева наиболее свежую версию этой строки. Таким образом, количество операций объединения версий строки оказывается ограничено количеством уровней LSM-дерева, которое обычно невелико.



Строки, хранимые в SST-структурах, не содержат полного набора атрибутов, а фактически содержат только изменения данных. Каждая строка при хранении в конкретной SST-структуре содержит полный набор изменений данных строки относительно наиболее актуальной версии той же строки в предыдущей SST-структуре в рамках LSM-дерева. Такой избыточный формат хранения помогает сделать поиск данных и компактификацию более эффективными.

Представим себе гипотетическую ситуацию, когда пользователь выполняет 1 миллион обновлений строки с ключом K, каждый раз изменяя лишь одну из множества колонок таблицы. Для повышения эффективности операций записи YDB избегает считывания состояния строки в момент обновления данных, а вместо этого фиксирует изменение в формате "обновить колонку C для ключа K". Версии строки накапливаются в оперативной памяти в структуре MemTable, и с некоторой периодичностью сбрасываются в новые SST-структуры, каждая из которых в нашем примере будет содержать некоторое (скорее всего, достаточно большое) количество версий изменяемой строки. Если бы каждая версия строки содержала лишь изменения по сравнению с предыдущей версией, то в итоге для построения актуального состояния строки потребовалось бы объединить данные из 1 миллиона накопленных версий, просканировав значительный объём данных. Вместо этого YDB на уровне каждой SST-структуры хранит агрегированные изменения строк, начиная с MemTable как нулевого уровня LSM (в котором предыдущее состояние строки всегда доступно в оперативной памяти, что исключает необходимость чтения с диска).

При компактификации, когда нескольких уровней SST собираются в новую SST-структуру, каждая измененная строка объединяется со своими наиболее актуальными версиями из более старых SST. Тем самым сложность сокращается в части количества операций объединения строк (количество которых для каждой строки оказывается ограничено количеством уровней LSM-дерева) как при компактификации, так и при чтении данных. В то же время сохраняется возможность выполнять "слепую запись", не восстанавливая состояние строки при изменении ее колонок.

PK=51 при различных версиях чтения			
Версия	**SST3**	**SST2**	**SST1**
v35	SET C=30 D=20	нет изменений (C=15 перезаписано C=30)	SET A=10 B=5
v25	SET D=20 нашли (27, v20) <= v25	SET C=15	SET A=10 B=5
v19	нет изменений для v19	SET C=15	SET A=10 B=5
v7	нет изменений для v7	нет изменений для v7	SET A=1 B=5 нашли (16, v5) <= v7

В конечном счете старые диапазоны версий помечаются как устаревшие и недоступные для чтения, что позволяет автоматически удалять неиспользуемые версии строк в процессе компактификации: недоступные версии пропускаются при компактификациях и не попадают в новые SST-структуры. Для каждой SST-структуры YDB поддерживает небольшую гистограмму хранимых версий, что позволяет своевременно обнаруживать накопление излишнего количества неиспользуемых версий в LSM-дереве и автоматически запускать дополнительные операции компактификации для сборки версионного мусора.

Как MVCC используется в YDB

Применение MVCC позволяет таблеткам DataShard более свободно изменять порядок выполнения транзакций. Работа с глобальными снимками данных дает еще больше возможностей, поэтому YDB использует глобально-координируемые значения системного времени в качестве меток версий, которые соответствуют глобальному логическому порядку выполнения операций, поддерживаемому механизмом детерминированных распределенных транзакций. Это позволяет создавать глобальные снимки путем выбора корректного значения системного времени. Чтение данных из таблеток DataShard с указанием снимка позволяет получить согласованное состояние всей базы данных по состоянию на конкретный момент времени.

Перед выполнением первой операции чтения в интерактивной (многошаговой) транзакции YDB выбирает метку времени снимка данных, для которой гарантировано завершение всех ранее подтвержденных транзакций. В существующей реализации метка времени снимка вычисляется на основе последней метки времени, разосланной Координаторами. Вычисленная метка времени может оказаться в недалеком будущем из-за того, что часть передаваемых от Координаторов в таблетки DataShard транзакций могла не успеть начать свое выполнение. Сдвиг метки времени в будущее обычно оказывается очень небольшим, а для передаваемых транзакций YDB ожидает, что они вскоре начнут свое выполнение просто в силу того, что они уже передаются. YDB предоставляет важную гарантию, что выбранная метка времени будет больше или равна всех меток времени подтверждения транзакций, которые могли быть видимы пользователю. Таким образом, при генерации меток времени для новых операций чтения обеспечивается видимость всех ранее завершенных (подтвержденных) операций записи.

Снимок, соответствующий вычисленной метке времени, используется для всех последующих операций чтения в этой же транзакции без какой-либо дополнительной координации между участниками, поскольку снимки обеспечивают глобальную согласованность на уровне базы данных в целом. Чтобы гарантировать, что все значимые изменения учтены в снимке, таблетка DataShard проверяет наличие потенциально конфликтующих пишущих транзакций, которые должны завершиться перед выбранным снимком, и дожидается их завершения перед началом чтения. Снимок гарантирует отсутствие изменений в данных со стороны конкурентно выполняемых транзакций (повторяемость чтения), но эта гарантия относится только к уже наблюдаемым изменениям. Данные за пределами текущей области наблюдения снимка могут меняться до тех пор, пока системное время не продвинется достаточно далеко за границу этого снимка для каждого из участников. Как интересный побочный эффект, некоторые более поздние (по отношению к снимку) пишущие транзакции могут быть переупорядочены перед фактическим доступом к данным снимка, что допускается на уровне изоляции Serializable.

Все изменения данных, выполняемые интерактивной транзакцией, буферизуются до момента подтверждения транзакции. В момент подтверждения распределенной транзакции выполняется проверка (с использованием оптимистических блокировок) на отсутствие конфликтующих изменений, выполненных конкурирующими транзакциями между моментом создания снимка и моментом подтверждения транзакции. Такая проверка имитирует перемещение всех операций чтения во времени к моменту подтверждения транзакции. Подтверждение транзакции производится только тогда, когда конфликтующие изменения отсутствуют.

Интерактивные транзакции, выполняющие чтение исторических (заведомо устаревших) данных из своего снимка, автоматически помечаются флагом "только для чтения", поскольку для них заранее известно, что соответствующие операции чтения нельзя будет переместить во времени на будущий момент подтверждения транзакции. При попытке выполнения в такой транзакции операций записи возвращается ошибка сериализации без каких-либо буферизаций или дополнительных взаимодействий. Если же транзакция в конечном итоге оказывается только читающей, то при ее завершении производится освобождение ресурсов и возврат успешного статуса пользователю, поскольку чтение из согласованного снимка данных не приводит к нарушению гарантий сериализации. До реализации в YDB поддержки основанных на MVCC глобальных снимков данных требовалось проверять состояние блокировок для всех транзакций, что создавало сложности с выполнением многострочных операций чтения на фоне высокой нагрузки на запись.

Сохранение высокой производительности на нагрузках ключ-значение

Наличие в YDB поддержки сложных распределенных транзакций, затрагивающих множество таблиц и их шардов, не отменяет потребности в высокой производительности простых одношардовых транзакций. Такие транзакции применительно к YDB называются "немедленными" ("immediate"), и исторически (до реализации MVCC) для них не использовалась какая-либо дополнительная координация. Например, если транзакция читает или пишет данные по одному конкретному ключу в одной таблице, то при ее выполнении требуется взаимодействие с единственным шардом таблицы, к которому принадлежит этот ключ, и больше ни с чем.

На первый взгляд, такой подход противоречит использованию глобальных меток времени для реализации MVCC. Но если транзакция ограничена рамками конкретного шарда, то оказывается достаточным использование локальных (не

координируемых) меток времени, которые могут отличаться для разных шардов. Метки времени транзакций присваиваются таким образом, чтобы используемые локальные метки времени можно было использовать как корректные глобальные метки времени.

Для каждой распределенной транзакции присваиваемая метка времени представляет собой пару чисел (step, txId). В этой паре step - номер шага координатора (coordination tick), обычно автоматически увеличиваемый каждые 10 миллисекунд, а txId - глобально уникальный идентификатор транзакции. Немедленные транзакции не могут заполнить поле step корректным значением, поскольку они не обрабатываются координаторами. Для значения txId обеспечивается уникальность, но не предоставляется гарантий монотонного возрастания значений, поэтому их нельзя использовать при записи данных из-за возможного отнесения данных к моментам времени в прошлом.

Поскольку для немедленных транзакций отсутствует требование повторяемости, их меткам времени не обязательно быть уникальными. Это позволяет использовать в качестве метки времени немедленной транзакции метку времени другой существующей транзакции в очереди выполнения, в результате чего немедленная транзакция оказывается выполнена "непосредственно перед" распределенной транзакцией с такой же меткой времени. Если же очередь выполнения оказывается пустой, то в качестве step используется время, разосланное координаторами (подтвержденный всеми шардами максимальный номер шага координации, что гарантирует отсутствие транзакций до конца этого шага), а в качестве txId - максимально возможное значение, что обеспечивает выполнение немедленной транзакции "непосредственно перед" завершением текущего шага координации.

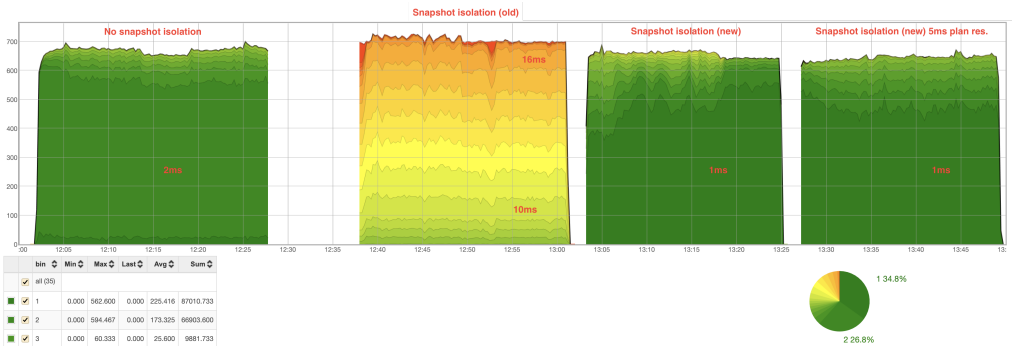
Использование описанных выше неуникальных меток времени для немедленных транзакций гарантирует, что всегда найдется некоторое новое неубывающее значение метки времени, позволяющее выполнить очередную немедленную транзакцию без задержек (если не учитывать другие ограничения, связанные с изменением порядка выполнения операций). Повторное использование меток времени дополнительно облегчает компактификацию для случая быстро меняющихся записей, поскольку все версии строки, порождаемые тысячами немедленных транзакций с одинаковой меткой времени, оказываются сжаты в единственную финальную версию. Таким образом, YDB хранит в истории изменений только те версии строк, для которых требуется согласованность на уровне нескольких шардов.

Приоритет операций чтения

Когда разработчики YDB впервые включили поддержку снимков MVCC, при тестировании они столкнулись с неожиданной головоломкой: на некоторых рабочих нагрузках произошло серьезное увеличение задержек чтения данных. YDB отсчитывает системное время в виде шагов координации транзакций каждые 10 миллисекунд (такое значение было подобрано в соответствии с типичными задержками взаимодействия между центрами обработки данных, составляющими около 4 миллисекунд). Как было указано в предыдущем разделе, одношардовые операции записи используют "текущее время" для формирования метки времени транзакции. Конкурентные операции чтения из снимков данных должны были ждать следующего шага координации для того, чтобы используемые в них метки времени снимков оказались "закрытыми" для новых операций записи. Результатом изменений стало вполне объяснимое, но неприемлемое увеличение задержек на операциях чтения.

Для решения проблемы с возросшими задержками операций чтения потребовались дополнительные меры, позволяющие повысить приоритет операций чтения без чрезмерного замедления операций записи. Реализованное в настоящее время решение "закрывает" возможность записи для выбранной метки времени снимка в момент начала чтения из этого снимка. Для одношардовых операций записи выбирается метка времени в "будущем", которая исключает возможность повреждения ранее созданных снимков (что обеспечивает повторяемость чтения). Поскольку метка времени одношардовых операций записи оказывается в "будущем", и поскольку создаваемые новые снимки должны включать в себя данные только подтвержденных транзакций, то YDB задерживает ответы с подтверждением выполненной записи до момента соответствия выбранной "будущей" и актуальной меток времени. В результате YDB ведет себя так, будто такие ожидающие транзакции не подтверждены (например, при выполнении конкурентных одношардовых операций чтения), до момента соответствия метки времени записи транзакции текущему времени.

В момент, когда требуется дополнительное ожидание, изменения одношардовой транзакции уже передаются для записи в хранилище, и обычно должны быть сохранены как минимум в еще одном центре обработки данных, поэтому дополнительное ожидание почти не влияет на средние задержки одношардовых транзакций записи даже при наличии большого количества конкурентных чтений из снимков. В то же время для транзакций чтения величина задержек значительно уменьшилась:



bin	Min	Max	Last	Avg	Sum
all (35)					
1	0.000	562.600	0.000	225.416	87010.733
2	0.000	594.467	0.000	173.325	66903.600
3	0.000	60.333	0.000	25.600	9881.733

Вторичные индексы

В YDB автоматически создается индекс по первичному ключу, поэтому выборки с условием по первичному ключу всегда выполняются эффективно, затрагивая только требуемые строки. Выборка с условием, наложенным на одну или несколько неключевых колонок, как правило, приводит к полному сканированию таблицы. Для того чтобы такие выборки были эффективными, нужно использовать *вторичные индексы* — глобальные структуры с отдельной индексной таблицей.

Отдельно существуют локальные индексы: вспомогательные структуры, которые хранятся вместе с данными таблицы и применяются при чтении на стороне хранилища, не материализуя отдельную индексную таблицу (см. раздел Локальные индексы ниже).

В текущей версии YDB реализованы *синхронные* и *асинхронные* глобальные вторичные индексы. Каждый индекс представляет собой скрытую таблицу, которая обновляется:

- для синхронных индексов — транзакционно при изменении основной таблицы;
- для асинхронных индексов — в фоне, получая необходимые изменения из основной таблицы.

Когда пользователь присылает SQL-запрос на вставку, изменение или удаление данных, БД прозрачно для пользователя формирует команды на изменение индексной таблицы. У таблицы может быть несколько вторичных индексов. Индекс может включать несколько колонок, при этом важен порядок указания колонок в индексе. Одна колонка может быть включена в нескольких индексов. В дополнение к указанным колонкам в индексе всегда неявно сохраняются значения колонок первичного ключа таблицы, чтобы от найденной записи в индексе можно было перейти к записи в таблице.

Синхронный вторичный индекс

Синхронный индекс обновляется одновременно с таблицей, которую он индексирует. Такой индекс обеспечивает строгую согласованность данных и использует для этого механизм распределенных транзакций. Таким образом, если операции чтения и слепой записи в таблицу без индекса можно выполнить без стадии планирования, и за счет этого существенно сократить задержки, то для записи в таблицу с синхронным индексом такая оптимизация невозможна.

Асинхронный вторичный индекс

Асинхронный индекс, в отличие от синхронного, не использует механизм распределенных транзакций, а в фоне получает изменения из индексируемой таблицы. Транзакции записи в таблицу с таким индексом выполняются без дополнительных затрат на планирование, за счет снижения гарантий: асинхронный индекс обеспечивает согласованность данных в конечном счете, но не строгую согласованность. Использование асинхронного индекса в транзакциях чтения возможно только в режиме Stale Read Only.

Покрывающий вторичный индекс

Имеется возможность сделать копию содержимого колонок в индекс (covering index), таким образом это исключает необходимость чтения из основной таблицы в операциях чтения по индексу, что заметно снижает задержки. В то же время такая денормализация приводит к увеличенному потреблению дискового пространства и к возможному замедлению операции вставки и обновления из-за необходимости дополнительного копирования данных.

Уникальный вторичный индекс

Этот тип индекса реализует семантику уникального значения в колонке или наборе колонок, а также, как и другие индексы, позволяет эффективно выполнять точечные чтения по набору индексируемых колонок. YDB использует его для выполнения дополнительных проверок, чтобы гарантировать, что каждое уникальное значение индексируемых колонок присутствует в таблице не более одного раза. Если модифицирующий запрос нарушает это ограничение, он отменяется со статусом `PRECONDITION_FAILED`. Таким образом, пользовательский код должен быть готов к обработке этого статуса.

Уникальный вторичный индекс является синхронным индексом, поэтому с точки зрения транзакционности процесс его обновления такой же, как у синхронного вторичного индекса, описанного выше.

Векторный индекс

Векторный индекс является особым видом вторичного индекса.

В отличие от традиционных вторичных индексов, оптимизирующих поиск по равенству или диапазону, векторные индексы позволяют выполнять векторный поиск на основе функций расстояния или схожести.

Полнотекстовый индекс

Полнотекстовый индекс является особым видом вторичного индекса.

В отличие от традиционных вторичных индексов, оптимизирующих поиск по равенству или диапазону, полнотекстовые индексы позволяют масштабируемо выполнять текстовый поиск по словам и их сочетаниям (а при использовании N-грамм — и по подстрокам). См. также: Полнотекстовый поиск.

JSON-индекс

JSON-индекс является особым видом вторичного индекса, как и полнотекстовый — оба построены поверх инвертированного индекса, но используют разные токенизаторы.

JSON-индексы позволяют ускорять предикаты с функциями JSON_EXISTS и JSON_VALUE по содержимому колонки типа `Json` или `JsonDocument`. Индекс строится с помощью разбиения JSON-документов на токены путей и пар вида «путь + значение», что позволяет находить соответствующие строки по путям JsonPath без полного сканирования таблицы. См. также: Поиск по JSON.

Локальные индексы

Локальные индексы — вспомогательные структуры, хранящиеся вместе с данными таблицы и применяемые при чтении на стороне хранилища. Они не материализуют отдельную индексную таблицу. Сейчас реализованы Блум-индексы и min_max-индекс.

Онлайн-создание вторичного индекса

В YDB доступно создание вторичного индекса, а также удаление существующего вторичного индекса без остановки обслуживания. Для одной таблицы можно создать только один индекс за раз.

Операция онлайн-создания индекса состоит из следующих шагов:

1. Взятие снапшота таблицы с данными, создание таблицы индекса с пометкой доступности для записи.

После этого шага пишущие транзакции становятся распределенными, происходит запись в основную таблицу и индекс. Для пользователя индекс пока не доступен.

2. Чтение снапшота основной таблицы и запись в индекс.

   Реализуется «запись в прошлое»: разрешаются ситуации, когда обновления данных на шаге 1 меняют данные, записанные на шаге 2.

3. Публикация результата, удаление снапшота.

   Индекс готов к использованию.

Возможное влияние на пользовательские транзакции:

- Может наблюдаться увеличение задержек из-за того, что транзакции становятся распределенными (при создании синхронного индекса).
- Возможен повышенный фон ошибок `OVERLOADED` из-за того, что во время записи данных активно работает автоматическое разделение шардов индексной таблицы.

> **Примечание**
>
> Скорость записи данных выбрана таким образом, чтобы минимизировать влияние процесса записи на пользовательские транзакции. Для управления скоростью настройте лимиты для соответствующей очереди брокера ресурсов.

Создание индекса — асинхронная операция. Если после запуска операции произойдет разрыв клиент-серверной связности, то построение индекса будет продолжено. Управлять асинхронной операцией можно через YDB CLI.

Создание и удаление вторичных индексов

Вторичный индекс может быть:

- Создан при создании таблицы командой YQL CREATE TABLE.
- Добавлен к существующей таблице командой YQL ALTER TABLE или командой YDB CLI table index add
- Удален у существующей таблицы командой YQL ALTER TABLE или командой YDB CLI table index drop.
- Удален вместе с таблицей командой YQL DROP TABLE или командой YDB CLI `table drop`.

Применение вторичных индексов

Детальная информация о применении вторичных индексов в приложениях находится в статье о них в разделе документации для разработчиков.

Векторный поиск

Понятие векторного поиска

Векторный поиск, также известный как поиск ближайшего соседа (NN), представляет собой задачу оптимизации, в которой необходимо найти ближайший вектор (или множество векторов) в данном наборе данных относительно заданного вектора запроса. Близость между векторами определяется с помощью метрик расстояния или сходства.

Одним из распространённых подходов, особенно при работе с большими наборами данных, является приближённый поиск ближайших соседей (ANN search), который позволяет получать более быстрые результаты за счёт возможной потери точности.

Векторный поиск активно используется в следующих областях:

- рекомендательные системы;
- семантический поиск;
- поиск похожих изображений;
- обнаружение аномалий;
- классификационные системы.

Кроме того, **векторный поиск** в YDB широко применяется в задачах машинного обучения (ML) и искусственного интеллекта (AI). Он особенно полезен в подходах Retrieval-Augmented Generation (RAG), которые используют векторный поиск для извлечения релевантной информации из больших объемов данных, что значительно улучшает качество генеративных моделей.

Методы решения задач векторного поиска можно разделить на три крупных категории:

- точные методы;
- приближенные методы без индекса;
- приближенные методы с индексом.

Выбор метода зависит от числа векторов и от характера нагрузки. Точные методы медленно ищут и быстро обновляются, индексы - наоборот.

Точный векторный поиск

В основе точного метода лежит вычисление расстояния от вектора запроса до всех векторов в наборе данных. Этот алгоритм, также известный как наивный подход или метод грубой силы, имеет время выполнения `O(dn)`, где `n` — количество векторов в наборе данных, а `d` — их размерность.

Точный векторный поиск лучше использовать, если полный перебор искомых векторов происходит за приемлемое время. В том числе, когда их можно предварительно отфильтровать по некоторому условию, например, по идентификатору пользователя. В таких случаях точный метод может работать быстрее, чем текущая реализация векторных индексов.

Основные преимущества:

- отсутствие необходимости в дополнительных структурах данных, таких как специализированные векторные индексы;
- полная поддержка транзакций, в том числе в режиме строгой согласованности;
- мгновенное применение операций модификации данных: вставка, обновление, удаление.

Подробнее о точном векторном поиске.

Приближенный векторный поиск без индекса

Приближенные методы могут избегать полного перебора исходных данных или снижать точность векторов. Это позволяет значительно ускорить процесс поиска, хотя и может привести к некоторому снижению качества результатов.

Скалярное квантование — это метод уменьшения точности координат с сохранением размерности векторов.

YDB поддерживает векторный поиск по векторам типов `Float`, Float16, BFloat16, `Int8`, `Uint8`, `Bit`. Следовательно, возможно применение скалярного квантования для преобразования данных из `Float` в любой из этих типов.

Скалярное квантование сокращает время, необходимое для чтения и записи данных, за счёт уменьшения числа байт. Например, при квантовании из `Float` в `Bit` объём координат сокращается в 32 раза без учёта служебных данных сериализации.

Приближенный векторный поиск без индекса использует очень простую дополнительную структуру данных - множество векторов с другим квантованием. Это позволяет использовать простой алгоритм поиска: сначала грубый предварительный поиск по сжатым векторам, а затем уточнять результаты по исходным векторам.

Основные преимущества:

- полная поддержка транзакций, в том числе в режиме строгой согласованности;
- мгновенное применение операций модификации данных: вставка, обновление, удаление.

Подробнее о приближенном векторном поиске без индекса.

Приближенный векторный поиск с индексом

Когда объем данных существенно увеличивается, подходы без индекса перестают работать за приемлемое время.
В таких случаях необходимы дополнительные структуры данных — векторные индексы, которые ускоряют процесс поиска.

Основное преимущество:

- возможность работы с большим числом векторов.

Недостатки:

- построение индекса может занимать существенное время.

Рецепты работы с векторным индексом

Для начала работы с векторным индексом можно воспользоваться следующими рецептами:

- YDB CLI & YQL

- YDB SDK: Python, C++

Полнотекстовый поиск

Понятие полнотекстового поиска

Полнотекстовый поиск — это способ находить документы по текстовому содержимому в колонке типа `String` или `Utf8` по словам и (при специальной индексации) по подстрокам. В отличие от простых строковых предикатов, полнотекстовый поиск обычно включает этапы **токенизации** и **нормализации** текста (например, приведение к нижнему регистру), что позволяет искать по "термам", а не по байтовым подстрокам. Типичные сценарии:

- поиск по каталогам и базам знаний;
- поиск по логам/сообщениям;
- поиск по описаниям и метаданным.

Ключевая идея полнотекстового индекса — **инвертированный индекс**: для каждого терма хранится список документов (первичных ключей), в которых этот терм встречается. Поэтому запросы вида "найти документы, где встречаются слова A и B" можно выполнять без полного сканирования таблицы.

В YDB полнотекстовый поиск можно выполнять двумя основными способами:

- **без индекса** — сканированием таблицы и применением строковых предикатов (например, `LIKE`). Подход прост, но плохо масштабируется: объём работы растёт вместе с размером таблицы.
- **с полнотекстовым индексом** — созданием полнотекстового индекса и выполнением запросов через `VIEW IndexName`. Этот подход предназначен для масштабируемого поиска.

Полнотекстовый поиск с полнотекстовым индексом

Полнотекстовые индексы строят инвертированный индекс по текстовой колонке и позволяют:

- выполнять полнотекстовое сопоставление через функцию FulltextMatch;
- ранжировать результаты по релевантности через функцию FulltextScore.

Поведение поиска (что считается термом, как обрабатываются регистр/формы слов, поддержка подстановок) задаётся при создании индекса через готовый анализатор или явные настройки токенизатора и фильтров (например, нижний регистр, стоп-слова, многоязычная лемматизация Snowball, N-граммы). Подробности — в документации создания индекса Полнотекстовый индекс.

Подробнее:

- Полнотекстовые индексы
- Базовые функции полнотекстового поиска
- VIEW (Полнотекстовый индекс)

Гибридный поиск

Понятие гибридного поиска

Гибридный поиск объединяет полнотекстовый поиск и векторный поиск в едином ранжированном результате. Вместо выбора одного из сигналов гибридный запрос оценивает каждый документ сразу по двум критериям — насколько его текст соответствует запросу (лексическая релевантность) и насколько его эмбеддинг близок к вектору запроса (семантическая близость) — а затем **объединяет** два ранжирования в одно.

Эти сигналы дополняют друг друга:

- **Полнотекстовый поиск** (BM25) точен для конкретных термов, имён, кодов и редких ключевых слов, но пропускает документы, в которых использованы другие формулировки.
- **Векторный поиск** (поиск ближайших соседей по эмбеддингам) улавливает семантическую близость и перефразирования, но может поставить тематически близкий документ выше того, который содержит точный введённый пользователем терм.

Их объединение даёт результаты, релевантные одновременно и лексически, и семантически. Это распространённый строительный блок для поиска по базам знаний и для этапа извлечения данных в подходе Retrieval-Augmented Generation (RAG).

Гибридный поиск в YDB

В YDB гибридный запрос выполняется по таблице, для которой созданы **оба** индекса:

- fulltext_relevance по текстовой колонке;
- vector_kmeans_tree по колонке с эмбеддингами.

Объединение задаётся в секции `ORDER BY` функцией `HybridRank`, которая принимает по одному оценивающему выражению на каждую ветвь — FullTextScore для текстовой ветви и Knn-расстояние или сходство для векторной ветви:

```
$queryText = "машинное обучение";
$queryVector = Knn::ToBinaryStringFloat([0.1, 0.2, 0.3, 0.4]);

SELECT id, title
FROM documents
ORDER BY HybridRank(
    FullTextScore(text, $queryText),
    Knn::CosineDistance(embedding, $queryVector))
LIMIT 10;
```

Каждая ветвь выполняется по своему индексу и возвращает собственный список наиболее подходящих документов. Затем YDB объединяет эти списки в итоговое ранжирование — одним из двух способов:

- **Reciprocal Rank Fusion (RRF)** — по умолчанию. Каждая ветвь вносит вклад на основе *ранга* документа в этой ветви, поэтому абсолютные значения оценок (которые несопоставимы между ветвями) не имеют значения.
- **Linear** — взвешенная сумма оценок ветвей (по умолчанию нормализованных методом min-max). Расстояния преобразуются так, чтобы меньшие расстояния давали более высокий ранг.

Веса ветвей, константа RRF и размеры пулов кандидатов настраиваются. Полный синтаксис запроса и список параметров — в разделе Гибридный поиск (HybridRank).

Подробнее:

- Использование гибридного поиска
- Синтаксис гибридного запроса (HybridRank)
- Полнотекстовый поиск
- Векторный поиск

Поиск по содержимому JSON-документов

Поиск по JSON — это способ находить строки таблицы по содержимому JSON-документа, хранящегося в колонке типа `Json` или `JsonDocument`: по существованию пути в документе и по значению, расположенному в указанном пути. Путь задаётся выражением JsonPath, а проверки выполняются уже существующими функциями JSON_EXISTS и JSON_VALUE. Типичные сценарии:

- поиск документов с определённым вложенным полем;
- поиск документов, у которых поле по заданному пути равно нужному значению;
- фильтрация полуструктурированных данных без заранее зафиксированной схемы.

В YDB поиск по JSON можно выполнять двумя основными способами:

- без индекса — сканированием таблицы и применением функций `JSON_EXISTS` / `JSON_VALUE` к каждой строке. Подход прост, но плохо масштабируется: объём работы по сканированию растёт вместе с размером таблицы.
- с JSON-индексом — созданием JSON-индекса по JSON-колонке. Этот подход предназначен для масштабируемого поиска.

Поиск по JSON с JSON-индексом

Ключевая идея JSON-индекса — построение инвертированного индекса по путям и парам «путь + значение». Каждый путь JSON-документа (а для проверок равенства — и скалярное значение по этому пути) кодируется в токен. Для каждого токена в индексе хранится список значений первичного ключа соответствующих строк таблицы. Запросы к индексу сводятся к инвертированному поиску по этим токенам — по той же схеме, что и у полнотекстового индекса, но с собственным токенизатором JSON.

JSON-индекс не хранит копию JSON-документа целиком, а раскладывает его на отдельные токены:

- для каждого пути в JSON-дереве создаётся токен «путь существует»;
- если путь ведёт к скалярному значению (строке, числу, логическому значению или `null`), дополнительно создаётся токен «путь + значение».

Массивы при этом прозрачны: элементы массива индексируются под тем же путём, что и сам массив, поэтому индекс одинаково отвечает на запросы к `$.items`, `$.items[0]` и `$.items[*]`.

Например, для документа:

```
{
    "id": 42042,
    "name": "Michael",
    "email": null,
    "items": [null, "str"],
    "parts": {
        "key": "k1",
        "value": false
    }
}
```

концептуально индексируются следующие токены (без учёта специфики внутреннего представления):

Токен	Что позволяет найти
`$`	документ не равен `NULL`
`$.id` / `$.id == 42042`	путь `id` существует / равен `42042`
`$.name` / `$.name == "Michael"`	путь `name` существует / равен `"Michael"`
`$.email` / `$.email == null`	путь `email` существует / равен `null`
`$.items` / `$.items == null` / `$.items == "str"`	путь `items` существует / содержит элемент `null` / содержит элемент `"str"`
`$.parts`	путь `parts` существует
`$.parts.key` / `$.parts.key == "k1"`	вложенный путь существует / равен `"k1"`
`$.parts.value` / `$.parts.value == false`	вложенный путь существует / равен `false`

JSON-индекс позволяет:

- находить строки по существованию пути через JSON_EXISTS;
- находить строки по значению в пути через JSON_VALUE или фильтр-предикат внутри `JSON_EXISTS`.

При выполнении запроса JSON-индекс может быть автоматически задействован оптимизатором. Достаточно написать обычное условие `WHERE` с выражениями `JSON_EXISTS` / `JSON_VALUE` по индексированной JSON-колонке — оптимизатор распознает такой предикат и применит чтение по JSON-индексу вместо перебора записей исходной таблицы. Кроме того, JSON-индекс, как и любой другой вторичный индекс, можно принудительно задействовать, указав его имя в секции `VIEW IndexName`.

Если предикат не удаётся превратить в обращение к индексу, поведение YDB зависит от того, был ли в запросе явно указан требуемый JSON-индекс:

- при автоматическом выборе индекса оптимизатором запрос просто выполняется без ускорения индексом (результат остаётся корректным);
- при явном указании индекса через выражение `VIEW` возвращается ошибка.

Подробнее см. VIEW (JSON-индекс).

Дополнительная информация:

- JSON-индексы — обзор возможностей, поддерживаемых предикатов и ограничений.
- Функции для работы с JSON — справка по `JSON_EXISTS`, `JSON_VALUE` и языку JsonPath.
- VIEW (JSON-индекс) — синтаксис запросов с JSON-индексом.

Локальные индексы

Понятие локального индекса

Локальный индекс — вспомогательная структура, которая хранится вместе с данными таблицы и применяется на этапе чтения из хранилища. Она не материализует отдельную индексную таблицу.

Локальные индексы работают как фильтры чтения основной таблицы: оптимизатор запросов и слой хранения используют их, чтобы пропускать нерелевантные фрагменты данных при сканировании.

Сейчас в YDB реализованы локальные Блум-индексы и min_max-индекс.

Блум-индексы

Блум-индексы — частный случай локального индекса, построенный на основе фильтра Блума.

При чтении для каждого фрагмента данных индекс проверяет, может ли в нём встречаться искомое значение (или набор n-грамм). Если фильтр сообщает, что значение точно не встречается, фрагмент пропускается без чтения проиндексированных колонок. Если фильтр «пропускает» проверку, значение может присутствовать — в том числе из-за ложноположительного срабатывания, — и фрагмент нужно читать. Это уменьшает объём фактически прочитанных данных при селективных запросах.

Типы блум-индексов

- `bloom_filter` — строит фильтр по точным значениям индексируемой колонки. Подходит для условий равенства (`=`), проверки вхождения в список (`IN`) и других сравнений на равенство для поддерживаемых типов.
- `bloom_ngram_filter` — строит фильтр по n-граммам строковой колонки (`String`, `Utf8`). При поиске по подстроке или шаблону (`LIKE`) запрос разбивается на n-граммы; если в индексе фрагмента нет хотя бы одной из требуемых n-грамм, в нём не может быть искомой подстроки, и фрагмент пропускается. Поддерживается только в колоночных таблицах.

Локальные блум-индексы

Тип `bloom_filter` работает как в строковых (OLTP), так и в колоночных (OLAP) таблицах, но реализация различается:

- В строковых таблицах фильтр строится как префиксный фильтр Блума по левому префиксу первичного ключа. Индексируемые колонки должны образовывать непрерывное ведущее подмножество колонок первичного ключа. Это ускоряет точечные чтения и сканы по диапазону, ограничивающие ведущие колонки ключа. Для создания префиксного фильтра Блума используйте ALTER TABLE ... ADD INDEX, для удаления — ALTER TABLE ... DROP INDEX.
- В колоночных таблицах фильтр строится по значениям индексируемой колонки в каждом фрагменте данных (порции) и применяется при аналитических сканах для пропуска фрагментов, не содержащих искомое значение.

Ограничения

- Секции `COVER (...)` и дополнительные колонки индекса не поддерживаются.
- Для колоночных таблиц индексируемая колонка должна быть одна. Для строковых таблиц допускается несколько индексируемых колонок.
- Для строковых таблиц тип индекса `bloom_ngram_filter` не поддерживается.
- На строковых таблицах индексируемые колонки `bloom_filter` должны образовывать левый префикс первичного ключа. Наборы колонок, не являющиеся префиксом, отклоняются.
- На строковых таблицах два индекса `bloom_filter` не могут иметь одинаковую длину префикса (один и тот же набор ведущих колонок первичного ключа).

Дополнительные материалы

- Блум-индексы
- ALTER TABLE ADD INDEX

min_max-индекс

min_max-индекс — частный случай локального индекса, который хранит минимальное и максимальное значение одной индексируемой колонки для каждого фрагмента данных.

При чтении с фильтром специального вида по колонке с min_max-индексом YDB сначала читает сохранённые для фрагмента минимальное и максимальное значения и проверяет, пересекается ли интервал из фильтра с этим диапазоном. Если интервалы не пересекаются, предикат гарантированно ложен на всех значениях фрагмента, поэтому фрагмент можно не читать. Это полезно для диапазонных предикатов и условий равенства (частный случай диапазонного предиката) по колонкам с небольшим разбросом между минимальным и максимальным значениями внутри хранимых фрагментов.

Пример:

В хранилище для таблицы `events` в одном из фрагментов колонки `level` типа `Int32` находятся значения `[5, 5, 9, 5, 9, 13]`. Тогда минимальное значение — 5, максимальное — 13. Для запроса `SELECT * FROM events WHERE level = 15` интервал фильтра — `[15, 15]`. Он не пересекается с интервалом `[5, 13]`, поэтому такой фрагмент можно не читать из хранилища.

Предикаты min_max-индекса

Оптимизатор может использовать min_max-индекс для предикатов `=`, `<`, `<=`, `>`, `>=`, `BETWEEN`, а также совместимых комбинаций с `AND` или `OR`.

Ограничения

- Поддерживаются только для колоночных таблиц.
- В `ON (...)` должна быть указана ровно одна колонка.
- `COVER (...)` и дополнительные колонки данных не поддерживаются.
- Специфичные параметры `WITH (...)` не поддерживаются.
- `ALTER INDEX` для min_max-индекса не поддерживается.
- Колонки типов `Json` и `JsonDocument` не поддерживаются.

Дополнительные материалы

- min_max-индекс
- ALTER TABLE ADD INDEX

Федеративные запросы

Федеративные запросы позволяют получать информацию из различных источников данных без необходимости переноса данных этих источников внутрь YDB. С помощью YQL-запросов вы можете обращаться к внешним базам данных без необходимости дублирования данных между системами.

Для работы с данными, хранящимися во внешних СУБД, достаточно создать внешний источник данных. Для работы с несхематизированными данными, хранящимися в бакетах S3, нужно дополнительно создать внешнюю таблицу. В обоих случаях необходимо предварительно создать секреты, хранящие конфиденциальные данные, необходимые для аутентификации во внешних системах.

Вы сможете узнать о внутреннем устройстве системы обработки федеративных запросов в разделе об архитектуре. Подробная информация про работу с различными источниками данных приведена в соответствующих разделах:

- S3

Экспериментальные источники данных

- ClickHouse
- Greenplum
- Microsoft SQL Server
- MySQL
- PostgreSQL
- YDB (федеративные запросы между базами YDB)

Архитектура системы обработки федеративных запросов

Внешние источники данных и внешние таблицы

Ключевым элементом системы обработки федеративных запросов YDB является понятие внешнего источника данных (external data source). В качестве таких источников могут выступать реляционные СУБД, объектные хранилища и другие системы хранения данных. При обработке федеративного запроса YDB потоково вычитывает данные из внешних систем и позволяет выполнять над ними точно такой же спектр операций, что и для локальных данных.

Для того, чтобы работать с данными, размещёнными во внешних системах, YDB должна располагать информацией о внутренней структуре этих данных (например, о количестве, названиях и типах столбцов в таблицах). Некоторые источники предоставляют подобную метаинформацию о данных вместе с самими данными, тогда как для работы с другими, несхематизированными источниками требуется задание этой метаинформации извне. Последней цели служат внешние таблицы (external tables).

Зарегистрировав в YDB внешние источники данных и (в случае необходимости) внешние таблицы, клиент может приступать к описанию федеративных запросов.

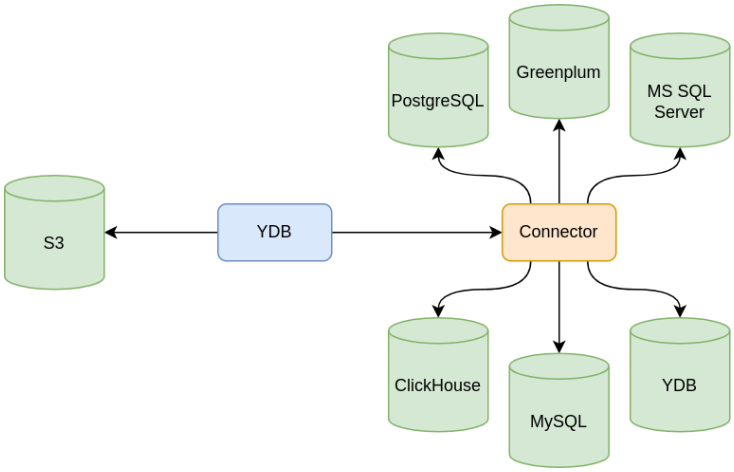
Коннекторы

В ходе выполнения федеративных запросов YDB необходимо обращаться по сети к сторонним системам хранения данных, для чего приходится использовать их клиентские библиотеки. Появление таких зависимостей негативно сказывается на объёме кодовой базы, времени компиляции и размере бинарных файлов YDB, а также на стабильности всего продукта в целом.

Перечень поддерживаемых источников данных для федеративных запросов постоянно расширяется.
Наиболее популярные источники, такие как S3, поддерживаются YDB нативно. Однако не всем пользователям требуется поддержка одновременно всех источников. Её можно включить опционально с помощью *коннекторов* - специальных микросервисов, реализующих унифицированный интерфейс доступа к внешним источникам данных.

В функции коннекторов входят:

- Трансляция YQL-запросов в запросы на языке, специфичном для внешнего источника (например, в запросы на другом диалекте SQL или в обращения к HTTP API).
- Организация сетевых соединений с источниками данных.
- Конвертация данных, извлечённых из внешних источников, в колоночное представление в формате Arrow IPC Stream, поддерживаемом YDB.



Таким образом, благодаря коннекторам формируется слой абстракции, скрывающий от YDB специфику внешних источников данных. Лаконичность интерфейса коннектора позволяет легко расширять перечень поддерживаемых источников, внося минимальные изменения в код YDB.

Пользователи могут развернуть один из готовых коннекторов или написать свою реализацию на любом языке программирования по gRPC спецификации.

Перечень поддерживаемых внешних источников данных

YDB имеет встроенную в `ydbd` поддержку следующих внешних источников данных:

- S3

Через коннектор fq-connector-go дополнительно поддерживается доступ к ряду экспериментальных внешних СУБД.
Подробности см. в разделе Федеративные запросы.

Работа с базами данных ClickHouse

> **Важно**
>
> Внешние коннекторы являются экспериментальной возможностью YDB. Для работы с внешними СУБД через коннекторы необходимо развернуть fq-connector-go и явно включить соответствующие источники в конфигурации кластера YDB. Функциональность может измениться, поэтому использование в производственной среде не рекомендуется без тщательного тестирования.

В этом разделе описана основная информация про работу с внешней базой данных ClickHouse.

Для работы с внешней базой данных ClickHouse необходимо выполнить следующие шаги:

1. Создать секрет, содержащий пароль для подключения к базе данных.

```
CREATE SECRET clickhouse_datasource_user_password WITH (value = "<password>");
```

2. Создать внешний источник данных, описывающий целевую базу данных внутри кластера ClickHouse. Для соединения с ClickHouse можно использовать либо нативный TCP-протокол (`PROTOCOL="NATIVE"`), либо протокол HTTP (`PROTOCOL="HTTP"`). Включить шифрование соединений к внешней базе данных можно с помощью параметра `USE_TLS="TRUE"` .

```
CREATE EXTERNAL DATA SOURCE clickhouse_datasource WITH (
    SOURCE_TYPE="ClickHouse",
    LOCATION="<host>:<port>",
    DATABASE_NAME="<database>",
    AUTH_METHOD="BASIC",
    LOGIN="<login>",
    PASSWORD_SECRET_PATH="clickhouse_datasource_user_password",
    PROTOCOL="NATIVE",
    USE_TLS="TRUE"
);
```

3. Развернуть коннектор и настроить динамические узлы YDB на взаимодействие с ним. Также необходимо обеспечить сетевой доступ с динамических узлов YDB к внешнему источнику данных (по адресу, указанному в параметре `LOCATION` запроса `CREATE EXTERNAL DATA SOURCE`). В случае, если на предыдущем шаге было включено шифрование сетевых соединений к внешнему источнику, коннектор будет использовать системные корневые сертификаты (более подробно о настройке TLS можно узнать в инструкции по разворачиванию коннектора).
4. Выполнить запрос к базе данных.

Синтаксис запросов

Для работы с ClickHouse используется следующая форма SQL-запроса:

```
SELECT * FROM clickhouse_datasource.<table_name>
```

где:

- `clickhouse_datasource` - идентификатор внешнего источника данных;
- `<table_name>` - имя таблицы внутри внешнего источника данных.

Ограничения

При работе с кластерами ClickHouse существует ряд ограничений:

1. Внешние источники доступны только для чтения данных через запросы `SELECT` . Запросы, модифицирующие таблицы во внешних источниках, движком обработки федеративных запросов в настоящее время не поддерживаются.
2. Если значение даты, хранящейся во внешнем источнике данных, находится вне допустимого для YDB диапазона (все используемые даты должны быть позднее 1970-01-01, но ранее 2105-12-31), в YDB такое значение будет преобразовано в `NULL` .
3. Система обработки федеративных запросов YDB умеет передавать исполнение некоторых частей запроса системе, выступающей в качестве источника данных. Фрагменты запроса передаются сквозь YDB непосредственно во внешнюю систему и обрабатываются внутри неё. С помощью этой оптимизации, которая носит название «пушдауна предикатов» (predicate pushdown), удаётся значительно снизить объём данных, передаваемых от источника к движку обработки федеративных запросов. Благодаря этому снижается нагрузка на сеть и экономятся вычислительные ресурсы YDB.

   Частный случай пушдауна предикатов, при котором выполняется передача фильтрующих выражений, указанных после ключевого слова `WHERE` , называется «пушдауном фильтров» (filter pushdown). Пушдаун фильтров возможен при использовании:

Описание	Пример
Проверка на `NULL`	`WHERE column1 IS NULL` или `WHERE column1 IS NOT NULL`
Логических условий `OR` , `NOT` , `AND` и круглых скобок для управление приоритетом вычислений.	`WHERE column1 IS NULL OR (column2 IS NOT NULL AND column3 > 10)` .
Операторов сравнения с другими колонками или константами.	`WHERE column1 > column2 OR column3 <= 10` , `WHERE column1 + column2 > 10` , `WHERE column1 = (10 + 10)`

   При использовании других видов фильтров пушдаун на источник не выполняется: фильтрация строк внешней таблицы будет выполнена на стороне федеративной YDB, что означает, что YDB выполнит полное чтение (full scan) внешней таблицы в момент обработки запроса.

   Поддерживаемые типы данных для пушдауна фильтров:

Тип данных YDB
`Bool`
`Int8`
`Uint8`
`Int16`
`Uint16`
`Int32`
`Uint32`
`Int64`
`Uint64`
`Float`
`Double`
`String`

Поддерживаемые типы данных

По умолчанию в ClickHouse колонки физически не могут содержать значение `NULL`, однако пользователь имеет возможность создать таблицу с колонками опциональных, или nullable типов. Типы колонок, отображаемые YDB при извлечении данных из внешней базы данных ClickHouse, будут зависеть от того, используются ли в таблице ClickHouse примитивные или опциональные типы. При этом в связи с рассмотренными выше ограничениями типов YDB, использующихся для хранения дат и времени, все аналогичные типы ClickHouse отображаются в YDB как опциональные.

Ниже приведены таблицы соответствия типов ClickHouse и YDB. Все остальные типы данных, за исключением перечисленных, не поддерживаются.

Примитивные типы данных

Тип данных ClickHouse	Тип данных YDB	Примечания
`Bool`	`Bool`	
`Int8`	`Int8`	
`UInt8`	`Uint8`	
`Int16`	`Int16`	
`UInt16`	`Uint16`	
`Int32`	`Int32`	
`UInt32`	`Uint32`	
`Int64`	`Int64`	
`UInt64`	`Uint64`	
`Float32`	`Float`	
`Float64`	`Double`	
`Date`	`Date`	
`Date32`	`Optional<Date>`	Допустимый диапазон дат с 1970-01-01 и до 2105-12-31. При выходе значения за границы диапазона возвращается `NULL`.
`DateTime`	`Optional<DateTime>`	Допустимый диапазон значений времени с 1970-01-01 00:00:00 и до 2105-12-31 23:59:59. При выходе значения за границы диапазона возвращается значение `NULL`.
`DateTime64`	`Optional<Timestamp>`	Допустимый диапазон значений времени с 1970-01-01 00:00:00 и до 2105-12-31 23:59:59. При выходе значения за границы диапазона возвращается значение `NULL`.
`String`	`String`	
`FixedString`	`String`	Нулевые байты `FixedString` переносятся в `String` без изменений.

Опциональные типы данных

Тип данных ClickHouse	Тип данных YDB	Примечания
`Nullable(Bool)`	`Optional<Bool>`	
`Nullable(Int8)`	`Optional<Int8>`	
`Nullable(UInt8)`	`Optional<Uint8>`	

`Nullable(Int16)`	`Optional<Int16>`	
`Nullable(UInt16)`	`Optional<Uint16>`	
`Nullable(Int32)`	`Optional<Int32>`	
`Nullable(UInt32)`	`Optional<Uint32>`	
`Nullable(Int64)`	`Optional<Int64>`	
`Nullable(UInt64)`	`Optional<Uint64>`	
`Nullable(Float32)`	`Optional<Float>`	
`Nullable(Float64)`	`Optional<Double>`	
`Nullable(Date)`	`Optional<Date>`	
`Nullable(Date32)`	`Optional<Date>`	Допустимый диапазон дат с 1970-01-01 и до 2105-12-31. При выходе значения за границы диапазона возвращается `NULL`.
`Nullable(DateTime)`	`Optional<DateTime>`	Допустимый диапазон значений времени с 1970-01-01 00:00:00 и до 2105-12-31 23:59:59. При выходе значения за границы диапазона возвращается значение `NULL`.
`Nullable(DateTime64)`	`Optional<Timestamp>`	Допустимый диапазон значений времени с 1970-01-01 00:00:00 и до 2105-12-31 23:59:59. При выходе значения за границы диапазона возвращается значение `NULL`.
`Nullable(String)`	`Optional<String>`	
`Nullable(FixedString)`	`Optional<String>`	Нулевые байты `FixedString` переносятся в `String` без изменений.

Работа с базами данных Greenplum

> **Важно**
>
> Внешние коннекторы являются экспериментальной возможностью YDB. Для работы с внешними СУБД через коннекторы необходимо развернуть fq-connector-go и явно включить соответствующие источники в конфигурации кластера YDB. Функциональность может измениться, поэтому использование в производственной среде не рекомендуется без тщательного тестирования.

В этом разделе описана основная информация про работу с внешней базой данных Greenplum. Поскольку Greenplum основан на PostgreSQL, интеграции с ними работают похожим образом, а некоторые ссылки ниже могут вести на документацию PostgreSQL.

Для работы с внешней базой данных Greenplum необходимо выполнить следующие шаги:

1. Создать секрет, содержащий пароль для подключения к базе данных.

   ```
   CREATE SECRET greenplum_datasource_user_password WITH (value = "<password>");
   ```

2. Создать внешний источник данных, описывающий определённую базу данных в составе кластера Greenplum. В параметр `LOCATION` нужно передать сетевой адрес мастер-ноды Greenplum. При чтении по умолчанию используется пространство имен `public`, но это значение можно изменить с помощью опционального параметра `SCHEMA`. Включить шифрование соединений к внешней базе данных можно с помощью параметра `USE_TLS="TRUE"`.

   ```
   CREATE EXTERNAL DATA SOURCE greenplum_datasource WITH (
       SOURCE_TYPE="Greenplum",
       LOCATION="<host>:<port>",
       DATABASE_NAME="<database>",
       AUTH_METHOD="BASIC",
       LOGIN="user",
       PASSWORD_SECRET_PATH="greenplum_datasource_user_password",
       USE_TLS="TRUE",
       SCHEMA="<schema>"
   );
   ```

3. Развернуть коннектор и настроить динамические узлы YDB на взаимодействие с ним. Также необходимо обеспечить сетевой доступ с динамических узлов YDB к внешнему источнику данных (по адресу, указанному в параметре `LOCATION` запроса `CREATE EXTERNAL DATA SOURCE`). В случае, если на предыдущем шаге было включено шифрование сетевых соединений к внешнему источнику, коннектор будет использовать системные корневые сертификаты (более подробно о настройке TLS можно узнать в инструкции по разворачиванию коннектора).
4. Выполнить запрос к базе данных.

Синтаксис запросов

Для работы с Greenplum используется следующая форма SQL-запроса:

```
SELECT * FROM greenplum_datasource.<table_name>
```

где:

- `greenplum_datasource` - идентификатор внешнего источника данных;
- `<table_name>` - имя таблицы внутри внешнего источника данных.

Ограничения

При работе с кластерами Greenplum существует ряд ограничений:

1. Внешние источники доступны только для чтения данных через запросы `SELECT`. Запросы, модифицирующие таблицы во внешних источниках, движком обработки федеративных запросов в настоящее время не поддерживаются.
2. Если значение даты, хранящейся во внешнем источнике данных, находится вне допустимого для YDB диапазона (все используемые даты должны быть позднее 1970-01-01, но ранее 2105-12-31), в YDB такое значение будет преобразовано в `NULL`.
3. Система обработки федеративных запросов YDB умеет передавать исполнение некоторых частей запроса системе, выступающей в качестве источника данных. Фрагменты запроса передаются сквозь YDB непосредственно во внешнюю систему и обрабатываются внутри неё. С помощью этой оптимизации, которая носит название «пушдауна предикатов» (predicate pushdown), удаётся значительно снизить объём данных, передаваемых от источника к движку обработки федеративных запросов. Благодаря этому снижается нагрузка на сеть и экономятся вычислительные ресурсы YDB.

   Частный случай пушдауна предикатов, при котором выполняется передача фильтрующих выражений, указанных после ключевого слова `WHERE`, называется «пушдауном фильтров» (filter pushdown). Пушдаун фильтров возможен при использовании:

Описание	Пример
Проверка на `NULL`	`WHERE column1 IS NULL` или `WHERE column1 IS NOT NULL`
Логических условий `OR`, `NOT`, `AND` и круглых скобок для управление приоритетом вычислений.	`WHERE column1 IS NULL OR (column2 IS NOT NULL AND column3 > 10)`.
Операторов сравнения с другими колонками или константами.	`WHERE column1 > column2 OR column3 <= 10`, `WHERE column1 + column2 > 10`, `WHERE column1 = (10 + 10)`

   При использовании других видов фильтров пушдаун на источник не выполняется: фильтрация строк внешней таблицы будет выполнена на стороне федеративной YDB, что означает, что YDB выполнит полное чтение (full scan) внешней таблицы в момент обработки запроса.

Поддерживаемые типы данных для пушдауна фильтров:

Тип данных YDB
`Bool`
`Int8`
`Int16`
`Int32`
`Int64`
`Float`
`Double`

Поддерживаемые типы данных

В базе данных Greenplum признак опциональности значений колонки (разрешено или запрещено колонке содержать значения `NULL`) не является частью системы типов данных. Ограничение (constraint) `NOT NULL` для каждой колонки реализуется в виде атрибута `attnotnull` в системном каталоге pg_attribute, то есть на уровне метаданных таблицы. Следовательно, все базовые типы Greenplum по умолчанию могут содержать значения `NULL`, и в системе типов YDB они должны отображаться в опциональные типы.

Ниже приведена таблица соответствия типов Greenplum и YDB. Все остальные типы данных, за исключением перечисленных, не поддерживаются.

Тип данных Greenplum	Тип данных YDB	Примечания
`boolean`	`Optional<Bool>`	
`smallint`	`Optional<Int16>`	
`int2`	`Optional<Int16>`	
`integer`	`Optional<Int32>`	
`int`	`Optional<Int32>`	
`int4`	`Optional<Int32>`	
`serial`	`Optional<Int32>`	
`serial4`	`Optional<Int32>`	
`bigint`	`Optional<Int64>`	
`int8`	`Optional<Int64>`	
`bigserial`	`Optional<Int64>`	
`serial8`	`Optional<Int64>`	
`real`	`Optional<Float>`	
`float4`	`Optional<Float>`	
`double precision`	`Optional<Double>`	
`float8`	`Optional<Double>`	
`json`	`Optional<Json>`	
`date`	`Optional<Date>`	Допустимый диапазон дат с 1970-01-01 и до 2105-12-31. При выходе значения за границы диапазона возвращается `NULL`.
`timestamp`	`Optional<Timestamp>`	Допустимый диапазон времени с 1970-01-01 00:00:00 и до 2105-12-31 23:59:59. При выходе значения за границы диапазона возвращается значение `NULL`.
`bytea`	`Optional<String>`	
`character`	`Optional<Utf8>`	Правила сортировки по умолчанию, строка дополняется пробелами до требуемой длины.
`character varying`	`Optional<Utf8>`	Правила сортировки по умолчанию.
`text`	`Optional<Utf8>`	Правила сортировки по умолчанию.

Работа с базами данных Microsoft SQL Server

Важно

Внешние коннекторы являются экспериментальной возможностью YDB. Для работы с внешними СУБД через коннекторы необходимо развернуть fq-connector-go и явно включить соответствующие источники в конфигурации кластера YDB. Функциональность может измениться, поэтому использование в производственной среде не рекомендуется без тщательного тестирования.

В этом разделе описана основная информация про работу с внешней базой данных Microsoft SQL Server.

Для работы с внешней базой данных Microsoft SQL Server необходимо выполнить следующие шаги:

1. Создать секрет, содержащий пароль для подключения к базе данных.

```
CREATE SECRET ms_sql_server_datasource_user_password WITH (value = "<password>");
```

2. Создать внешний источник данных, описывающий определённую базу данных Microsoft SQL Server. Параметр `LOCATION` содержит сетевой адрес экземпляра Microsoft SQL Server, к которому осуществляется подключение. В `DATABASE_NAME` указывается имя базы данных (например, `master`). Для аутентификации во внешнюю базу используются значения параметров `LOGIN` и `PASSWORD_SECRET_PATH`. Включить шифрование соединений к внешней базе данных можно с помощью параметра `USE_TLS="TRUE"`.

```
CREATE EXTERNAL DATA SOURCE ms_sql_server_datasource WITH (
    SOURCE_TYPE="MsSQLServer",
    LOCATION="<host>:<port>",
    DATABASE_NAME="<database>",
    AUTH_METHOD="BASIC",
    LOGIN="user",
    PASSWORD_SECRET_PATH="ms_sql_server_datasource_user_password",
    USE_TLS="TRUE"
);
```

3. Развернуть коннектор и настроить динамические узлы YDB на взаимодействие с ним. Также необходимо обеспечить сетевой доступ с динамических узлов YDB к внешнему источнику данных (по адресу, указанному в параметре `LOCATION` запроса `CREATE EXTERNAL DATA SOURCE`). В случае, если на предыдущем шаге было включено шифрование сетевых соединений к внешнему источнику, коннектор будет использовать системные корневые сертификаты (более подробно о настройке TLS можно узнать в инструкции по разворачиванию коннектора).
4. Выполнить запрос к базе данных.

Синтаксис запросов

Для работы с Microsoft SQL Server используется следующая форма SQL-запроса:

```
SELECT * FROM ms_sql_server_datasource.<table_name>
```

где:

- `ms_sql_server_datasource` - идентификатор внешнего источника данных;
- `<table_name>` - имя таблицы внутри внешнего источника данных.

Ограничения

При работе с кластерами Microsoft SQL Server существует ряд ограничений:

1. Внешние источники доступны только для чтения данных через запросы `SELECT`. Запросы, модифицирующие таблицы во внешних источниках, движком обработки федеративных запросов в настоящее время не поддерживаются.
2. Если значение даты, хранящейся во внешнем источнике данных, находится вне допустимого для YDB диапазона (все используемые даты должны быть позднее 1970-01-01, но ранее 2105-12-31), в YDB такое значение будет преобразовано в `NULL`.
3. Система обработки федеративных запросов YDB умеет передавать исполнение некоторых частей запроса системе, выступающей в качестве источника данных. Фрагменты запроса передаются сквозь YDB непосредственно во внешнюю систему и обрабатываются внутри неё. С помощью этой оптимизации, которая носит название «пушдауна предикатов» (predicate pushdown), удаётся значительно снизить объём данных, передаваемых от источника к движку обработки федеративных запросов. Благодаря этому снижается нагрузка на сеть и экономятся вычислительные ресурсы YDB.

   Частный случай пушдауна предикатов, при котором выполняется передача фильтрующих выражений, указанных после ключевого слова `WHERE`, называется «пушдауном фильтров» (filter pushdown). Пушдаун фильтров возможен при использовании:

Описание	Пример
Проверка на `NULL`	`WHERE column1 IS NULL` или `WHERE column1 IS NOT NULL`
Логических условий `OR`, `NOT`, `AND` и круглых скобок для управление приоритетом вычислений.	`WHERE column1 IS NULL OR (column2 IS NOT NULL AND column3 > 10)`.
Операторов сравнения с другими колонками или константами.	`WHERE column1 > column2 OR column3 <= 10`, `WHERE column1 + column2 > 10`, `WHERE column1 = (10 + 10)`

   При использовании других видов фильтров пушдаун на источник не выполняется: фильтрация строк внешней таблицы будет выполнена на стороне федеративной YDB, что означает, что YDB выполнит полное чтение (full scan) внешней таблицы в момент обработки запроса.

Тип данных YDB
`Bool`
`Int8`
`Int16`
`Int32`
`Int64`
`Float`
`Double`

Поддерживаемые типы данных

В базе данных Microsoft SQL Server признак опциональности значений колонки (разрешено или запрещено колонке содержать значения `NULL`) не является частью системы типов данных. Ограничение (constraint) `NOT NULL` для любой колонки любой таблицы хранится в виде значения столбца `IS_NULLABLE` системной таблицы INFORMATION_SCHEMA.COLUMNS, то есть на уровне метаданных таблицы. Следовательно, все базовые типы Microsoft SQL Server по умолчанию могут содержать значения `NULL`, и в системе типов YDB они должны отображаться в опциональные типы.

Ниже приведена таблица соответствия типов Microsoft SQL Server и YDB. Все остальные типы данных, за исключением перечисленных, не поддерживаются.

Тип данных Microsoft SQL Server	Тип данных YDB	Примечания
`bit`	`Optional<Bool>`	
`tinyint`	`Optional<Int8>`	
`smallint`	`Optional<Int16>`	
`int`	`Optional<Int32>`	
`bigint`	`Optional<Int64>`	
`real`	`Optional<Float>`	
`float`	`Optional<Double>`	
`date`	`Optional<Date>`	Допустимый диапазон дат с 1970-01-01 и до 2105-12-31. При выходе значения за границы диапазона возвращается `NULL`.
`smalldatetime`	`Optional<Datetime>`	Допустимый диапазон времени с 1970-01-01 00:00:00 и до 2105-12-31 23:59:59. При выходе значения за границы диапазона возвращается значение `NULL`.
`datetime`	`Optional<Timestamp>`	Допустимый диапазон времени с 1970-01-01 00:00:00 и до 2105-12-31 23:59:59. При выходе значения за границы диапазона возвращается значение `NULL`.
`datetime2`	`Optional<Timestamp>`	Допустимый диапазон времени с 1970-01-01 00:00:00 и до 2105-12-31 23:59:59. При выходе значения за границы диапазона возвращается значение `NULL`.
`binary`	`Optional<String>`	
`varbinary`	`Optional<String>`	
`image`	`Optional<String>`	
`char`	`Optional<Utf8>`	
`varchar`	`Optional<Utf8>`	
`text`	`Optional<Utf8>`	
`nchar`	`Optional<Utf8>`	
`nvarchar`	`Optional<Utf8>`	
`ntext`	`Optional<Utf8>`	

Работа с базами данных MySQL

> **Важно**
>
> Внешние коннекторы являются экспериментальной возможностью YDB. Для работы с внешними СУБД через коннекторы необходимо развернуть fq-connector-go и явно включить соответствующие источники в конфигурации кластера YDB. Функциональность может измениться, поэтому использование в производственной среде не рекомендуется без тщательного тестирования.

В этом разделе описана основная информация про работу с внешней базой данных MySQL.

Для работы с внешней базой данных MySQL необходимо выполнить следующие шаги:

1. Создать секрет, содержащий пароль для подключения к базе данных.

```
CREATE SECRET mysql_datasource_user_password WITH (value = "<password>");
```

2. Создать внешний источник данных, описывающий определённую базу данных MySQL. Параметр `LOCATION` содержит сетевой адрес экземпляра MySQL, к которому осуществляется подключение. В `DATABASE_NAME` указывается имя базы данных (например, `mysql`). Для аутентификации во внешнюю базу используются значения параметров `LOGIN` и `PASSWORD_SECRET_PATH`. Включить шифрование соединений к внешней базе данных можно с помощью параметра `USE_TLS="TRUE"`.

```
CREATE EXTERNAL DATA SOURCE mysql_datasource WITH (
    SOURCE_TYPE="MySQL",
    LOCATION="<host>:<port>",
    DATABASE_NAME="<database>",
    AUTH_METHOD="BASIC",
    LOGIN="user",
    PASSWORD_SECRET_PATH="mysql_datasource_user_password",
    USE_TLS="TRUE"
);
```

3. Развернуть коннектор и настроить динамические узлы YDB на взаимодействие с ним. Также необходимо обеспечить сетевой доступ с динамических узлов YDB к внешнему источнику данных (по адресу, указанному в параметре `LOCATION` запроса `CREATE EXTERNAL DATA SOURCE`). В случае, если на предыдущем шаге было включено шифрование сетевых соединений к внешнему источнику, коннектор будет использовать системные корневые сертификаты (более подробно о настройке TLS можно узнать в инструкции по разворачиванию коннектора).
4. Выполнить запрос к базе данных.

Синтаксис запросов

Для работы с MySQL используется следующая форма SQL-запроса:

```
SELECT * FROM mysql_datasource.<table_name>
```

где:

- `mysql_datasource` - идентификатор внешнего источника данных;
- `<table_name>` - имя таблицы внутри внешнего источника данных.

Ограничения

При работе с кластерами MySQL существует ряд ограничений:

1. Внешние источники доступны только для чтения данных через запросы `SELECT`. Запросы, модифицирующие таблицы во внешних источниках, движком обработки федеративных запросов в настоящее время не поддерживаются.
2. Если значение даты, хранящейся во внешнем источнике данных, находится вне допустимого для YDB диапазона (все используемые даты должны быть позднее 1970-01-01, но ранее 2105-12-31), в YDB такое значение будет преобразовано в `NULL`.
3. Система обработки федеративных запросов YDB умеет передавать исполнение некоторых частей запроса системе, выступающей в качестве источника данных. Фрагменты запроса передаются сквозь YDB непосредственно во внешнюю систему и обрабатываются внутри неё. С помощью этой оптимизации, которая носит название «пушдауна предикатов» (predicate pushdown), удаётся значительно снизить объём данных, передаваемых от источника к движку обработки федеративных запросов. Благодаря этому снижается нагрузка на сеть и экономятся вычислительные ресурсы YDB.

   Частный случай пушдауна предикатов, при котором выполняется передача фильтрующих выражений, указанных после ключевого слова `WHERE`, называется «пушдауном фильтров» (filter pushdown). Пушдаун фильтров возможен при использовании:

Описание	Пример
Проверка на `NULL`	`WHERE column1 IS NULL` или `WHERE column1 IS NOT NULL`
Логических условий `OR`, `NOT`, `AND` и круглых скобок для управление приоритетом вычислений.	`WHERE column1 IS NULL OR (column2 IS NOT NULL AND column3 > 10)`.
Операторов сравнения с другими колонками или константами.	`WHERE column1 > column2 OR column3 <= 10`, `WHERE column1 + column2 > 10`, `WHERE column1 = (10 + 10)`

   При использовании других видов фильтров пушдаун на источник не выполняется: фильтрация строк внешней таблицы будет выполнена на стороне федеративной YDB, что означает, что YDB выполнит полное чтение (full scan) внешней таблицы в момент обработки запроса.

   Поддерживаемые типы данных для пушдауна фильтров:

Тип данных YDB
`Bool`
`Int8`
`Uint8`
`Int16`
`Uint16`
`Int32`
`Uint32`
`Int64`
`Uint64`
`Float`
`Double`

Поддерживаемые типы данных

В базе данных MySQL признак опциональности значений колонки (разрешено или запрещено колонке содержать значения `NULL`) не является частью системы типов данных. Ограничение (constraint) `NOT NULL` для любой колонки любой таблицы хранится в виде значения столбца `IS_NULLABLE` системной таблицы INFORMATION_SCHEMA.COLUMNS, то есть на уровне метаданных таблицы. Следовательно, все базовые типы MySQL по умолчанию могут содержать значения `NULL`, и в системе типов YDB они должны отображаться в опциональные типы.

Ниже приведена таблица соответствия типов MySQL и YDB. Все остальные типы данных, за исключением перечисленных, не поддерживаются.

Тип данных MySQL	Тип данных YDB	Примечания
`bool`	`Optional<Bool>`	
`tinyint`	`Optional<Int8>`	
`tinyint unsigned`	`Optional<Uint8>`	
`smallint`	`Optional<Int16>`	
`smallint unsigned`	`Optional<Uint16>`	
`mediumint`	`Optional<Int32>`	
`mediumint unsigned`	`Optional<Uint32>`	
`int`	`Optional<Int32>`	
`int unsigned`	`Optional<Uint32>`	
`bigint`	`Optional<Int64>`	
`bigint unsigned`	`Optional<Uint64>`	
`float`	`Optional<Float>`	
`real`	`Optional<Float>`	
`double`	`Optional<Double>`	
`date`	`Optional<Date>`	Допустимый диапазон дат с 1970-01-01 и до 2105-12-31. При выходе значения за границы диапазона возвращается `NULL`.
`datetime`	`Optional<Timestamp>`	Допустимый диапазон времени с 1970-01-01 00:00:00 и до 2105-12-31 23:59:59. При выходе значения за границы диапазона возвращается значение `NULL`.
`timestamp`	`Optional<Timestamp>`	Допустимый диапазон времени с 1970-01-01 00:00:00 и до 2105-12-31 23:59:59. При выходе значения за границы диапазона возвращается значение `NULL`.
`tinyblob`	`Optional<String>`	
`blob`	`Optional<String>`	
`mediumblob`	`Optional<String>`	
`longblob`	`Optional<String>`	
`tinytext`	`Optional<String>`	
`text`	`Optional<String>`	
`mediumtext`	`Optional<String>`	

`longtext`	`Optional<String>`	
`char`	`Optional<Utf8>`	
`varchar`	`Optional<Utf8>`	
`binary`	`Optional<String>`	
`varbinary`	`Optional<String>`	
`json`	`Optional<Json>`	

Работа с базами данных PostgreSQL

> **Важно**
>
> Внешние коннекторы являются экспериментальной возможностью YDB. Для работы с внешними СУБД через коннекторы необходимо развернуть fq-connector-go и явно включить соответствующие источники в конфигурации кластера YDB. Функциональность может измениться, поэтому использование в производственной среде не рекомендуется без тщательного тестирования.

В этом разделе описана основная информация про работу с внешней базой данных PostgreSQL.

Для работы с внешней базой данных PostgreSQL необходимо выполнить следующие шаги:

1. Создать секрет, содержащий пароль для подключения к базе данных.

```
CREATE SECRET postgresql_datasource_user_password WITH (value = "<password>");
```

2. Создать внешний источник данных, описывающий определённую базу данных в составе кластера PostgreSQL. При чтении по умолчанию используется пространство имен `public`, но это значение можно изменить с помощью опционального параметра `SCHEMA`. Сетевое подключение выполняется по стандартному (Frontend/Backend Protocol) по транспорту TCP (`PROTOCOL="NATIVE"`). Включить шифрование соединений к внешней базе данных можно с помощью параметра `USE_TLS="TRUE"`.

```
CREATE EXTERNAL DATA SOURCE postgresql_datasource WITH (
    SOURCE_TYPE="PostgreSQL",
    LOCATION="<host>:<port>",
    DATABASE_NAME="<database>",
    AUTH_METHOD="BASIC",
    LOGIN="user",
    PASSWORD_SECRET_PATH="postgresql_datasource_user_password",
    PROTOCOL="NATIVE",
    USE_TLS="TRUE",
    SCHEMA="<schema>"
);
```

3. Развернуть коннектор и настроить динамические узлы YDB на взаимодействие с ним. Также необходимо обеспечить сетевой доступ с динамических узлов YDB к внешнему источнику данных (по адресу, указанному в параметре `LOCATION` запроса `CREATE EXTERNAL DATA SOURCE`). В случае, если на предыдущем шаге было включено шифрование сетевых соединений к внешнему источнику, коннектор будет использовать системные корневые сертификаты (более подробно о настройке TLS можно узнать в инструкции по разворачиванию коннектора).
4. Выполнить запрос к базе данных.

Синтаксис запросов

Для работы с PostgreSQL используется следующая форма SQL-запроса:

```
SELECT * FROM postgresql_datasource.<table_name>
```

где:

- `postgresql_datasource` - идентификатор внешнего источника данных;
- `<table_name>` - имя таблицы внутри внешнего источника данных.

Ограничения

При работе с кластерами PostgreSQL существует ряд ограничений:

1. Внешние источники доступны только для чтения данных через запросы `SELECT`. Запросы, модифицирующие таблицы во внешних источниках, движком обработки федеративных запросов в настоящее время не поддерживаются.
2. Если значение даты, хранящейся во внешнем источнике данных, находится вне допустимого для YDB диапазона (все используемые даты должны быть позднее 1970-01-01, но ранее 2105-12-31), в YDB такое значение будет преобразовано в `NULL`.
3. Система обработки федеративных запросов YDB умеет передавать исполнение некоторых частей запроса системе, выступающей в качестве источника данных. Фрагменты запроса передаются сквозь YDB непосредственно во внешнюю систему и обрабатываются внутри неё. С помощью этой оптимизации, которая носит название «пушдауна предикатов» (predicate pushdown), удаётся значительно снизить объём данных, передаваемых от источника к движку обработки федеративных запросов. Благодаря этому снижается нагрузка на сеть и экономятся вычислительные ресурсы YDB.

   Частный случай пушдауна предикатов, при котором выполняется передача фильтрующих выражений, указанных после ключевого слова `WHERE`, называется «пушдауном фильтров» (filter pushdown). Пушдаун фильтров возможен при использовании:

Описание	Пример
Проверка на `NULL`	`WHERE column1 IS NULL` или `WHERE column1 IS NOT NULL`
Логических условий `OR`, `NOT`, `AND` и круглых скобок для управление приоритетом вычислений.	`WHERE column1 IS NULL OR (column2 IS NOT NULL AND column3 > 10)`.
Операторов сравнения с другими колонками или константами.	`WHERE column1 > column2 OR column3 <= 10`, `WHERE column1 + column2 > 10`, `WHERE column1 = (10 + 10)`

   При использовании других видов фильтров пушдаун на источник не выполняется: фильтрация строк внешней таблицы будет выполнена на стороне федеративной YDB, что означает, что YDB выполнит полное чтение (full scan) внешней таблицы в момент обработки запроса.

Поддерживаемые типы данных для пушдауна фильтров:

Тип данных YDB
`Bool`
`Int8`
`Int16`
`Int32`
`Int64`
`Float`
`Double`
`Decimal`

Поддерживаемые типы данных

В базе данных PostgreSQL признак опциональности значений колонки (разрешено или запрещено колонке содержать значения `NULL`) не является частью системы типов данных. Ограничение (constraint) `NOT NULL` для каждой колонки реализуется в виде атрибута `attnotnull` в системном каталоге pg_attribute, то есть на уровне метаданных таблицы. Следовательно, все базовые типы PostgreSQL по умолчанию могут содержать значения `NULL`, и в системе типов YDB они должны отображаться в опциональные типы.

Ниже приведена таблица соответствия типов PostgreSQL и YDB. Все остальные типы данных, за исключением перечисленных, не поддерживаются.

Тип данных PostgreSQL	Тип данных YDB	Примечания
`boolean`	`Optional<Bool>`	
`smallint`	`Optional<Int16>`	
`int2`	`Optional<Int16>`	
`integer`	`Optional<Int32>`	
`int`	`Optional<Int32>`	
`int4`	`Optional<Int32>`	
`serial`	`Optional<Int32>`	
`serial4`	`Optional<Int32>`	
`bigint`	`Optional<Int64>`	
`int8`	`Optional<Int64>`	
`bigserial`	`Optional<Int64>`	
`serial8`	`Optional<Int64>`	
`real`	`Optional<Float>`	
`float4`	`Optional<Float>`	
`double precision`	`Optional<Double>`	
`float8`	`Optional<Double>`	
`date`	`Optional<Date>`	Допустимый диапазон дат с 1970-01-01 и до 2105-12-31. При выходе значения за границы диапазона возвращается `NULL`.
`timestamp`	`Optional<Timestamp>`	Допустимый диапазон времени с 1970-01-01 00:00:00 и до 2105-12-31 23:59:59. При выходе значения за границы диапазона возвращается значение `NULL`.
`bytea`	`Optional<String>`	
`character`	`Optional<Utf8>`	Правила сортировки по умолчанию, строка дополняется пробелами до требуемой длины.
`character varying`	`Optional<Utf8>`	Правила сортировки по умолчанию.
`text`	`Optional<Utf8>`	Правила сортировки по умолчанию.
`json`	`Optional<Json>`	
`numeric(p,s)`	`Optional<Decimal(p,s)>`	`p` (precision) - общее количество знаков в числе, `s` (scale) - количество знаков после запятой. Типы `numeric` без указания параметров (так называемые «неограниченные», unconstrained) преобразуются в `Optional<Decimal(35, 0)>`. Типы `numeric`, у которых `p > 35` или `s < 0`, не поддерживаются.

Работа с базами данных YDB

> **Важно**
>
> Внешние коннекторы являются экспериментальной возможностью YDB. Для работы с внешними СУБД через коннекторы необходимо развернуть fq-connector-go и явно включить соответствующие источники в конфигурации кластера YDB. Функциональность может измениться, поэтому использование в производственной среде не рекомендуется без тщательного тестирования.

YDB может выступать в качестве внешнего источника данных для другой базы YDB. В данном разделе рассматривается организация совместной работы двух независимых баз данных YDB в режиме обработки федеративных запросов.

Для подключения к внешней базе YDB со стороны другой базы YDB, выступающей в роли движка обработки федеративных запросов, на последней требуется выполнить следующие шаги:

1. Подготовить аутентификационные данные для доступа к удалённой базе YDB. В настоящее время в федеративных запросах к YDB доступен метод аутентификации по логину и паролю (остальные методы не поддерживаются). Пароль к внешней базе сохраняется в виде секрета:

   ```
   CREATE SECRET ydb_datasource_user_password WITH (value = "<password>");
   ```

2. Создать внешний источник данных, описывающий стороннюю базу YDB. Параметр `LOCATION` содержит сетевой адрес экземпляра YDB, к которому осуществляется сетевое подключение. В `DATABASE_NAME` указывается имя базы данных (например, `local`). Для аутентификации во внешнюю базу используются значения параметров `LOGIN` и `PASSWORD_SECRET_PATH`. Включить шифрование соединений к внешней базе данных можно с помощью параметра `USE_TLS="TRUE"`. Если шифрование включено, то в поле `<port>` параметра `LOCATION` необходимо указать порт gRPCs внешней YDB, в противном случае - порт gRPC.

   ```
   CREATE EXTERNAL DATA SOURCE ydb_datasource WITH (
       SOURCE_TYPE="Ydb",
       LOCATION="<host>:<port>",
       DATABASE_NAME="<database>",
       AUTH_METHOD="BASIC",
       LOGIN="user",
       PASSWORD_SECRET_PATH="ydb_datasource_user_password",
       USE_TLS="TRUE"
   );
   ```

3. Развернуть коннектор и настроить динамические узлы YDB на взаимодействие с ним. Также необходимо обеспечить сетевой доступ с динамических узлов YDB к внешнему источнику данных (по адресу, указанному в параметре `LOCATION` запроса `CREATE EXTERNAL DATA SOURCE`). В случае, если на предыдущем шаге было включено шифрование сетевых соединений к внешнему источнику, коннектор будет использовать системные корневые сертификаты (более подробно о настройке TLS можно узнать в инструкции по разворачиванию коннектора).
4. Выполнить запрос к внешнему источнику данных.

Синтаксис запросов

Для извлечения данных из таблиц внешней базы YDB используется следующая форма SQL-запроса:

```
SELECT * FROM ydb_datasource.`<table_name>`
```

где:

- `ydb_datasource` - идентификатор внешнего источника данных;
- `<table_name>` - полное имя таблицы внутри иерархии каталогов в базе данных YDB, например, `table`, `dir1/table1` или `dir1/dir2/table3`.

В случае, если таблица находится на верхнем уровне иерархии (не принадлежит ни одному из каталогов), допускается не заключать имя таблицы в обратные апострофы "`":

```
SELECT * FROM ydb_datasource.<table_name>
```

Ограничения

При работе с внешними источниками данных YDB существует ряд ограничений:

1. Внешние источники доступны только для чтения данных через запросы `SELECT`. Запросы, модифицирующие таблицы во внешних источниках, движком обработки федеративных запросов в настоящее время не поддерживаются.
2. Система обработки федеративных запросов YDB умеет передавать исполнение некоторых частей запроса системе, выступающей в качестве источника данных. Фрагменты запроса передаются сквозь YDB непосредственно во внешнюю систему и обрабатываются внутри неё. С помощью этой оптимизации, которая носит название «пушдауна предикатов» (predicate pushdown), удаётся значительно снизить объём данных, передаваемых от источника к движку обработки федеративных запросов. Благодаря этому снижается нагрузка на сеть и экономятся вычислительные ресурсы YDB.

   Частный случай пушдауна предикатов, при котором выполняется передача фильтрующих выражений, указанных после ключевого слова `WHERE`, называется «пушдауном фильтров» (filter pushdown). Пушдаун фильтров возможен при использовании:

Описание	Пример	Ограничение
Фильтров вида `IS NULL` / `IS NOT NULL`	`WHERE column1 IS NULL` или `WHERE column1 IS NOT NULL`	

Логических условий `OR`, `NOT`, `AND` и круглых скобок для управление приоритетом вычислений.	`WHERE column1 IS NULL OR (column2 IS NOT NULL AND column3 > 10)`.	
Операторов сравнения с другими колонками или константами.	`WHERE column1 > column2 OR column3 <= 10`.	
Оператора сопоставления строк с образцом `LIKE`.	`WHERE column1 LIKE '_abc%'`	В настоящее время поддерживается пушдаун только простых паттернов, основанных на префиксах (`'abc_'`, `'abc%'`), суффиксах (`'_abc'`, `'%abc'`) или поиске подстроки в строке (`'_abc_'`, `'%abc%'`, `'_abc%'`, `'%abc_'`). При необходимости пушдауна более сложных паттернов рекомендуется воспользуется `REGEXP`.
Оператора сопоставления строк с образцом `REGEXP`.	`WHERE column1 REGEXP '.*abc.*'`	

При использовании других видов фильтров пушдаун на источник не выполняется: фильтрация строк внешней таблицы будет выполнена на стороне федеративной YDB, что означает, что YDB выполнит полное чтение (full scan) внешней таблицы в момент обработки запроса.

Поддерживаемые типы данных для пушдауна фильтров:

Тип данных YDB
`Bool`
`Int8`
`Uint8`
`Int16`
`Uint16`
`Int32`
`Uint32`
`Int64`
`Uint64`
`Float`
`Double`
`String`
`Utf8`

Поддерживаемые типы данных

При работе с таблицами, размещёнными во внешней базе YDB, пользователям доступен ограниченный набор типов данных. Все остальные типы, за исключением перечисленных ниже, не поддерживаются. В некоторых случаях производится преобразование типов, то есть колонки таблицы из внешней базы YDB меняют свой тип после вычитывания этой таблицы базой YDB, обрабатывающей федеративный запрос.

Тип данных внешнего источника YDB	Тип данных в федеративной YDB
`Bool`	`Bool`
`Int8`	`Int8`
`Int16`	`Int16`
`Int32`	`Int32`
`Int64`	`Int64`
`Uint8`	`Uint8`

`Uint16`	`Uint16`
`Uint32`	`Uint32`
`Uint64`	`Uint64`
`Float`	`Float`
`Double`	`Double`
`String`	`String`
`Utf8`	`Utf8`
`Date`	`Date`
`Datetime`	`Datetime`
`Timestamp`	`Timestamp`
`Json`	`Json`
`JsonDocument`	`Json`

Чтение из бакетов S3 через внешние источники данных

Перед началом работы с S3 необходимо настроить подключение к хранилищу данных. Для этого существует DDL для настройки таких подключений. Далее рассмотрим SQL синтаксис и управление этими настройками.

Бакеты в S3 бывают двух видов: публичные и приватные. Для подключения к публичному бакету необходимо использовать `AUTH_METHOD="NONE"`, а для подключения к приватному — `AUTH_METHOD="AWS"`. Подробное описание `AWS` можно найти здесь. `AUTH_METHOD="NONE"` означает, что аутентификация не требуется. В случае `AUTH_METHOD="AWS"` необходимо указать несколько дополнительных параметров:

- `AWS_ACCESS_KEY_ID_SECRET_PATH` — секрет, в котором хранится `AWS_ACCESS_KEY_ID`.
- `AWS_SECRET_ACCESS_KEY_SECRET_PATH` — секрет, в котором хранится `AWS_SECRET_ACCESS_KEY`.
- `AWS_REGION` — регион, из которого будет происходить чтение, например `ru-central-1`.

Для настройки соединения с публичным бакетом достаточно выполнить следующий SQL-запрос. Запрос создаст внешний источник данных с именем `s3_data_source`, который будет указывать на конкретный S3-бакет с именем `bucket`.

```
CREATE EXTERNAL DATA SOURCE s3_data_source WITH (
  SOURCE_TYPE="ObjectStorage",
  LOCATION="https://s3_storage_domain/bucket/",
  AUTH_METHOD="NONE"
);
```

Для настройки соединения с приватным бакетом необходимо выполнить несколько SQL-запросов. Сначала нужно создать секреты, содержащие `AWS_ACCESS_KEY_ID` и `AWS_SECRET_ACCESS_KEY`.

```
CREATE SECRET aws_access_id WITH (value=<id>);
CREATE SECRET aws_access_key WITH (value=<key>);
```

Следующим шагом создаётся внешний источник данных с именем `s3_data_source`, который будет указывать на конкретный S3-бакет с именем `bucket`, а также использовать `AUTH_METHOD="AWS"`, для которого задаются параметры `AWS_ACCESS_KEY_ID_SECRET_PATH`, `AWS_SECRET_ACCESS_KEY_SECRET_PATH`, `AWS_REGION`. Значения этих параметров описаны выше.

```
CREATE EXTERNAL DATA SOURCE s3_data_source WITH (
  SOURCE_TYPE="ObjectStorage",
  LOCATION="https://s3_storage_domain/bucket/",
  AUTH_METHOD="AWS",
  AWS_ACCESS_KEY_ID_SECRET_PATH="aws_access_id",
  AWS_SECRET_ACCESS_KEY_SECRET_PATH="aws_access_key",
  AWS_REGION="ru-central-1"
);
```

Использование внешнего источника данных для S3-бакета

При работе с S3-совместимым хранилищем данных с помощью внешних источников данных можно выяснить свойства файлов в бакете S3 до создания внешних таблиц:

- Быстро просмотреть файлы
- Проверить права доступа
- Проверить пути и параметры хранения данных

Пример запроса для чтения данных:

```
SELECT
  *
FROM
  s3_data_source.`*.tsv`
WITH
(
  FORMAT = "tsv_with_names",
  SCHEMA =
  (
    ts Uint32,
    action Utf8
  )
);
```

Список поддерживаемых форматов и алгоритмов сжатия данных для чтения из S3-совместимого хранилища данных приведен в разделе Форматы данных и алгоритмы сжатия.

Модель данных

В S3 данные хранятся в файлах. Для чтения данных необходимо указать формат данных, сжатие, списки полей. Для этого используется следующее SQL-выражение:

```
SELECT
  <expression>
FROM
  <s3_external_datasource_name>.<file_path>
WITH
(
  FORMAT = "<file_format>",
  COMPRESSION = "<compression>",
  SCHEMA = (<schema_definition>),
  <format_settings>
```

```
)
WHERE
  <filter>;
```

Где:

- `s3_external_datasource_name` — название внешнего источника данных, ведущего на бакет с S3-совместимым хранилищем данных.
- `file_path` — путь к файлу или файлам внутри бакета. Поддерживаются подстановочные знаки `*`, `?`, `{ ... }`; подробнее в разделе.
- `file_format` — формат данных в файлах, обязательно.
- `compression` — формат сжатия файлов, опционально.
- `schema_definition` — описание схемы хранимых данных в файлах, обязательно.
- `format_settings` — параметры форматирования, опционально.

Описание схемы данных

Описание схемы данных состоит из набора полей:

- Названия поля.
- Типа поля.
- Признака обязательности данных.

Например, схема данных ниже описывает поле схемы с названием `Year` типа `Int32` и требованием наличия этого поля в данных:

```
Year Int32 NOT NULL
```

Если поле данных помечено как обязательное, `NOT NULL`, но это поле отсутствует в обрабатываемом файле, то работа с таким файлом будет завершена с ошибкой. Если поле помечено как необязательное, `NULL`, то при отсутствии поля в обрабатываемом файле ошибки не возникнет, но поле примет значение `NULL`. Ключевое слово `NULL` в необязательных полях является опциональным.

Список поддерживаемых типов данных, которые можно указать в схеме в зависимости от формата данных, приведён в разделе Поддерживаемые типы данных.

Автоматический вывод схемы данных

YDB может автоматически определять схему данных в файлах бакета, чтобы вам не пришлось указывать все поля схемы вручную.

> **Примечание**
>
> Автоматический вывод схемы доступен для всех форматов данных, кроме `raw` и `json_as_string`. Для этих форматов придётся описывать схему данных вручную.

Чтобы включить автоматический вывод схемы, используйте параметр `WITH_INFER`:

```
SELECT
  <expression>
FROM
  <s3_external_datasource_name>.<file_path>
WITH
(
  FORMAT = "<file_format>",
  COMPRESSION = "<compression>",
  WITH_INFER = "true"
)
WHERE
  <filter>;
```

Где:

- `s3_external_datasource_name` — название внешнего источника данных, ведущего на S3 бакет.
- `file_path` — путь к файлу или файлам внутри бакета. Поддерживаются подстановочные знаки `*`, `?`, `{ ... }`; подробнее ниже.
- `file_format` — формат данных в файлах. Поддерживаются все форматы, кроме `raw` и `json_as_string`.
- `compression` — опциональный формат сжатия файлов.

В результате выполнения такого запроса будут автоматически выведены названия и типы полей.

Ограничения для автоматического вывода схемы:

- Вывод схемы делается по данным только из одного произвольного непустого файла.
- Для форматов данных `csv_with_names`, `tsv_with_names`, `json_list`, `json_each_row` вывод схемы выполняется по первым 10 МБ данных из файла.
- Вывод схемы для файлов с форматом `parquet` возможен только в случае, если размер метаданных файла не превышает 10 МБ.
- Если файлы имеют разную схему, то запрос завершится с ошибкой парсинга данных в случае несовпадения типов колонок или пропуска не опциональных колонок.

Форматы путей к данным

В YDB поддерживаются следующие пути к данным:

Формат пути	Описание	Пример

Путь завершается символом `/`	Путь к каталогу	Путь `/a/` адресует все содержимое каталога:
		`/a/b/c/d/1.txt`
		`/a/b/2.csv`

Путь содержит символ подстановки `?`	Файлы, имеющие путь с возможным отличием в одном символе	Путь `/a?c/1.csv` адресует файлы в каталогах:
		`/abc/1.csv`
		`/afc/1.csv`
		`/adc/1.csv`

Путь содержит символ макроподстановки `*`	Любые файлы, вложенные в путь	Путь `/a/*.csv` адресует файлы в каталогах:
		`/a/b/c/1.csv`
		`/a/2.csv`
		`/a/b/c/d/e/f/g/2.csv`

Путь содержит символы `{` и `}`	Файлы, имеющие путь, соответствующий одному из вариантов в фигурных скобках	Путь `/{a,b,c}/1.csv` адресует файлы в каталогах:
		`/a/1.csv`
		`/b/f.csv`
		`/c/1.csv`

Путь не завершается символом `/` и не содержит символов макроподстановок	Путь к отдельному файлу	Путь `/a/b.csv` адресует конкретный файл `/a/b.csv`

Такие же значения можно использовать для параметра `file_pattern`.

Параметры форматирования

В YDB поддерживаются следующие параметры форматирования:

Имя параметра	Описание	Принимаемые значения
`file_pattern`	Шаблон имени файла	Строка шаблона имени. Поддерживаются подстановочные знаки `*`, `?`, `{ ... }`.
`data.interval.unit`	Единица измерения для парсинга типа `Interval`	`MICROSECONDS`, `MILLISECONDS`, `SECONDS`, `MINUTES`, `HOURS`, `DAYS`, `WEEKS`
`data.datetime.format_name`	Предопределенный формат, в котором записаны данные типа `Datetime`	`POSIX`, `ISO`
`data.datetime.format`	Шаблон, определяющий как записаны данные типа `Datetime`	Строка форматирования, например: `%Y-%m-%dT%H-%M`
`data.timestamp.format_name`	Предопределенный формат, в котором записаны данные типа `Timestamp`	`POSIX`, `ISO`, `UNIX_TIME_SECONDS`, `UNIX_TIME_MILLISECONDS`, `UNIX_TIME_MICROSECONDS`
`data.timestamp.format`	Шаблон, определяющий как записаны данные типа `Timestamp`	Строка форматирования, например: `%Y-%m-%dT%H-%M-%S`
`data.date.format`	Формат, в котором записаны данные типа `Date`	Строка форматирования, например: `%Y-%m-%d`
`csv_delimiter`	Разделитель данных в формате `csv_with_names`	Любой символ (UTF-8)

Подробное описание поддерживаемых подстановочных знаков для параметра `file_pattern` приведено выше. Параметр `file_pattern` можно использовать только в том случае, если `file_path` — путь к каталогу.

В строках форматирования для даты и времени можно использовать любые шаблонные переменные, поддерживаемые функцией `strftime` (C99). В YDB поддерживаются следующие форматы типов `Datetime` и `Timestamp`:

Имя	Описание	Пример
`POSIX`	Строка формата `%Y-%m-%d %H:%M:%S`	2001-03-26 16:10:00
`ISO`	Формат, соответствующий стандарту ISO 8601	2001-03-26 16:10:00Z
`UNIX_TIME_SECONDS`	Количество секунд, прошедших с 1 января 1970 года (00:00:00 UTC)	985623000
`UNIX_TIME_MILLISECONDS`	Количество миллисекунд, прошедших с 1 января 1970 года (00:00:00 UTC)	985623000000
`UNIX_TIME_MICROSECONDS`	Количество микросекунд, прошедших с 1 января 1970 года (00:00:00 UTC)	985623000000000

Пример

Пример запроса для чтения данных из S3-совместимого хранилища данных:

```
SELECT
  *
FROM
  external_source.`folder/`
WITH(
  FORMAT = "csv_with_names",
```

```
  COMPRESSION = "gzip"
  SCHEMA =
  (
    Id Int32 NOT NULL,
    UserId Int32 NOT NULL,
    TripDate Date NOT NULL,
    TripDistance Double NOT NULL,
    UserComment Utf8
  ),
  FILE_PATTERN = "*.csv.gz",
  `DATA.DATE.FORMAT` = "%Y-%m-%d",
  CSV_DELIMITER = "/"
);
```

Где:

- `external_source` — название внешнего источника данных, ведущего на бакет S3-совместимого хранилища данных.
- `folder/` — путь к папке с данными в бакете S3.
- `SCHEMA` — описание схемы данных в файле.
- `*.csv.gz` — шаблон имени файлов с данными.
- `%Y-%m-%d` — формат записи данных типа `Date` в S3.

Чтение из бакетов S3 через внешние таблицы

Иногда одни и те же запросы к данным нужно выполнять регулярно. Чтобы не указывать все детали работы с этими данными при каждом вызове запроса, используйте режим с внешними таблицами. В этом случае запрос выглядит, как обычный запрос к таблицам YDB.

Пример запроса для чтения данных:

```
SELECT
    *
FROM
    s3_test_data
WHERE
    version > 1
```

Создание внешней таблицы, ведущей на бакет S3-совместимого хранилища данных

Чтобы создать внешнюю таблицу, описывающую бакет S3, выполните следующий SQL-запрос. Запрос создает внешнюю таблицу с именем `s3_test_data`, в котором расположены файлы в формате `CSV` со строковыми полями `key` и `value`, находящиеся внутри бакета по пути `test_folder`, при этом для указания реквизитов подключения используется объект внешний источник данных `bucket`:

```
CREATE EXTERNAL TABLE s3_test_data (
  key Utf8 NOT NULL,
  value Utf8 NOT NULL
) WITH (
  DATA_SOURCE="bucket",
  LOCATION="folder",
  FORMAT="csv_with_names",
  COMPRESSION="gzip"
);
```

Где:

- `key, value` - список колонок данных и их типов, список допустимых типов описан в разделе Поддерживаемые типы данных;
- `bucket` - имя внешнего источника данных к S3-совместимому хранилищу данных;
- `folder` - путь внутри бакета с данными. Поддерживаются подстановочные знаки `*`, `?`, `{ ... }`; подробнее в разделе;
- `csv_with_names` - один из допустимых типов хранения данных;
- `gzip` - один из допустимых алгоритмов сжатия.

Также при создании внешних таблиц поддерживаются параметры форматирования.

Модель данных

Чтение данных с помощью внешних таблиц из S3-совместимого хранилища данных выполняется с помощью обычных SQL-запросов, как к обычной таблице.

```
SELECT
    <expression>
FROM
    s3_test_data
WHERE
    <filter>;
```

Ограничения

При работе с бакетами S3-совместимого хранилища данных существует ряд ограничений.

Ограничения:

1. Поддерживаются только запросы чтения данных - `SELECT` и `INSERT`, остальные виды запросов не поддерживаются.
2. Если значение даты, хранящейся во внешнем источнике данных, находится вне допустимого для YDB диапазона (все используемые даты должны быть позднее 1970-01-01, но ранее 2105-12-31), в YDB такое значение будет преобразовано в `NULL`.

Запись данных в бакеты S3

В YDB для записи данных в бакет S3 можно использовать внешние источники данных или внешние таблицы.

Запись в S3 через внешний источник данных

Запись данных с помощью внешних источников данных удобно использовать для прототипирования и первоначальной настройки работы с записью данных. В общем виде запрос записи в S3-совместимое хранилище данных с использованием внешнего источника данных выглядит следующим образом:

```
INSERT INTO
    <s3_external_datasource_name>.<write_folder>
WITH
(
    FORMAT = "<file_format>",
    COMPRESSION = "<compression>",
    SCHEMA = (<schema_definition>),
    <format_settings>
)
    <expression>
```

Где:

- `s3_external_datasource_name` — название внешнего источника данных, ведущего на бакет с S3-совместимого хранилища данных.
- `write_folder` — путь к директории внутри бакета, в которую будет вестись запись; путь должен быть не пустым и заканчиваться на `/`; запись в корень бакета не поддерживается.
- `file_format` — формат данных в файлах, обязательно.
- `compression` — формат сжатия файлов, опционально.
- `schema_definition` — описание схемы хранимых данных в файлах, опционально; если не указано, будет выведено автоматически из типа `<expression>`.
- `format_settings` — параметры форматирования, опционально.

SQL-выражение иллюстрирует запись данных во внешний источник данных напрямую.

```
INSERT INTO external_source.`test/`
WITH
(
    FORMAT = "csv_with_names"
)
SELECT
    "value" AS value, "name" AS name
```

Запись будет произведена в указанный путь, создаваемые при записи файлы получают случайные имена. Для задания партицирования данных нужно использовать `PARTITIONED_BY` в секции `WITH`, так же как и при чтении из внешних источников данных, подробнее здесь. Расширенное партицирование не поддерживается для записи в S3. Партицирование данных при записи поддерживается для всех форматов записи, кроме `json_list` и `raw`.

При работе с внешними источниками данных и таблицами возможны только операции чтения (`SELECT`) и вставки (`INSERT`) данных, другие виды операций не поддерживаются.

Запись данных через внешние таблицы

Если записывать данные нужно регулярно, то удобно делать это с помощью внешних таблиц. При этом нет необходимости указывать все детали работы с этими данными в каждом запросе. Для записи данных в бакет создайте внешнюю таблицу для S3-совместимого хранилища данных и используйте обычное SQL-выражение `INSERT INTO`:

```
INSERT INTO test
SELECT
    "value" AS value, "name" AS name
```

Экспорт данных в объектное хранилище S3

YDB поддерживает экспорт данных из таблиц в S3 с помощью внешнего источника данных.

Для экспорта необходимо использовать запрос записи во внешний источник данных с указанием формата экспортируемых файлов:

```
INSERT INTO external_source.`test/`
WITH
(
    FORMAT = "parquet"
)
SELECT
    *
FROM table
```

Экспортировать можно как обычные YDB таблицы, так и любые внешние таблицы.

> **Совет**
>
> Рекомендованным форматом экспорта является `parquet`, для него оптимизированы сценарии импорта и экспорта.

Для экспорта в S3-совместимое хранилище данных всех таблиц, расположенных в определённой директории в схеме данных, а также информации о расположении таблиц, директорий и индексов таблиц, нужно использовать YDB CLI. Подробнее см. в статье Выгрузка в S3-совместимое хранилище.

Форматы данных и алгоритмы сжатия

В данном разделе описываются поддерживаемые в YDB форматы данных, хранимых в S3, поддерживаемые алгоритмы сжатия и список поддерживаемых YQL типов для каждого формата данных.

Поддерживаемые форматы данных

Список поддерживаемых в YDB форматов данных приведен в таблице ниже.

Формат	Чтение	Запись
`csv`	✓	
`csv_with_names`	✓	✓
`tsv_with_names`	✓	✓
`json_list`	✓	✓
`json_each_row`	✓	✓
`json_as_string`	✓	
`parquet`	✓	✓
`raw`	✓	✓

Формат csv

Данный формат основан на формате CSV и отличается от `csv_with_names` тем, что в нём нет заголовка с названиями столбцов. Все строки файла трактуются как данные, а имена колонок берутся из схемы запроса в порядке их объявления в `SCHEMA`.

> **Примечание**
>
> Формат `csv` доступен только для чтения данных из S3. Запись в этом формате не поддерживается.

Пример данных (без строки заголовка):

```
1997,Man_1,Model_1,3000.00
1999,Man_2,Model_2,4900.00
```

Пример запроса

```
SELECT
    *
FROM external_source.path
WITH
(
    FORMAT = "csv",
    SCHEMA =
    (
        Year Int32,
        Manufacturer Utf8,
        Model Utf8,
        Price Double
    )
)
```

Результат выполнения запроса:

#	Manufacturer	Model	Price	Year
1	Man_1	Model_1	3000	1997
2	Man_2	Model_2	4900	1999

Формат csv_with_names

Данный формат основан на формате CSV. Данные размещены в колонках, разделены запятыми, в первой строке файла находятся имена колонок. В отличие от формата `csv`, где имена колонок берутся из схемы запроса и сопоставляются позиционно, в формате `csv_with_names` имена колонок читаются непосредственно из заголовочной строки файла и сопоставляются со схемой по именам.

Пример данных:

```
Year,Manufacturer,Model,Price
1997,Man_1,Model_1,3000.00
1999,Man_2,Model_2,4900.00
```

Пример запроса

```
SELECT
    *
```

```
FROM external_source.path
WITH
(
    FORMAT = "csv_with_names",
    SCHEMA =
    (
        Year Int32,
        Manufacturer Utf8,
        Model Utf8,
        Price Double
    )
)
```

Результат выполнения запроса:

#	Manufacturer	Model	Price	Year
1	Man_1	Model_1	3000	1997
2	Man_2	Model_2	4900	1999

Формат tsv_with_names

Данный формат основан на формате TSV. Данные размещены в колонках, разделены символами табуляции (код `0x9`), в первой строке файла находятся имена колонок.

Пример данных:

```
Year    Manufacturer    Model   Price
1997    Man_1   Model_1    3000.00
1999    Man_2   Model_2    4900.00
```

Пример запроса

```
SELECT
    *
FROM external_source.path
WITH
(
    FORMAT = "tsv_with_names",
    SCHEMA =
    (
        Year Int32,
        Manufacturer Utf8,
        Model Utf8,
        Price Double
    )
)
```

Результат выполнения запроса:

#	Manufacturer	Model	Price	Year
1	Man_1	Model_1	3000	1997
2	Man_2	Model_2	4900	1999

Формат json_list

Данный формат основан на JSON-представлении данных. В этом формате внутри каждого файла должен находиться объект в корректном JSON-представлении.

Пример корректных данных (данные представлены в виде списка объектов JSON):

```
[
    { "Year": 1997, "Manufacturer": "Man_1", "Model": "Model_1", "Price": 3000.0 },
    { "Year": 1999, "Manufacturer": "Man_2", "Model": "Model_2", "Price": 4900.00 }
]
```

Пример **НЕ**корректных данных (на каждой отдельной строке находится отдельный объект в формате JSON, но эти объекты не объединены в список):

```
{ "Year": 1997, "Manufacturer": "Man_1", "Model": "Model_1", "Price": 3000.0 }
{ "Year": 1999, "Manufacturer": "Man_2", "Model": "Model_2", "Price": 4900.00 }
```

Пример запроса

```
SELECT
    *
FROM external_source.path
WITH
(
    FORMAT = "json_list",
    SCHEMA =
```

```
    (
        Year Int32,
        Manufacturer Utf8,
        Model Utf8,
        Price Double
    )
)
```

Результат выполнения запроса:

#	Manufacturer	Model	Price	Year
1	Man_1	Model_1	3000	1997
2	Man_2	Model_2	4900	1999

Формат json_each_row

Данный формат основан на JSON-представлении данных. В этом формате внутри каждого файла на каждой отдельной строке файла должен находиться объект в корректном JSON-представлении, но эти объекты не объединены в JSON-список. Такой формат используется при передаче данных через потоковые системы, например, Apache Kafka или Топики YDB.

Пример корректных данных (на каждой отдельной строке находится отдельный объект в формате JSON, но эти объекты не объединены в список):

```
{ "Year": 1997, "Manufacturer": "Man_1", "Model": "Model_1", "Price": 3000.0 },
{ "Year": 1999, "Manufacturer": "Man_2", "Model": "Model_2", "Price": 4900.00 }
```

Пример запроса

```
SELECT
    *
FROM external_source.path
WITH
(
    FORMAT = "json_each_row",
    SCHEMA =
    (
        Year Int32,
        Manufacturer Utf8,
        Model Utf8,
        Price Double
    )
)
```

Результат выполнения запроса:

#	Manufacturer	Model	Price	Year
1	Man_1	Model_1	3000	1997
2	Man_2	Model_2	4900	1999

Формат json_as_string

Данный формат основан на JSON-представлении данных. Формат `json_as_string` не разбивает входной JSON-документ на поля, а представляет каждую строку файла в виде одного объекта JSON (или одной строки). Такой формат удобен, если список полей не одинаков во всех строках, а может изменяться.

В этом формате внутри каждого файла должен находиться:

- объект в корректном JSON-представлении в каждой отдельной строке файла;
- объекты в корректном JSON-представлении, объединенные в список.

Пример корректных данных (данные представлены в виде списка объектов JSON):

```
{ "Year": 1997, "Attrs": { "Manufacturer": "Man_1", "Model": "Model_1" }, "Price": 3000.0 }
{ "Year": 1999, "Attrs": { "Manufacturer": "Man_2", "Model": "Model_2" }, "Price": 4900.00 }
```

В этом формате схема читаемых данных должна состоять только из одной колонки с одним из разрешённых типов данных, подробнее см. ниже.

Пример запроса

```
SELECT
    CAST(JSON_VALUE(Data, "$.Year") AS Int32) AS Year,
    JSON_VALUE(Data, "$.Attrs.Manufacturer") AS Manufacturer,
    JSON_VALUE(Data, "$.Attrs.Model") AS Model,
    CAST(JSON_VALUE(Data, "$.Price") AS Double) AS Price
FROM external_source.path
WITH
(
    FORMAT = "json_as_string",
    SCHEMA =
    (
```

```
        Data Json
    )
)
```

Результат выполнения запроса:

#	Manufacturer	Model	Price	Year
1	Man_1	Model_1	3000	1997
2	Man_2	Model_2	4900	1999

Формат parquet

Данный формат позволяет считывать содержимое файлов в формате Apache Parquet.

Поддерживаемые алгоритмы сжатия данных внутри файлов Parquet для чтения из S3:

- Без сжатия
- SNAPPY
- GZIP
- BROTLI
- LZ4
- ZSTD
- LZ4_RAW

Примечание

Запись в формате Parquet будет производится с использованием алгоритма сжатия Snappy.

Пример запроса

```
SELECT
    *
FROM external_source.path
WITH
(
    FORMAT = "parquet",
    SCHEMA =
    (
        Year Int32,
        Manufacturer Utf8,
        Model Utf8,
        Price Double
    )
)
```

Результат выполнения запроса:

#	Manufacturer	Model	Price	Year
1	Man_1	Model_1	3000	1997
2	Man_2	Model_2	4900	1999

Формат raw

Данный формат позволяет считывать содержимое файлов как есть, в "сыром" виде. Считанные таким образом данные можно обработать средствами YQL, разделив на строки и столбцы.

Этот формат стоит использовать, если встроенных возможностей парсинга исходных данных в YDB недостаточно. В этом формате схема данных для чтения и записи должна состоять только из одной колонки с одним из разрешённых типов данных, подробнее см. ниже.

Примечание

Размер каждого из считываемых файлов в формате `raw` не может превышать общего ограничения на потребление памяти одним запросом в YDB.

Пример запроса

Для парсинга следующих данных, где строки разделены точкой с запятой, а значения запятой:

```
1997,Man_1,Model_1,3000.00;
1999,Man_2,Model_2,4900.00;
```

Можно использовать запрос:

```
$input = SELECT
    String::SplitToList(        -- разделение каждой строки по ','
        String::Strip(RowData), -- удаление всех пробельных символов из начала и конца строк
        ","
```

```
    ) AS Row
FROM external_source.path
WITH
(
    FORMAT = "raw",
    SCHEMA =
    (
        FileData Utf8
    )
)
FLATTEN LIST BY (String::SplitToList(FileData, ";", TRUE AS SkipEmpty) AS RowData); -- разделение файла п
о ';'

SELECT -- Получение нужных колонок
    CAST(Row[0] AS Int32) AS Year,
    Row[1] AS Manufacturer,
    Row[2] AS Model,
    CAST(Row[3] AS Double) AS Price
FROM $input
```

Результат выполнения запроса:

#	Manufacturer	Model	Price	Year
1	Man_1	Model_1	3000	1997
2	Man_2	Model_2	4900	1999

Поддерживаемые алгоритмы сжатия

Использование алгоритмов сжатия зависит от форматов файлов. Для всех форматов файлов за исключением Parquet возможно использование следующих алгоритмов сжатия:

Алгоритм	Название в YDB	Чтение	Запись
Gzip	gzip	✓	✓
Zstd	zstd	✓	✓
LZ4	lz4	✓	✓
Brotli	brotli	✓	✓
Bzip2	bzip2	✓	✓
Xz	xz	✓	✓

Для формата файлов Parquet поддерживаются собственные внутренние алгоритмы сжатия:

Формат сжатия	Чтение	Запись
Без сжатия	✓	
Snappy	✓	✓
Gzip	✓	
Brotli	✓	
LZ4	✓	
Zstd	✓	
LZ4_RAW	✓	

В YDB не поддерживается работа со сжатыми "снаружи" parquet-файлами, например, с файлами вида `<myfile>.parquet.gz` или аналогичными. Все файлы в формате Parquet должны быть без внешнего сжатия.

Поддерживаемые типы данных

Таблица всех поддерживаемых типов при чтении из S3 в схеме запроса:

Тип	csv	csv_with_names	tsv_with_names	json_list	json_each_row	json_as_string	parquet	raw
`Bool`, `Int8`, `Int16`, `Int32`, `Int64`, `Uint8`, `Uint16`, `Uint32`, `Uint64`, `Float`, `Double`	✓	✓	✓	✓	✓		✓	
`DyNumber`				✓				
`String`, `Utf8`, `Json`	✓	✓	✓	✓	✓	✓	✓	✓

`JsonDocument`							✓	
`Yson`				✓			✓	✓
`Uuid`	✓	✓	✓		✓			
`Date`, `Datetime`, `Timestamp`, `TzDate`, `TzDateTime`, `TzTimestamp`	✓	✓	✓		✓		✓	
`Interval`	✓	✓	✓		✓		✓	
`Date32`, `Datetime64`, `Timestamp64`, `Interval64`, `TzDate32`, `TzDateTime64`, `TzTimestamp64`							✓	
`Optional<T>`	✓	✓	✓	✓	✓	✓	✓	✓

Таблица всех поддерживаемых типов при записи в S3:

Тип	csv_with_names	tsv_with_names	json_list	json_each_row	parquet	raw
`Bool`, `Int8`, `Int16`, `Int32`, `Int64`, `Uint8`, `Uint16`, `Uint32`, `Uint64`, `Float`, `Double`	✓	✓	✓	✓	✓	
`DyNumber`			✓			
`String`, `Utf8`, `Json`	✓	✓	✓	✓	✓	✓
`JsonDocument`						
`Yson`			✓			✓
`Uuid`	✓	✓		✓		
`Date`, `Datetime`, `Timestamp`, `TzDate`, `TzDateTime`, `TzTimestamp`	✓	✓		✓	✓	
`Interval`						
`Date32`, `Datetime64`, `Timestamp64`, `Interval64`, `TzDate32`, `TzDateTime64`, `TzTimestamp64`						
`Optional<T>`	✓	✓	✓	✓	✓	

Для всех форматов чтения из S3 и записи в S3, кроме `json_list`, разрешено использовать тип `Optional<T>` только в том случае, если `T` — примитивный YQL тип. Подробнее об опциональных типах см. в статье column0.

Отображение типов при чтении и записи данных в формате Parquet

При чтении и записи данных в формате Parquet YDB использует логическую схему типов Apache Arrow — стандарт, применяемый в Parquet для описания семантики данных. Ниже приведено соответствие между типами YQL и логическими типами Arrow, которые записываются в Parquet-файл.

Тип YQL	Тип Arrow при экспорте	Тип Arrow при импорте	Комментарий
`Bool`	`UINT8`	`BOOL`, `UINT8`	
`Int8`	`INT8`	`INT8`	
`Int16`	`INT16`	`INT16`	
`Int32`	`INT32`	`INT32`	
`Int64`	`INT64`	`INT64`	
`Uint8`	`UINT8`	`UINT8`	
`Uint16`	`UINT16`	`UINT16`	
`Uint32`	`UINT32`	`UINT32`	
`Uint64`	`UINT64`	`UINT64`	
`Float`	`FLOAT (32)`	`FLOAT (32)`	
`Double`	`FLOAT (64)`	`FLOAT (64)`	
`Date`	`UINT16`	`UINT16`, `INT32`, `UINT32`, `INT64`, `UINT64`, `DATE`, `TIMESTAMP (s, ms, us)`	количество дней, прошедших с начала Unix-эпохи
`Date32`	`DATE (s)`	`DATE (s)`	
`Datetime`	`UINT32`	`UINT16`, `INT32`, `UINT32`, `INT64`, `UINT64`, `DATE`, `TIMESTAMP (s, ms, us)`	количество секунд, прошедших с начала Unix-эпохи
`Datetime64`	`TIMESTAMP (us)`	`TIMESTAMP (s, ms, us)`	
`Timestamp`	`TIMESTAMP (us)`	`TIMESTAMP (s, ms, us)`	
`Timestamp64`	`TIMESTAMP (us)`	`TIMESTAMP (s, ms, us)`	
`Decimal(s, p)`	`DECIMAL (128, s, p)`	`DECIMAL (128, s, p)`	
`String`	`BINARY`	`BINARY`	
`Utf8`	`BINARY`	`BINARY`	
`Json`	`BINARY`	`BINARY`	

Партицирование данных

В S3-совместимом хранилище данных можно хранить очень большие объемы данных. При этом запросы к этим данным могут затрагивать не все данные, а только их часть. Если описать правила разметки структуры хранения ваших данных в YDB, тогда данные, которые не нужны для запроса, можно даже не считывать из S3. Это значительно ускоряет выполнение запросов без влияния на результат.

Например, данные хранятся в следующей структуре каталогов:

```
year=2021
    month=01
    month=02
    month=03
year=2022
    month=01
```

Запрос ниже явно подразумевает, что обработать нужно только данные за февраль 2021-го года и другие данные не нужны.

```
SELECT
    *
FROM
    s3_data_source.`/`
WITH
(
    SCHEMA =
    (
        data String,
        year Int32,
        month Int32
    )
)
WHERE
    year=2021
    AND month=02
```

Если схема партицирования данных не указана, то из S3-совместимого хранилища данных будут считаны *все* хранимые данные, но в результате обработки данные за все другие даты будут отброшены.

Если описать структуру хранения явно, указав, что данные в S3-совместимом хранилище данных размещены в каталогах по годам и месяцам

```
SELECT
    *
FROM
    s3_data_source.`/`
WITH
(
    SCHEMA =
    (
        data String,
        year Int32,
        month Int32
    ),
    PARTITIONED_BY = "['year', 'month']"
)
WHERE
    year=2021
    AND month=02
```

то в процессе исполнения запроса из S3-совместимого хранилища данных будут считаны не все данные, а только данные за февраль 2021-го года, что существенно сократит объем обрабатываемых данных и ускорит обработку, при этом результаты обоих запросов будут идентичны.

> **Примечание**
>
> В примере выше показана работа с данными на уровне внешних источников данных. Такой пример выбран только для иллюстративных целей. Мы настоятельно рекомендуем для работы с данными использовать внешние таблицы и не использовать прямую работу с внешними источниками.

Синтаксис

При работе на уровне внешних источников данных партицирование задается с помощью параметра `partitioned_by`, где в JSON-формате задается список колонок.

```
SELECT
    *
FROM
    <s3_external_data_source>.<path>
WITH
(
    SCHEMA=(<field1>, <field2>, <field3>),
    PARTITIONED_BY="['field2', 'field3']"
)
```

В параметре `partitioned_by` перечисляются колонки схемы данных, по которым партицированы хранимые в S3 данные. Порядок указания полей в параметре partitioned_by определяет вложенность каталогов S3-совместимого хранилища данных

друг в друга.

Например, `PARTITIONED_BY=['year', 'month']` определяет структуру каталогов

```
year=2021
    month=01
    month=02
    month=03
year=2022
    month=01
```

А `partitioned_by=['month', 'year']` определяет другую структуру каталогов

```
month=01
    year=2021
    year=2022
month=02
    year=2021
month=03
    year=2021
```

Поддерживаемые типы данных

Партицирование возможно только по следующему набору типов данных YQL:

- Uint16, Uint32, Uint64
- Int16, Int32, Int64
- String, Utf8

При использовании других типов для указания партицирования возвращается ошибка.

Поддерживаемые форматы путей хранения

Формат путей хранения, когда в имени каждого каталога явно указано название колонки, называется "Hive-Metastore форматом" или просто "Hive-форматом".

Такой формат выглядит следующим образом:

```
month=01
    year=2021
    year=2022
month=02
    year=2021
month=03
    year=2021
```



Важно

Базовый режим партицирования в YDB поддерживает только Hive-формат.

Для указания произвольных путей хранения необходимо воспользоваться режимом Расширенного партицирования данных.

Расширенное партицирование

Партицирование позволяет подсказать для YDB правила размещения данных в S3-совместимом хранилище данных.

Предположим, что данные в S3 хранятся в следующей структуре каталогов:

```
year=2021
    month=01
    month=02
    month=03
year=2022
    month=01
```

При выполнении запроса ниже YDB выполнит следующие действия:

1. Получит полный список подкаталогов внутри '/'.
2. Для каждого подкаталога выполнит попытку обработать имя подкаталога в формате `year=<DIGITS>`.
3. Для каждого подкаталога `year=<DIGITS>` получит список всех подкаталогов в формате `month=<DIGITS>`.
4. Обработает считанные данные.

```
SELECT
    *
FROM
    s3_data_source.`/`
WITH
(
    SCHEMA =
    (
        data String,
        year Int32,
        month Int32
    ),
    PARTITIONED_BY = "['year', 'month']"
)
WHERE
    year=2021
    AND month=02
```

То есть при работе с партицированными данными выполняется полный листинг содержимого S3-совместимого хранилища данных, что может занимать продолжительное время на бакетах большого размера.

Для оптимизации работы на больших объемах данных следует использовать "расширенное партицирование". В этом режиме не происходит сканирования каталогов S3-совместимого хранилища данных, вместо этого все пути вычисляются заранее и обращение происходит только к ним.

Для работы расширенного партицирования необходимо задать правила работы через специальный параметр - "projection". В этом параметре описываются все правила размещения данных в каталогах S3-совместимого хранилища данных.

Синтаксис

Расширенное партицирование называется "partition projection" и задается через параметр `projection`.

Пример указания расширенного партицирования:

```

SELECT
    *
FROM
    <s3_external_data_source>.`/`
WITH
(
    SCHEMA =
    (
        data String,
        year Int32,
        month Int32
    ),
    PARTITIONED_BY = "['year', 'month']",
    `projection.enabled` : "true",

    `projection.year.type` : "integer",
    `projection.year.min` : "2010",
    `projection.year.max` : "2022",
    `projection.year.interval` : "1",

    `projection.month.type` : "integer",
    `projection.month.min` : "1",
    `projection.month.max` : "12",
    `projection.month.interval` : "1",
    `projection.month.digits` : "2",

    `storage.location.template` : "${year}/${month}"
)
```

В примере выше указывается, что данные существуют за каждый год и каждый месяц с 2010 по 2022 годы, при этом в бакете данные размещены в каталогах вида `2022/12`. Если данные за какой-то период отсутствуют внутри бакета, то это не приводит к ошибкам, запрос выполнится успешно, а данные будут пропущены в расчетах.

В общем виде настройка расширенного партицирования выглядит следующим образом:

```
SELECT
    *
FROM
    <s3_external_data_source>.<path>
WITH
(
    SCHEMA = (<fields>, <field1>, <field2>),
    PARTITIONED_BY = "'['field1', 'field2']",
    `projection.enabled` : <"true"|"false">,

    `projection.<field1_name>.type` : "<type>",
    `projection.<field1_name>....` : "<extended_properties>",

    `projection.<field2_name>.type` : "<type>",
    `projection.<field2_name>....` : "<extended_properties>",

    `storage.location.template` : ".../${field2}/${field1}/..."

)
```

Описание полей

Название поля	Описание поля	Допустимые значения
`projection.enabled`	Включено или нет расширенное партицирование	`true`, `false`
`projection.<field1_name>.type`	Тип данных поля	`integer`, `enum`, `date`
`projection.<field1_name>.XXX`	Свойства конкретного типа	

Поле типа integer

Используется для колонок, чьи значения можно представить в виде целых чисел диапазона 2^{-63} до 2^{63}-1.

Название поля	Обязательное	Описание поля	Пример значений
`projection.<field_name>.type`	Да	Тип данных поля	integer
`projection.<field_name>.min`	Да	Определяет минимально допустимое значение. Задается в виде целого числа	-100 004
`projection.<field_name>.max`	Да	Определяет максимально допустимое значение. Задается в виде целого числа	-10 5000
`projection.<field_name>.interval`	Нет, по умолчанию `1`	Определяет шаг между элементами внутри диапазона значений. Например, шаг 3 при диапазоне значений 2, 10, приведет к следующим значениям: 2, 5, 8	2 11
`projection.<field_name>.digits`	Нет, по умолчанию `0`	Определяет количество цифр в числе. Если ненулевых цифр в числе меньше, чем указанное значение, то значение дополняется нулями спереди вплоть до числа указанного числа цифр. Например, если указано значение .digits=3, а передается число 2, то оно будет превращено в значение 002	2 4

Поле типа enum

Используется для колонок, чьи значения можно представить в виде перечисления значений.

Название поля	Обязательное	Описание поля	Пример значений
`projection.<field_name>.type`	Да	Тип данных поля	enum
`projection.<field_name>.values`	Да	Определяет допустимые значения, указанные через запятую. Пробелы не игнорируются	1, 2 A,B,C

Поле типа date

Используется для колонок, чьи значения можно представить в виде даты. Допустимый диапазон дат с 1970-01-01 до 2105-01-01.

Название поля	Обязательное	Описание поля	Пример значений
`projection.<field_name>.type`	Да	Тип данных поля	date
`projection.<field_name>.min`	Да	Определяет минимально допустимую дату. Разрешены значения в формате `YYYY-MM-DD` или в виде выражения, содержащего специальную макроподстановку NOW	
`projection.<field_name>.max`	Да	Определяет максимально допустимую дату. Разрешены значения в формате `YYYY-MM-DD` или в виде выражения, содержащего специальную макроподстановку NOW	2020-01-01 NOW-5DAYS NOW+3HOURS

`projection.<field_name>.format`	Да	Строка форматирования даты на основе strptime	%Y-%m-%d %D
`projection.<field_name>.unit`	Нет, по умолчанию `DAYS`	Единицы измерения интервалов времени. Допустимые значения: `YEARS`, `MONTHS`, `WEEKS`, `DAYS`, `HOURS`, `MINUTES`, `SECONDS`, `MILLISECONDS`	SECONDS YEARS
`projection.<field_name>.interval`	Нет, по умолчанию `1`	Определяет шаг между элементами внутри диапазона значений с размерностью, указанной в `projection.<field_name>.unit`. Например, для диапазона 2021-02-02, 2021-03-05 шаг 15 с размерностью DAYS приведет к значениям: 2021-02-17, 2021-03-04	2 6

Работа с макроподстановкой NOW

1. Поддерживается ряд арифметических операция с макроподстановкой NOW: прибавление интервала времени и вычитание интервала времени. Например: `NOW-3DAYS`, `NOW+1MONTH`, `NOW-6YEARS`, `NOW + 4HOURS`, `NOW - 5MINUTES`, `NOW + 6SECONDS`. Возможные варианты использования макроподстановки описываются регулярным выражением: `^\s*(NOW)\s*(([\+\-])\s*([0-9]+)\s*(YEARS?|MONTHS?|WEEKS?|DAYS?|HOURS?|MINUTES?|SECONDS?)\s*)?$`
2. Допустимые размерности интервалов: YEARS, MONTHS, WEEKS, DAYS, HOURS, MINUTES, SECONDS, MILLISECONDS.
3. В выражениях возможно использовать только одно арифметическое действие, выражения вида `NOW-5MINUTES+6SECONDS` не поддерживаются.
4. Работа с интервалами всегда приводит к получению корректной даты, но в зависимости от размерности, итоговые результаты могут быть разными:
   - При прибавлении `MONTHS` к дате происходит добавление календарного месяца, а не фиксированного числа дней. Например, если текущая дата - `2023-01-31`, то прибавление `1 MONTHS` приведет к получению даты `2023-02-28`.
   - При прибавлении `30 DAYS` к дате происходит добавление фиксированного числа дней. Например, если текущая дата - `2023-01-31`, то прибавление `30 DAYS` приведет к получению даты `2023-03-02`.
   - Минимальная возможная дата - `1970-01-01` года (время 0 в Unix time). Если в результате вычислений возникает дата меньше минимальной, то весь запрос завершается с ошибкой.
   - Максимальная возможная дата - `2105-12-31` года (максимальная дата в Unix time). Если в результате вычислений возникает дата больше максимальной, то весь запрос завершается с ошибкой.

Шаблоны путей

Данные в бакетах S3-совместимого хранилища данных могут быть размещены в каталогах с произвольными названиями. С помощью настройки `storage.location.template` можно указать правила именования каталогов, где хранятся данные.

Название поля	Описание поля	Пример значений
`storage.location.template`	Шаблон пути имен каталогов. Путь задается в виде текстовой строки с макроподстановками параметров `...${<field_name>}...${<field_name>}...`	`root/a/${year}/b/${month}/d` `${year}/${month}`

Если в пути находится знак `$`, `\` или знаки `{}`, то их необходимо экранировать с помощью символа `\`. Например, для работы с каталогом с именем `my$folder` необходимо указать путь следующим образом: `my\$folder`.

Импорт и экспорт данных с использованием федеративных запросов

Примечание

При импорте/экспорте данных из/в S3 в формате Parquet необходимо учитывать маппинг типов YQL и Apache Arrow.

Импорт данных

Федеративные запросы позволяют импортировать данные из подключенных внешних источников в таблицы базы данных YDB. Для импорта данных нужно использовать запрос чтения из внешнего источника данных или внешней таблицы и записи в YDB таблицу.

Для колоночных таблиц поддерживается массивно-параллельный импорт из внешнего источника при использовании конструкций UPSERT и INSERT — несколько рабочих потоков параллельно читают данные из внешнего источника и пишут в таблицу. Для строковых таблиц функциональность в разработке.

Операция	Запись в строковые таблицы	Запись в колоночные таблицы
UPSERT	однопоточная	параллельная
REPLACE	однопоточная	параллельная
INSERT	однопоточная	параллельная

Совет

Рекомендованными вариантами импорта с использованием федеративных запросов являются операции `UPSERT` и `REPLACE` — для них значительно оптимизирован сценарий импорта данных.

Пример импорта данных из PostgreSQL таблицы в YDB таблицу:

```
UPSERT INTO target_table
SELECT * FROM postgresql_datasource.source_table
```

Подробнее про создание внешних источников данных и внешних таблиц, а также про запросы чтения данных из них можно прочитать в статьях:

- S3

Экспериментальные источники данных

- ClickHouse
- Greenplum
- Microsoft SQL Server
- MySQL
- PostgreSQL
- YDB (федеративные запросы между базами YDB)

Экспорт данных

На данный момент экспорт данных с использованием федеративных запросов поддержан только для S3-совместимых хранилищ, подробнее смотреть в статье Экспорт данных в объектное хранилище S3.

Спиллинг

Спиллинг в общем

Спиллинг — это механизм управления памятью, при котором промежуточные данные, возникающие в результате выполнения запросов и превышающие доступный объём оперативной памяти узла, временно выгружаются во внешнее хранилище. В YDB для спиллинга в настоящее время используется диск. Спиллинг обеспечивает выполнение пользовательских запросов, которые требуют обработки больших объёмов данных, превышающих доступную память узла.

В системах обработки данных, включая YDB, спиллинг важен для:

- обработки запросов с большими объёмами данных, когда промежуточные результаты не помещаются в оперативную память;
- выполнения сложных аналитических операций (агрегации, соединения таблиц) над большими наборами данных;
- оптимизации производительности запросов за счет промежуточной материализации части данных во внешней памяти, что в определенных сценариях может ускорить общее время выполнения.

Спиллинг функционирует на основе принципа иерархии памяти:

1. **Оперативная память (RAM)** — быстрый, но ограниченный ресурс.
2. **Внешняя память** — медленнее, но более ёмкая.

Когда использование памяти приближается к лимиту, система:

- сериализует часть данных;
- сохраняет их во внешней памяти;
- освобождает соответствующую память;
- продолжает обработку запроса, используя данные, находящиеся в памяти;
- при необходимости загружает данные обратно в память для продолжения вычислений.

Спиллинг в YDB

YDB реализует механизм спиллинга через **Spilling Service** — акторный сервис, который предоставляет временное хранилище для блобов данных. Спиллинг осуществляется только на узлах базы данных. Подробная техническая информация о Spilling Service доступна в разделе Spilling Service.

Типы спиллинга в YDB

YDB реализует два основных типа спиллинга, функционирующие на различных уровнях вычислительного процесса:

- Спиллинг в вычислениях
- Спиллинг в транспорте

Эти типы работают независимо друг от друга и могут активироваться одновременно в рамках одного запроса, обеспечивая комплексное управление памятью.

Спиллинг в вычислениях

Вычислительные ядра YDB автоматически выгружают промежуточные данные на диск при выполнении операций, требующих значительного объема памяти. Данный тип спиллинга реализован на уровне отдельных вычислительных операций и активируется при достижении лимитов памяти.

Основные сценарии использования:

- **Агрегации** — при группировке больших объемов данных система выгружает промежуточные хеш-таблицы на диск
- **Join операции** — при объединении таблиц большого размера используется алгоритм Grace Hash Join с разделением данных на партиции и их выгрузкой на диск

Механизм функционирования

Вычислительные узлы содержат специализированные объекты для мониторинга использования памяти. При приближении объема данных к установленному лимиту:

1. Система переключается в режим спиллинга
2. Данные сериализуются и разделяются на блоки (бакеты)
3. Часть блоков передается в Spilling Service для сохранения на диске
4. В памяти сохраняется метаинформация о расположении данных
5. Система продолжает обработку данных, оставшихся в памяти, что позволяет освободить дополнительное место
6. При необходимости данные загружаются обратно и обрабатываются

Спиллинг в транспорте

Данный тип спиллинга функционирует на уровне передачи данных между различными этапами выполнения запроса. Подсистема передачи данных автоматически буферизует и выгружает данные при переполнении буферов. Это помогает не блокировать операции, генерирующие данные, даже в ситуациях, когда принимающая сторона временно не готова принять данные.

Механизм функционирования

Подсистема передачи данных осуществляет постоянный мониторинг своего состояния:

1. **Буферизация**: Входящие данные накапливаются во внутренних буферах подсистемы
2. **Контроль заполнения**: Система отслеживает уровень заполнения буферов
3. **Автоматический спиллинг**: При достижении лимитов данные автоматически сериализуются и передаются в Spilling Service
4. **Продолжение функционирования**: Подсистема продолжает прием новых данных после освобождения места в памяти
5. **Восстановление**: При готовности следующего этапа данные читаются из внешнего хранилища и передаются далее

Взаимодействие с контроллером памяти

При выполнении запросов YDB старается умещаться в заданный лимит памяти, который устанавливается контроллером памяти. Чтобы продолжать помещаться в этот лимит даже при росте промежуточных вычислений, используется спиллинг.

На уровне Resource Manager порог перехода к спиллингу задаётся параметром `resource_manager.spilling_percent`: когда пул памяти Query Processor на узле заполнен на указанный процент, вычислительные операции начинают предпочитать выгрузку данных на диск дальнейшему росту потребления RAM. Размер пула задаётся параметрами `query_execution_limit_percent` / `query_execution_limit_bytes`.

Подробнее о настройках спиллинга и связанных лимитах памяти см. разделы Управление памятью и Лимиты памяти выполнения запросов.

См. также

- Spilling Service
- Конфигурация спиллинга
- Мониторинг YDB
- Диагностика производительности

Скан запросы (Scan Query) в YDB

Важно

Использование `ScanQuery` не рекомендуется для новых сценариев. Для выполнения запросов используйте стандартные механизмы.

Подробнее см. в разделе Выполнение запросов.

Исключение — выполнение длительных (более 5 минут) запросов к строчным таблицам. В этом случае мы по-прежнему рекомендуем использовать `ScanQuery`, так как стандартные механизмы выполнения запросов пока не полностью поддерживают этот сценарий.

Скан запросы (Scan Queries) — это отдельный интерфейс доступа к данным, предназначенный в первую очередь для выполнения аналитических ad-hoc запросов над базой данных.

Данный способ исполнения запросов обладает следующими отличительными свойствами:

- Это только *Read-Only* запросы.
- В режиме *SERIALIZABLE_RW* берется снапшот данных, над которым в дальнейшем и происходит вся работа. В результате влияние на поток OLTP-транзакций минимальное (только снятие снапшота).
- Результат запроса — это стрим данных (gRPC stream). Таким образом, у скан запросов нет лимита на количество строк в результате.
- Из-за высоких накладных расходов подходит только для ad-hoc запросов.

Примечание

Через интерфейс *Scan Queries* можно выполнять запросы к системным представлениям.

Скан запросы не считаются полноценным способом выполнения OLAP-запросов, поскольку они обладают рядом технических ограничений (которые со временем будут сняты):

- Длительность запроса ограничена 10 минутами.
- Многие операции (включая сортировку) выполняются целиком в памяти, поэтому на сложных запросах можно получить ошибку нехватки ресурсов.
- Для соединений на текущий момент используется только одна стратегия — *MapJoin* (aka *Broadcast Join*), где «правая» таблица конвертируется в карту, поэтому она должна быть не более единиц гигабайт.
- Не поддерживается prepared-форма, т.е. на каждый вызов происходит компиляция запроса.
- Нет оптимизаций под точечные чтения и чтения небольших диапазонов данных.
- В SDK не поддерживается автоматический retry.

Для работы с OLAP-нагрузками в YDB существует специализированный тип таблиц — колоночные таблицы. Они хранят данные каждого столбца отдельно от других столбцов. Благодаря этому при выполнении запроса считываются только те столбцы, которые непосредственно участвуют в запросе.

Примечание

Несмотря на то, что *Scan Queries* явно не мешают выполнению OLTP-транзакций, они все же используют общие ресурсы базы: CPU, память, диск, сеть. Поэтому выполнение тяжелых запросов **может привести к голоданию по ресурсам**, что скажется на производительности всей базы.

Как воспользоваться

Как и другие виды запросов, *Scan Queries* доступны через CLI и SDK.

C++ SDK

Для запуска запроса через механизм *Scan Queries* предназначены 2 метода в классе `Ydb::TTableClient`:

```
class TTableClient {
 ...
 TAsyncScanQueryPartIterator StreamExecuteScanQuery(const TString& query,
 const TStreamExecScanQuerySettings& settings = TStreamExecScanQuerySettings());

 TAsyncScanQueryPartIterator StreamExecuteScanQuery(const TString& query, const TParams& params,
 const TStreamExecScanQuerySettings& settings = TStreamExecScanQuerySettings());
 ...
};
```

YQL-запросы к топикам

Для чтения и записи сообщений в топики используются привычные YQL-конструкции: SELECT для чтения и INSERT для записи.

Локальные и внешние топики

YQL-запросы к топикам работают одинаково независимо от того, находится топик в текущей базе или в другой базе YDB. Источником и приёмником сообщений может быть как топик **в той же базе данных**, в которой выполняется запрос, так и топик **в другой базе**.

Локальные топики

Локальные топики — топики, созданные в **той же базе** YDB, что и выполняемый запрос.

В тексте запроса к ним обращаются **по короткому имени** — так же, как к таблице в текущей базе:

```
SELECT * FROM input_topic WITH (FORMAT = json_each_row, SCHEMA = (...));
```

```
INSERT INTO output_topic SELECT ...;
```

Внешние топики

Внешние топики — топики, расположенные **в другой базе** YDB.

Доступ к ним выполняется только через заранее созданный внешний источник данных с типом источника YDB.

После создания источника, например с именем `ext_source`, обращение к топику `input_topic` во внешней базе записывается так:

```
SELECT * FROM ext_source.input_topic WITH (FORMAT = json_each_row, SCHEMA = (...));
```

Имя `ext_source` в документации **условное** — в вашей базе источник может называться иначе; важно, чтобы оно совпадало в `CREATE EXTERNAL DATA SOURCE` и в префиксе перед именем топика.

Чтение из топика

Чтение из топика может ограничиваться только текущими данными топика или работать с ожиданием новых записанных сообщений.

Чтение текущих данных

В данном режиме чтение выполняется от первого до последнего смещения, хранящегося в топике на момент запуска запроса. Если в топик продолжается запись данных, то запрос остановится после достижения последнего смещения, известного на момент запуска. Указание фильтров по Служебным полям ускоряет чтение, так как чтение происходит только по указанным диапазонам.

```
SELECT
    Data    -- тело сообщения
FROM
    input_topic  -- локальный топик; для внешнего: ext_source.input_topic
LIMIT 10;
```

Чтение с ожиданием данных

Для ожидания новых сообщений используйте опцию `WITH (STREAMING = "TRUE")`. Чтение начинается с текущего момента и продолжается до тех пор, пока не будет прочитано заданное в выражении `LIMIT` количество сообщений. Параметр `LIMIT` обязателен — без него запрос не завершится, так как будет ожидать новые сообщения бесконечно.

```
SELECT
    Data
FROM
    ext_source.input_topic  -- внешний топик; для локального: input_topic
WITH (STREAMING = "TRUE")
LIMIT 10;
```

Для непрерывной обработки поступающих данных используйте потоковые запросы.

Формат и схема сообщений

При чтении из топика тело сообщения можно получить двумя способами: сырые данные и форматированные данные.

Сырые данные

Используйте, когда содержимое сообщения не нужно разбирать — достаточно прочитать тело как есть.

```
SELECT
    Data
FROM
    input_topic  -- локальный топик; для внешнего: ext_source.input_topic
WITH (
    FORMAT = raw,
    SCHEMA = (
        Data String
    )
)
LIMIT 10;
```

В результате доступна только колонка `Data` — тело сообщения в исходном виде.

Тот же результат можно получить без блока `WITH` — см. табличное чтение.

Форматированные данные

Используйте, когда сообщения сериализованы в известном формате (JSON, CSV и др.). Параметр `FORMAT` задаёт способ разбора, а `SCHEMA` — имена и типы полей, которые появятся в результате `SELECT`:

```
SELECT
    Id,
    Name
FROM
    input_topic  -- локальный топик; для внешнего: ext_source.input_topic
WITH (
    FORMAT = json_each_row,
    SCHEMA = (
        Id Uint64 NOT NULL,
        Name Utf8 NOT NULL
    )
);
```

Поля из `SCHEMA` доступны в `SELECT` по имени — как колонки таблицы.

Подробнее о поддерживаемых форматах: Форматы данных при чтении/записи из топиков.

Использование читателя

Читатель (consumer) — именованная подписка на топик, которая хранит текущую позицию чтения.

Читатель создаётся через CLI или при создании топика с помощью CREATE TOPIC. Имя читателя указывается в тексте запроса прагмой:

```
PRAGMA pq.Consumer="my_consumer";
```

Если читатель не указан, чтение из топика выполняется без него. Указание читателя позволяет отслеживать позицию чтения и лаг со стороны топика, например через CLI.

Перенос данных из топика в таблицу через UPSERT

Данные из топика можно переложить в таблицу через UPSERT INTO:

```
UPSERT INTO
    table_name
SELECT
    Data            -- можно использовать любые преобразования
FROM
    ext_source.input_topic;  -- внешний топик; для локального: input_topic
```

Служебные поля

При чтении можно запрашивать служебные поля и пользовательские атрибуты сообщений:

Поле	Тип	Описание
`__ydb_create_time`	`Timestamp`	Время создания сообщения
`__ydb_write_time`	`Timestamp`	Время записи сообщения в топик
`__ydb_offset`	`Uint64`	Смещение сообщения в партиции
`__ydb_partition_id`	`Uint64`	Номер партиции
`__ydb_message_group_id`	`String`	Идентификатор группы сообщений
`__ydb_seq_no`	`Uint64`	Порядковый номер сообщения внутри группы
`__ydb_user_attributes`	`Dict<String,String>`	Пользовательские атрибуты сообщения

Пример использования служебных полей:

```
SELECT
    Data,                                           -- тело сообщения
    __ydb_create_time AS CreateTime,                -- время создания сообщения
    __ydb_write_time AS WriteTime,                  -- время записи сообщения
    __ydb_offset AS Offset,                         -- смещение сообщения в топике
    __ydb_partition_id AS Partition,                -- номер партиции
    __ydb_message_group_id AS MessageGroupId,       -- идентификатор группы сообщений
    __ydb_seq_no AS SeqNo                           -- порядковый номер внутри партиции
FROM
    input_topic  -- локальный топик; для внешнего: ext_source.input_topic
LIMIT 10;
```

Фильтры по служебным полям вычисляются до чтения данных из топика и существенно сокращают объём считываемых сообщений. Поддерживаются операторы сравнения (`=` , `<>` , `<` , `<=` , `>` , `>=` , `IN`), логические условия (`AND` , `OR`) и поля `partition_id` , `write_time` , `offset` . Предикаты по остальным служебным полям не ограничивают объём чтения.

```
SELECT
    Data
FROM
    ext_source.input_topic  -- внешний топик; для локального: input_topic
WHERE
    __ydb_partition_id = 42
        AND __ydb_offset >= 1000
        AND __ydb_offset <= 1100
        AND __ydb_write_time > CurrentUtcTimestamp() - Interval("PT2H");
```

Пример использования пользовательских атрибутов:

```
SELECT
    COUNT(*) AS ErrorCount
FROM
    input_topic  -- локальный топик; для внешнего: ext_source.input_topic
WHERE
    __ydb_user_attributes["type"] = "log"
        AND __ydb_user_attributes["level"] = "error";
```

Запись в топик

Запись одного сообщения

```
INSERT INTO
    output_topic  -- локальный топик; для внешнего: ext_source.output_topic
SELECT
    "my_message";       -- тело сообщения
```

Запись данных из таблицы

Чтобы записать в топик строку таблицы с несколькими колонками, сформируйте JSON-объект. Функция `TableRow` создаёт структуру из всех колонок, `Yson::From` преобразует её в Yson, `Yson::SerializeJson` сериализует в JSON-строку, а `ToBytes` конвертирует результат в тип `String`, который требуется для записи в топик:

```
INSERT INTO
    ext_source.output_topic  -- внешний топик; для локального: output_topic
SELECT
    ToBytes(Unwrap(Yson::SerializeJson(Yson::From(TableRow()))))
FROM
    table_name;
```

Ограничения

Важно

Запись пользовательских атрибутов через YQL не поддерживается.

Важно

Транзакционная запись через YQL/`INSERT INTO` не поддерживается — в топике могут появиться частичные результаты запроса.

См. также

- CREATE TOPIC
- ALTER TOPIC
- DROP TOPIC
- Потоковые запросы

Транзакции и запросы к YDB

Этот раздел описывает особенности реализации YQL для YDB транзакций.

Язык запросов

Основным средством создания, модификации и управления данными в YDB является декларативный язык запросов YQL. YQL — это диалект SQL, который может считаться стандартом для общения с базами данных. Кроме того, YDB поддерживает набор специальных RPC, например, для работы с древовидной схемой или для управления кластером.

Режимы транзакций

YDB поддерживает несколько режимов выполнения транзакций. По умолчанию транзакции выполняются в режиме *Serializable*, который предоставляет самый строгий уровень изоляции для пользовательских транзакций. Режим выполнения транзакции задаётся в настройках при её создании. Примеры для YDB SDK см. в Установка режима выполнения транзакции. Во YDB UI режим транзакции также можно выбрать в настройках.

Поддерживаются **режимы чтения-записи**: *Serializable* (по умолчанию) и *Snapshot Read-Write*; **режимы только для чтения**: *Snapshot Read-Only* и *Stale Read-Only*. Режим *Online Read-Only* оставлен для совместимости со старым кодом (**legacy**); в новых сценариях вместо него следует использовать *Snapshot Read-Only*.

Serializable

Суть. Сериализуемое исполнение пользовательских транзакций (Serializable уровень изоляции).

Особенности. Используется механизм Optimistic Concurrency Control. На прочитанные в ходе выполнения транзакции строки ставятся оптимистичные блокировки. При завершении транзакции проверяется, что блокировки не были инвалидированы. Оптимистичность блокировок выливается в важное для пользователя свойство — в случае конфликта выигрывает транзакция, которая завершается первой. Конкурирующие транзакции завершатся с ошибкой `Transaction locks invalidated`.

Гарантии.

- Результат успешно выполненных параллельных транзакций эквивалентен некоторому последовательному порядку их выполнения;
- Для успешных транзакций нет аномалий чтений;
- Транзакция видит все изменения, которые были зафиксированы (закоммичены) до её первого чтения (времени взятия снапшота), плюс собственные изменения, сделанные ранее в этой же транзакции (read-your-own-writes);
- Гарантируется линеаризуемость **по ключу**: если транзакция T1, затрагивающая некоторый ключ, завершилась до начала транзакции T2, затрагивающей **тот же ключ**, то в порядке выполнения T1 будет находиться перед T2. Для транзакций, работающих с разными ключами, такой порядок не гарантируется — наблюдаемый порядок их фиксации может отличаться от реального порядка их завершения во времени.

Snapshot Read-Write

Суть. Представляет собой изоляцию Snapshot Isolation (аналог Repeatable Read в PostgreSQL).
Чтения производятся из согласованного снимка данных, зафиксированного до момента выполнения первого чтения. Транзакция успешно зафиксируется только при условии, что от момента взятия снимка данных до момента коммита транзакции модифицированные ей строки не были модифицированы другими транзакциями.

Особенности. Используется механизм Optimistic Concurrency Control. В отличие от Serializable, транзакции не берут блокировки на читаемые строки — это позволяет им успешно зафиксироваться, даже если прочитанные ими данные были модифицированы другими транзакциями. На модифицируемые же строки берутся блокировки: при параллельных транзакциях, изменяющих те же ключи, только транзакция, которая завершается первой, завершится успешно. Конкурирующие транзакции будут отклонены с write-write конфликтом на этапе фиксации с ошибкой `Transaction locks invalidated`.

Гарантии.

- Все чтения в транзакции видят одно и то же состояние данных на снимке, полученном до момента выполнения первого чтения, плюс собственные изменения, сделанные ранее в этой же транзакции (read-your-own-writes);
- При наличии write-write конфликта транзакция не сможет зафиксироваться;
- Может наблюдаться аномалия write skew.

Snapshot Read-Only

Суть. Транзакция работает с целостным снимком базы, зафиксированным до момента выполнения первого чтения. Гарантии на чтение данных такие же, как Snapshot Read-Write. Запись в данном режиме запрещена, за счет этого режим эффективнее, чем Snapshot Read-Write, когда необходимо только чтение данных.

Особенности. Обеспечивает максимальную актуальность данных на момент начала транзакции, но может иметь более высокую задержку ответа из-за необходимости формирования снимка.

Гарантии. Все чтения в транзакции видят одно и то же состояние данных на снимке; коммиты после момента взятия снапшота не видны.

Stale Read-Only

Суть. Чтения выполняются на репликах таблетки (шарда) с возможным отставанием от лидера таблетки (шарда) (обычно доли секунды). Режим хорошо подходит для сценария чтения по ключу, когда нужны минимальные задержки. Пользователь читает закоммиченные, но возможно устаревшие данные.

Особенности. Низкая задержка и высокая пропускная способность за счёт чтения с реплик. Реплика для чтения выбирается обычно в той же зоне доступности, но точное расположение не гарантируется.

Гарантии. Согласованность данных на уровне ключа внутри **одного** `SELECT` выражения; между **разными** `SELECT` выражениями в одной транзакции согласованность **не** гарантируется.

Ограничения. Нет единого снимка на всю транзакцию; возможна задержка относительно данных на лидере (данные могут быть не самыми свежими). Использование реплик для чтений возможно только в случае, если все читаемые строки находятся в одном шарде. Если же чтение затрагивает несколько шардов, то чтение будет произведено из лидеров со взятием снимка аналогично Snapshot Read-Only. Чтение из колоночных таблиц в этом режиме не поддерживается (см. предупреждение ниже). Не поддерживаются интерактивные транзакции. Дополнительные ограничения и нюансы по типам запросов см. в документации SDK для выбранного языка.

Online Read-Only

Устаревший (**legacy**) режим, сохранённый для совместимости. Для новых приложений при чтении без записи используйте *Snapshot Read-Only*. Подробности и примеры вызова в SDK по-прежнему можно найти в Online Read-Only.

> **Ограничение для Online Read-Only и Stale Read-Only**
>
> Эти режимы не поддерживают чтение из колоночных таблиц. Попытка чтения вызовет ошибку следующего вида:
>
> ```
> Read from column tables is not supported in Online Read-Only or
> Stale Read-Only transaction modes. Use Serializable or
> Snapshot Read-Only mode instead.
> ```
>
> Для транзакций с чтением из колоночных таблиц используйте:
>
> - Serializable — режим по умолчанию;
> - Snapshot Read-Only — режим чтения из согласованного снимка.

Неявные транзакции

Логика неявных транзакций применяется при отправке одного YQL-скрипта на сервер без явного выбора режима транзакции. Типичные точки входа:

- YDB UI — вкладка **Query** на странице базы данных (форма выполнения запроса), при запуске без выбора режима транзакции в настройках.
- YDB CLI — разовая отправка скрипта командой `ydb sql`.
- Приложения на YDB SDK — режим ImplicitTx.

Если для запроса не задан режим транзакции, YDB автоматически управляет его поведением. Такой режим называется **неявной транзакцией**.

В этом режиме YDB на основе запроса определяет, выполнить его вне транзакции или обернуть в транзакцию с режимом *Serializable*. Режим неявной транзакции является универсальным для выполнения запроса, так как поддерживает инструкции любого вида с определённым поведением, описанным ниже.

Поведение для разных видов инструкций

- **Инструкции Data Definition Language (DDL)**
  DDL-инструкции (такие как CREATE TABLE, DROP TABLE и т.д.) выполняются вне транзакции. Запрос может состоять только из DDL-инструкций. При возникновении ошибки изменения, внесённые предыдущими инструкциями запроса, не откатываются.
- **Инструкции Data Manipulation Language (DML)**
  DML-инструкции (такие как UPSERT, SELECT, UPDATE и т.д.) оборачиваются в транзакцию с режимом *Serializable*. Запрос может состоять только из DML-инструкций. При успешном выполнении изменения коммитятся (фиксируются), а при возникновении ошибки — откатываются.
- **Инструкции пакетного изменения**
  Инструкции пакетного изменения (такие как BATCH UPDATE и BATCH DELETE FROM) выполняются вне транзакции. Запрос может состоять только из одной инструкции пакетного изменения. При возникновении ошибки изменения инструкции не откатываются.

Сводная таблица

Тип инструкции	Обработка неявной транзакции	Поддержка нескольких инструкций	Откат при ошибке
DDL	Вне транзакции	Да (только DDL)	Нет
DML	Автоматическая транзакция (Serializable)	Да (только DML)	Да
Инструкции пакетного изменения	Вне транзакции	Нет	Нет

Чтобы явно задать режим транзакции, используйте соответствующие настройки в каждой точке входа:

- YDB UI — выберите режим транзакции в настройках выполнения на вкладке **Query**.
- YDB CLI — у подкоманды `table query execute` для запросов типа `data` задайте параметр `--tx-mode` (по умолчанию `serializable-rw`, что соответствует режиму *Serializable*).
- YDB SDK — см. установку режима в YDB SDK.

Язык YQL

Реализованные конструкции YQL можно разделить на два класса: data definition language (DDL) и data manipulation language (DML).

Подробнее о поддерживаемых конструкциях YQL можно почитать в документации YQL.

Ниже перечислены возможности и ограничения поддержки YQL в YDB, на которые стоит обратить внимание:

- Допускаются multistatement transactions, то есть транзакции, состоящие из последовательности выражений YQL. При выполнении транзакции допускается взаимодействие с клиентской программой, иначе говоря, взаимодействие клиента с базой может выглядеть следующим образом: `начать транзакцию и выполнить SELECT; проанализировать результаты SELECT на клиенте; ...; выполнить UPDATE и коммит транзакции`. Каждый из запросов внутри транзакции может также содержать несколько выражений YQL. Стоит отметить, что если тело транзакции полностью сформировано до обращения к базе данных, то транзакция может обрабатываться эффективнее;
- В YDB не поддерживается возможность смешивать DDL и DML запросы в одной транзакции. Традиционное понятие ACID транзакции применимо именно к DML запросам, то есть к запросам, которые меняют данные. DDL запросы должны быть идемпотентными, то есть повторяемы в случае ошибки. Если необходимо выполнить действие со схемой, то каждое из действий будет транзакционно, а набор действий — нет;

- Любые ошибки инвалидируют всю транзакцию целиком, а не отдельный запрос, поэтому после завершения транзакции с ошибкой, сообщающей о временном сбое, выполнение транзакции необходимо повторять с самого начала;
- Чтения в транзакции видят все изменения данных, которые были выполнены в той же транзакции ранее;
- Все изменения, производимые в рамках транзакции, накапливаются в памяти сервера базы данных. Они не видны другим транзакциям до момента успешного завершения текущей и применяются атомарно при коммите. Описанная схема накладывает ограничение: объём изменений в рамках одной транзакции должен умещаться в оперативную память. **Деталь реализации:** если транзакция читает данные из таблицы, которую она ранее модифицировала, накопленные изменения записываются в шарды досрочно — это влияет на эффективность (см. рекомендацию ниже). Досрочно записанные данные не видны другим транзакциям и откатываются при отмене транзакции;
- Для эффективности транзакций избегайте чтения из таблиц, ранее модифицированных в этой же транзакции (read-after-write), так как это приводит к преждевременной записи данных в шарды. Для каждой таблицы выполняйте все чтения раньше модификаций.

Подробнее о поддержке YQL в YDB можно прочитать в документации YQL.

Распределенные транзакции

Таблица в YDB может быть шардирована по диапазонам значений первичного ключа. Различные шарды таблицы могут обслуживаться разными серверами распределенной БД (в том числе расположенными в разных локациях), а также могут независимо друг от друга перемещаться между серверами для перебалансировки или поддержания работоспособности шарда при отказах серверов или сетевого оборудования.

Топик в YDB может быть шардирован на несколько партиций. Различные партиции топика, как и шарды таблицы, могут обслуживаться разными серверами распределенной БД.

В YDB поддерживаются распределенные транзакции. Распределенные транзакции — это транзакции, которые затрагивают более одного шарда одной или нескольких таблиц и топиков. Они требуют больше ресурсов и выполняются дольше. В то время как точечные чтения и записи могут выполняться за время до 10 мс в 99 перцентиле, распределенные транзакции, как правило, занимают от 20 до 500 мс.

Транзакции с участием топиков и таблиц

Важно

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

YDB поддерживает транзакции с участием строковых таблиц и/или топиков. Таким образом, можно транзакционно перекладывать данные из таблиц в топики и в обратном направлении, а также между топиками, чтобы данные не терялись и не дублировались даже в случае непредвиденных обстоятельств.

Подробнее о транзакционных операциях при работе с топиками см. в Транзакции с участием топиков и Работа с топиками.

Транзакции с участием строковых и колоночных таблиц

В настоящее время одновременное использование колоночных и строковых таблиц поддерживается в транзакциях, в которых данные только читаются, но не изменяются. Поддержка транзакций с возможностью модификации данных при одновременном использовании строковых и колоночных таблиц находится в разработке.

Если попытаться выполнить операцию записи в транзакции, в которой задействованы и колоночные, и строковые таблицы, транзакция завершится с ошибкой:

```
Write transactions that use both row-oriented and column-oriented tables are disabled at current time
```

Вторичные индексы

В YDB автоматически создается индекс по первичному ключу, поэтому выборки с условием по первичному ключу всегда выполняются эффективно, затрагивая только требуемые строки. Выборка с условием, наложенным на одну или несколько неключевых колонок, как правило, приводит к полному сканированию таблицы. Для того чтобы такие выборки были эффективными, нужно использовать *вторичные индексы* — глобальные структуры с отдельной индексной таблицей.

Отдельно существуют локальные индексы: вспомогательные структуры, которые хранятся вместе с данными таблицы и применяются при чтении на стороне хранилища, не материализуя отдельную индексную таблицу (см. раздел Локальные индексы ниже).

В текущей версии YDB реализованы *синхронные* и *асинхронные* глобальные вторичные индексы. Каждый индекс представляет собой скрытую таблицу, которая обновляется:

- для синхронных индексов — транзакционно при изменении основной таблицы;
- для асинхронных индексов — в фоне, получая необходимые изменения из основной таблицы.

Когда пользователь присылает SQL-запрос на вставку, изменение или удаление данных, БД прозрачно для пользователя формирует команды на изменение индексной таблицы. У таблицы может быть несколько вторичных индексов. Индекс может включать несколько колонок, при этом важен порядок указания колонок в индексе. Одна колонка может быть включена в нескольких индексов. В дополнение к указанным колонкам в индексе всегда неявно сохраняются значения колонок первичного ключа таблицы, чтобы от найденной записи в индексе можно было перейти к записи в таблице.

Синхронный вторичный индекс

Синхронный индекс обновляется одновременно с таблицей, которую он индексирует. Такой индекс обеспечивает строгую согласованность данных и использует для этого механизм распределенных транзакций. Таким образом, если операции чтения и слепой записи в таблицу без индекса можно выполнить без стадии планирования, и за счет этого существенно сократить задержки, то для записи в таблицу с синхронным индексом такая оптимизация невозможна.

Асинхронный вторичный индекс

Асинхронный индекс, в отличие от синхронного, не использует механизм распределенных транзакций, а в фоне получает изменения из индексируемой таблицы. Транзакции записи в таблицу с таким индексом выполняются без дополнительных затрат на планирование, за счет снижения гарантий: асинхронный индекс обеспечивает согласованность данных в конечном счете, но не строгую согласованность. Использование асинхронного индекса в транзакциях чтения возможно только в режиме Stale Read Only.

Покрывающий вторичный индекс

Имеется возможность сделать копию содержимого колонок в индекс (covering index), таким образом это исключает необходимость чтения из основной таблицы в операциях чтения по индексу, что заметно снижает задержки. В то же время такая денормализация приводит к увеличенному потреблению дискового пространства и к возможному замедлению операции вставки и обновления из-за необходимости дополнительного копирования данных.

Уникальный вторичный индекс

Этот тип индекса реализует семантику уникального значения в колонке или наборе колонок, а также, как и другие индексы, позволяет эффективно выполнять точечные чтения по набору индексируемых колонок. YDB использует его для выполнения дополнительных проверок, чтобы гарантировать, что каждое уникальное значение индексируемых колонок присутствует в таблице не более одного раза. Если модифицирующий запрос нарушает это ограничение, он отменяется со статусом `PRECONDITION_FAILED`. Таким образом, пользовательский код должен быть готов к обработке этого статуса.

Уникальный вторичный индекс является синхронным индексом, поэтому с точки зрения транзакционности процесс его обновления такой же, как у синхронного вторичного индекса, описанного выше.

Векторный индекс

Векторный индекс является особым видом вторичного индекса.

В отличие от традиционных вторичных индексов, оптимизирующих поиск по равенству или диапазону, векторные индексы позволяют выполнять векторный поиск на основе функций расстояния или схожести.

Полнотекстовый индекс

Полнотекстовый индекс является особым видом вторичного индекса.

В отличие от традиционных вторичных индексов, оптимизирующих поиск по равенству или диапазону, полнотекстовые индексы позволяют масштабируемо выполнять текстовый поиск по словам и их сочетаниям (а при использовании N-грамм — и по подстрокам). См. также: Полнотекстовый поиск.

JSON-индекс

JSON-индекс является особым видом вторичного индекса, как и полнотекстовый — оба построены поверх инвертированного индекса, но используют разные токенизаторы.

JSON-индексы позволяют ускорять предикаты с функциями JSON_EXISTS и JSON_VALUE по содержимому колонки типа `Json` или `JsonDocument`. Индекс строится с помощью разбиения JSON-документов на токены путей и пар вида «путь + значение», что позволяет находить соответствующие строки по путям JsonPath без полного сканирования таблицы. См. также: Поиск по JSON.

Локальные индексы

Локальные индексы — вспомогательные структуры, хранящиеся вместе с данными таблицы и применяемые при чтении на стороне хранилища. Они не материализуют отдельную индексную таблицу. Сейчас реализованы Блум-индексы и min_max-индекс.

Онлайн-создание вторичного индекса

В YDB доступно создание вторичного индекса, а также удаление существующего вторичного индекса без остановки обслуживания. Для одной таблицы можно создать только один индекс за раз.

Операция онлайн-создания индекса состоит из следующих шагов:

1. Взятие снапшота таблицы с данными, создание таблицы индекса с пометкой доступности для записи.

После этого шага пишущие транзакции становятся распределенными, происходит запись в основную таблицу и индекс. Для пользователя индекс пока не доступен.

2. Чтение снапшота основной таблицы и запись в индекс.

   Реализуется «запись в прошлое»: разрешаются ситуации, когда обновления данных на шаге 1 меняют данные, записанные на шаге 2.

3. Публикация результата, удаление снапшота.

   Индекс готов к использованию.

Возможное влияние на пользовательские транзакции:

- Может наблюдаться увеличение задержек из-за того, что транзакции становятся распределенными (при создании синхронного индекса).
- Возможен повышенный фон ошибок `OVERLOADED` из-за того, что во время записи данных активно работает автоматическое разделение шардов индексной таблицы.

> **Примечание**
>
> Скорость записи данных выбрана таким образом, чтобы минимизировать влияние процесса записи на пользовательские транзакции. Для управления скоростью настройте лимиты для соответствующей очереди брокера ресурсов.

Создание индекса — асинхронная операция. Если после запуска операции произойдет разрыв клиент-серверной связности, то построение индекса будет продолжено. Управлять асинхронной операцией можно через YDB CLI.

Создание и удаление вторичных индексов

Вторичный индекс может быть:

- Создан при создании таблицы командой YQL CREATE TABLE.
- Добавлен к существующей таблице командой YQL ALTER TABLE или командой YDB CLI table index add
- Удален у существующей таблицы командой YQL ALTER TABLE или командой YDB CLI table index drop.
- Удален вместе с таблицей командой YQL DROP TABLE или командой YDB CLI `table drop`.

Применение вторичных индексов

Детальная информация о применении вторичных индексов в приложениях находится в статье о них в разделе документации для разработчиков.

Change Data Capture (CDC)

Важно

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Change Data Capture (CDC) обеспечивает захват изменений строк строковой таблицы YDB, формирует из них *поток изменений (changefeed)*, записывает в распределенное хранилище и предоставляет доступ к этим записям для дальнейшей обработки. В качестве распределенного хранилища используется топик, который позволяет эффективно хранить лог изменений таблицы.

Когда в таблице добавляется, обновляется или удаляется строка, CDC формирует запись о произошедшем изменении с указанием первичного ключа строки и пишет ее в соответствующую данному ключу партицию топика.

Гарантии

- Записи об изменениях шардированы между партициями топика по первичному ключу.
- Каждое изменение доставляется ровно один раз (exactly-once семантика).
- Изменения по одному и тому же первичному ключу доставляются в том же порядке, в котором они происходили в таблице в одну и ту же партицию топика.
- Запись об изменении доставляется до партиции топика только после фиксации (коммита) соответствующей транзакции в таблице.

Ограничения

- В потоке изменений поддерживаются записи о следующих видах операций:
  - Обновление — перезапись значений указанных столбцов. Пример запроса: UPDATE.
  - Замена — перезапись значений указанных столбцов, значения неуказанных столбцов заменяются на значения по умолчанию. Пример запроса: REPLACE INTO.
  - Удаление. Пример запроса: DELETE FROM.

Добавление строки является частным случаем обновления или замены, и в потоке изменений запись о добавлении строки будет выглядеть аналогично записи об обновлении или замене, в зависимости от исходного запроса, приведшего к изменению.

Виртуальные метки времени

Все изменения в таблицах YDB упорядочены в соответствии с порядком выполнения транзакций. Каждое изменение маркируется виртуальной меткой времени, являющейся кортежем из двух элементов:

1. Глобального времени координатора.
2. Уникального идентификатора транзакции.

Используя эти метки, можно упорядочить записи из разных партиций топика относительного друг друга или использовать их для фильтрации (например, чтобы исключить записи о старых изменениях).

Для одношардовых транзакций может быть использована метка времени существующей распределённой транзакции в очереди выполнения, что позволяет метке времени быть неуникальной. Если в очереди выполнения нет распределённых транзакций, то первым элементом метки времени будет время, полученное от координатора, а уникальный идентификатор транзакции примет максимальное возможное значение `18446744073709551615` (2^{64}-1, максимальное значение `Uint64`).

Примечание

По умолчанию виртуальные метки времени не выгружаются в поток изменений. Для их включения используйте соответствующий параметр при создании потока.

Барьеры

Барьеры — служебные записи без данных об изменении или удалении с виртуальными метками времени, которые появляются в каждой партиции топика с заданной периодичностью. Барьер — это гарантия, что любое изменение с виртуальной меткой времени меньшей, чем у барьера, было записано в данную партицию топика.

Барьеры могут быть использованы для обеспечения строгой упорядоченности и глобальной согласованности данных путём буферизации данных между ними.

Примечание

По умолчанию барьеры не выгружаются в поток изменений. Для настройки периодичности выгрузки барьеров используйте соответствующий параметр при создании потока.

Первоначальное сканирование таблицы

По умолчанию в поток изменений выгружаются записи только о тех строках таблицы, которые изменились после создания потока. Первоначальное сканирование таблицы позволяет выгрузить в поток изменений значения всех строк, существовавших на момент его создания.

Сканирование производится в фоновом режиме поверх снапшота таблицы. При этом возможны следующие ситуации:

- В таблице меняется значение непросканированной строки. В поток изменений последовательно будут выгружены: запись с исходным значением и запись об изменении. При повторном изменении этой же строки будет выгружена только запись об изменении.
- Во время сканирования обнаруживается измененная строка. В поток изменений ничего не выгружается, так как исходное значение уже было выгружено в момент изменения (см. предыдущий пункт).
- В таблице меняется значение просканированной строки. В поток изменений выгружается только запись об изменении.

Таким образом, гарантируется, что для одной и той же строки (первичного ключа) сначала будет выгружено исходное значение, а затем — запись об изменении.

> **Примечание**
>
> Запись с исходным значением строки будет помечена как запись об обновлении. При использовании виртуальных меток времени записи маркируются меткой времени снапшота.

В процессе сканирования, в зависимости от частоты обновления данных таблицы, возможен повышенный фон ошибок `OVERLOADED` из-за того, что, помимо записей об изменениях, необходимо доставить также записи с исходными значениями строк. По окончании сканирования поток изменений переходит в нормальный режим работы.

> **Важно**
>
> На время первоначального сканирования в таблице приостанавливаются процессы автоматического партиционирования и в топик не выгружаются барьеры.

Структура записи

В зависимости от параметров потока структура записи может отличаться.

JSON-формат

Запись в формате JSON имеет следующую структуру:

```
{
    "key": [<key components>],
    "update": {<columns>},
    "reset": {<columns>},
    "erase": {},
    "newImage": {<columns>},
    "oldImage": {<columns>},
    "ts": [<step>, <txId>],
    "user": "<user SID>",
    "traceId": "<Trace ID>"
}
```

- `key` — массив значений компонент первичного ключа. Присутствует всегда. Порядок элементов соответствует порядку столбцов в первичном ключе таблицы.
- `update` — признак обновления. Присутствует, если запись соответствует операции обновления. В режиме `UPDATES` так же содержит названия и значения изменившихся столбцов.
- `reset` — признак замены. Присутствует, если запись соответствует операции замены. В режиме `UPDATES` так же содержит названия и значения столбцов, для которых задано значение.
- `erase` — признак удаления. Присутствует, если запись соответствует операции удаления.
- `newImage` — снимок состояния строки, получившегося в результате изменения. Присутствует в режимах `NEW_IMAGE` и `NEW_AND_OLD_IMAGES`. Содержит названия и значения столбцов.
- `oldImage` — снимок состояния строки, предшествовавшего изменению. Присутствует в режимах `OLD_IMAGE` и `NEW_AND_OLD_IMAGES`. Содержит названия и значения столбцов.
- `ts` — виртуальная метка времени. Присутствует, если включена настройка `VIRTUAL_TIMESTAMPS`. Содержит значение глобального времени координатора (`step`) и уникальный идентификатор транзакции (`txId`).
- `user` — идентификатор пользователя. Присутствует, если включена настройка `USER_SIDS`. Содержит SID пользователя. Если происходит удаление записей по TTL, то указывается значение `ttl@system` .
- `traceId` — идентификатор трассировки OpenTelemetry. Присутствует, если включена настройка `TRACE_IDS` .

Например, запись об обновлении в режиме `UPDATES` :

```
{
    "key": [1, "one"],
    "update": {
        "payload": "lorem ipsum",
        "date": "2022-02-22"
    }
}
```

Запись об удалении:

```
{
    "key": [2, "two"],
    "erase": {}
}
```

Запись со снимками строки:

```
{
    "key": [1, 2, 3],
    "update": {},
    "newImage": {
        "textColumn": "value1",
        "intColumn": 101,
        "boolColumn": true
    },
    "oldImage": {
```

```
        "textColumn": null,
        "intColumn": 100,
        "boolColumn": false
    }
}
```

Запись с виртуальными метками времени:

```
{
    "key": [1],
    "update": {
        "created": "2022-12-12T00:00:00.000000Z",
        "customer": "Name123"
    },
    "ts": [1670792400890, 562949953607163]
}
```

Запись с барьером содержит единственное поле `resolved` с виртуальной меткой времени:

```
{
    "resolved": [1670792500000, 0]
}
```

Примечание

- Одна и та же запись не может содержать поля `update`, `reset` и `erase` одновременно, так как эти поля являются признаками операции (невозможно одновременно обновить и удалить строку таблицы). Но каждая запись содержит одно из этих полей (любая операция является обновлением, заменой или удалением).
- В режиме `UPDATES` для операций обновления или замены поля `update` и `reset` выполняют роль не только признака операции, но и содержат названия и значения изменившихся столбцов.
- Поля JSON-объекта, содержащие названия и значения столбцов (`newImage`, `oldImage`, `update` и `reset` в режиме `UPDATES`), *не включают* в себя столбцы, являющиеся компонентами первичного ключа.
- Если в записи присутствует поле `erase` (то есть запись соответствует операции удаления), то это всегда пустой JSON-объект (`{}`).

JSON-формат, совместимый с Debezium

Запись в формате JSON, совместимого с Debezium, имеет следующую структуру:

```
{
    "payload": {
        "op": <op>,
        "before": {<columns>},
        "after": {<columns>},
        "source": {
            "connector": <connector>,
            "version": <version>,
            "ts_ms": <ts_ms>,
            "step": <step>,
            "txId": <txId>,
            "snapshot": <bool>,
            "user": <user SID>,
            "traceId": <Trace ID>
        }
    }
}
```

- `op` — операция, которая была произведена над строкой в таблице:
  - `c` — вставка. Допустимо только в режиме `NEW_AND_OLD_IMAGES`.
  - `u` — обновление.
  - `d` — удаление.
  - `r` — чтение из снапшота.
- `before` — снимок состояния строки, предшествовавшего изменению. Присутствует в режимах `OLD_IMAGE` и `NEW_AND_OLD_IMAGES`. Содержит названия и значения столбцов.
- `after` — снимок состояния строки, получившегося в результате изменения. Присутствует в режимах `NEW_IMAGE` и `NEW_AND_OLD_IMAGES`. Содержит названия и значения столбцов.
- `source` — метаданные записи.
  - `connector` — название коннектора. Текущее название: `ydb`.
  - `version` — версия коннектора, используемая для генерации записи. Текущая версия: `1.0.0`.
  - `ts_ms` — примерное время применения изменения в YDB, в миллисекундах.
  - `step` — глобальное время координатора. Компонент виртуальных меток времени.
  - `txId` — уникальный идентификатор транзакции. Компонент виртуальных меток времени.
  - `snapshot` — признак чтения из снапшота.
  - `user` — идентификатор пользователя. Присутствует, если включена настройка `USER_SIDS`. Содержит SID пользователя. Если происходит удаление записей по TTL, то указывается значение `ttl@system`.
  - `traceId` — идентификатор трассировки OpenTelemetry. Присутствует, если включена настройка `TRACE_IDS`.

При чтении с использованием Kafka API в качестве ключа сообщения указывается Debezium-совместимый первичный ключ измененной строки:

```
{
    "payload": {<columns>}
}
```

- `payload` — первичный ключ строки, которая была изменена. Содержит названия и значения столбцов, являющихся компонентами первичного ключа.

Время хранения записей

По умолчанию записи хранятся в потоке изменений в течение 24 часов с момента отправки. В зависимости от сценариев использования время хранения можно уменьшить или увеличить до 30 дней.

Важно

Записи, время хранения которых истекло, удаляются вне зависимости от того, успели их обработать (прочитать) или нет.

Удаление записей до их обработки клиентом приводит к возникновению пропусков офсетов, то есть офсеты последней прочитанной из партиции записи и самой ранней из доступных будут отличаться более, чем на единицу.

Для настройки времени хранения записей укажите параметр RETENTION_PERIOD при создании потока изменений. Данные хранятся в течение всего указанного периода независимо от того, были ли они прочитаны потребителями.

Количество партиций топика

По умолчанию количество партиций топика равно количеству партиций таблицы. Количество партиций топика можно переопределить, указав параметр TOPIC_MIN_ACTIVE_PARTITIONS при создании потока изменений. Кроме того, можно создать поток изменений, в котором количество партиций будет увеличиваться автоматически, задав параметр TOPIC_AUTO_PARTITIONING при создании потока изменений.

Примечание

В настоящий момент возможность явного указания числа партиций топика доступна только для таблиц, у которых первый компонент первичного ключа имеет тип `Uint64` или `Uint32`.

Создание и удаление потока изменений

Поток изменений может быть добавлен к существующей таблице или удален директивами ADD CHANGEFEED и DROP CHANGEFEED операции YQL `ALTER TABLE`. При удалении таблицы добавленный к ней поток изменений также будет удален.

Получение и изменение параметров топика

Для получения параметров топика можно воспользоваться SDK или YDB CLI, передав в аргументах путь до потока изменений, который формируется следующим образом:

```
путь/до/строковой_таблицы/имя_потока_изменений
```

> Например, если у строковой таблицы `table` в директории `my` есть поток изменений с именем `updates_feed`, то путь к нему будет выглядеть так:
>
> ```
> my/table/updates_feed
> ```

Параметры топика могут быть изменены с использованием выражения ALTER TOPIC. Поддерживаемые действия:

- изменение параметров:
  - `retention_period`;
  - `retention_storage_mb`;
  - `partition_write_burst_bytes`;
  - `partition_write_speed_bytes_per_second`.
- управление читателями.

Назначение и применение CDC

Об использовании CDC при разработке приложений смотрите в рекомендациях.

Time to Live (TTL) и вытеснение данных во внешнее хранилище

В разделе описан принцип работы TTL и его ограничения.

Принцип работы

TTL таблицы задаётся последовательностью тиров хранения. Для каждого тира задаётся выражение, при срабатывании которого строке присваивается данный тир. Когда строке будет присвоен тир, TTL автоматически выполнит заданное действие: перенесёт строку во внешнее хранилище или удалит её. Внешние хранилища представляются объектом external data source.

YDB позволяет указать для строковых и колоночных таблиц колонку (TTL-колонка), значения которой будут использоваться в TTL-выражениях. Выражение срабатывает, когда проходит указанное количество секунд от времени, записанного в TTL-колонку. Если в TTL-колонке `NULL`, то выражение никогда не сработает.

Момент времени, когда срабатывает TTL-выражение, определяется по следующей формуле:

```
eviction_time = valueof(ttl_column) + evict_after_seconds
```

> **Примечание**
>
> Не гарантируется, что удаление произойдет именно в `eviction_time` — оно может случиться позже. Если важно исключить из выборки логически устаревшие, но пока еще физически не удаленные строки, нужно использовать фильтрацию уровня запроса.

Непосредственно удалением данных занимается фоновая операция удаления — *Background Removal Operation* (*BRO*), состоящая из 2 стадий:

1. Проверка значений в TTL-колонке.
2. Удаление устаревших данных.

BRO обладает следующими свойствами:

- Единицей параллельности является партиция таблицы.
- Для таблиц со вторичными индексами стадия удаления является распределенной транзакцией.

Гарантии

- Для одной и той же партиции *BRO* запускается с периодичностью, заданной в настройках TTL. Интервал запуска по умолчанию — 1 час, минимально допустимое значение — 15 минут.
- Гарантируется согласованность данных. Значение в TTL-колонке повторно проверяется во время стадии удаления. Таким образом, если между стадиями 1 и 2 значение в TTL-колонке модифицируется (например, запросом `UPDATE`) и перестает соответствовать критерию удаления, такая строка не будет удалена.

Ограничения

- TTL-колонка должна быть одного из следующих типов:
  - `Date`;
  - `Datetime`;
  - `Timestamp`;
  - `Uint32`;
  - `Uint64`;
  - `DyNumber`.
- Значение TTL-колонки с числовым типом (`Uint32`, `Uint64`, `DyNumber`) интерпретируется как величина от Unix-эпохи. Поддерживаемые единицы измерения (задаются в настройках TTL):
  - секунды;
  - миллисекунды;
  - микросекунды;
  - наносекунды.
- Нельзя указать несколько TTL-колонок.
- Нельзя удалить TTL-колонку. Если это все же требуется, сначала нужно выключить TTL на таблице.
- В качестве типа внешнего хранилища доступен только Object Storage.
- Действие `DELETE` может быть задано только для последнего тира.

Настройка

Управление настройками TTL в настоящий момент возможно с использованием:

- YQL.
- Консольного клиента YDB.
- YDB C++, Go и Python SDK.

Ограничения базы данных

В разделе описаны параметры ограничений, установленных в YDB.

Ограничения объектов схемы

В таблице ниже приведены ограничения объектов схемы: таблиц, баз данных, столбцов. В столбце *Объект* указан тип объекта схемы, на который распространяется ограничение.
В столбце *Тип ошибки* указан статус, с которым завершится выполнение запроса в случае ошибки. Подробнее статусы описаны в разделе Обработка ошибок в API.

Объект	Ограничение	Значение	Пояснение	Внутреннее название	Тип ошибки	Текст ошибки
База данных	Максимальная глубина пути	32	Максимальное количество вложенных элементов пути (директорий, таблиц).	MaxDepth	SCHEME_ERROR	Paths depth limit exceeded
База данных	Максимальное количество путей (объектов в схеме)	10 000	Максимальное количество элементов пути (директорий, таблиц) в базе данных.	MaxPaths	PRECONDITION_FAILED	Paths count limit exceeded
База данных	Максимальное количество таблеток	200 000	Максимальное количество таблеток (шардов таблиц и системных таблеток), которое может работать в базе данных. На запрос создания, копирования или изменения таблицы, в результате выполнения которого данное ограничение может быть превышено, будет возвращена ошибка. При достижении максимального количества таблеток в базе автоматическое шардирование таблиц не происходит.	MaxShards	PRECONDITION_FAILED	Shards count limit exceeded (in subdomain)
База данных	Максимальная длина имени объекта	255	Ограничивает количество символов в имени объекта схемы, например директории или таблицы.	MaxPathElementLength	SCHEME_ERROR	Path part is too long
База данных	Максимальный размер ACL	10 Кб	Максимальный суммарный размер всех правил контроля доступа, которые можно сохранить для одного объекта схемы.	MaxAclBytesSize	BAD_REQUEST	ACL size limit exceeded
Директория	Максимальное количество объектов	100 000	Максимальное количество таблиц, директорий, созданных в директории.	MaxChildrenInDir	PRECONDITION_FAILED	Children count limit exceeded

Таблица	Максимальное количество шардов таблицы	35 000	Максимальное количество шардов таблицы.	MaxShardsInPath	PRECONDITION_FAILED	Shards count limit exceeded (in dir)
Таблица	Максимальное количество столбцов	200	Ограничивает общее количество столбцов в таблице.	MaxTableColumns	SCHEME_ERROR при создании / BAD_REQUEST при изменении	Too many columns
Таблица	Максимальная длина имени столбца	255	Ограничивает количество символов в имени столбца.	MaxTableColumnNameLength	SCHEME_ERROR	Column name too long
Таблица	Максимальное количество столбцов в первичном ключе	20	Каждая таблица должна иметь первичный ключ. Количество столбцов, входящих в состав первичного ключа, не может превышать это ограничение.	MaxTableKeyColumns	SCHEME_ERROR	Too many key columns
Таблица	Максимальное количество индексов	20	Максимальное количество индексов помимо индекса первичного ключа, которые могут быть созданы на таблице.	MaxTableIndices	PRECONDITION_FAILED	Indexes count has reached maximum value in the table
Таблица	Максимальное количество реплик	3	Максимальное количество read-only реплик, которое можно указать при создании таблицы с репликами.	MaxFollowersCount	BAD_REQUEST	Too much followers
Таблица	Максимальное количество копируемых таблиц	10 000	Ограничение на размер списка таблиц для операций консистентного копирования таблиц.	MaxConsistentCopyTargets	BAD_REQUEST	Consistent copy object count limit exceeded

Ограничения размеров хранимых данных

Параметр	Значение	Тип ошибки
Максимальный суммарный размер всех столбцов в первичном ключе	1 Мб	GENERIC_ERROR
Максимальный размер значения строкового столбца	16 Мб	GENERIC_ERROR

Ограничения аналитических таблиц

Параметр	Значение
Максимальный размер строки	8 Мб
Максимальный размер вставляемого блока данных	8 Мб

Ограничения при выполнении запросов

В таблице ниже перечислены ограничения, действующие при выполнении запросов.

Параметр	Значение	Пояснение	Эффект в случае нарушения ограничения
Длительность запроса	2 часа	Максимальное время на выполнение одного запроса.	`TIMEOUT`
Максимальное количество сессий на ноду кластера	1000	Ограничение на количество сессий, которые клиент может создать с каждой нодой YDB.	`OVERLOADED`

Максимальная длина текста запроса	10 Кб	Ограничение на длину текста YQL-запроса.	`BAD_REQUEST`
Максимальный размер значений параметров	50 Мб	Ограничение на суммарный размер параметров, передаваемых при выполнении параметризованного запроса.	`BAD_REQUEST`
Максимальный размер строки	50 Мб	Ограничение на суммарный размер всех полей одной строки, возвращаемой или формируемой запросом.	`PRECONDITION_FAILED`
Максимальное количество блокировок на DataShard	10000	Количество оптимистично заблокированных диапазонов ключей на каждом `DataShard`	Некоторые блокировки конвертируются в полношардовые (используют меньше памяти, но берут блокировку на весь шард вместо его части).

Устаревшие ограничения

В предыдущих версиях YDB запросы выполнялись с помощью API под названием «Table Service». Этот API имел следующие ограничения, которые были устранены при переходе на новый API под названием «Query Service».

Параметр	Значение	Пояснение	Статус в случае нарушения ограничения
Максимальное количество строк в результате запроса	1000	Полные результаты некоторых запросов, выполненных через метод `ExecuteDataQuery`, могут содержать больше строк, чем допустимо. В этом случае в ответ на запрос вернется максимально допустимое число строк, а у результата будет установлен флаг `truncated`.	SUCCESS
Максимальный размер результата запроса	50 Мб	Полный результат некоторых запросов может превышать установленное ограничение. В этом случае запрос завершится ошибкой, данные не будут возвращены.	PRECONDITION_FAILED

Ограничения пулов ресурсов

Параметр	Значение
Максимальное число классификаторов пулов ресурсов	1000

Ограничения на число ресурсов накладываются ограничениями на число схемных объектов. Эти ограничения можно найти выше.

Концепции резервного копирования

YDB обеспечивает сохранность данных при аппаратных сбоях за счёт репликации и отказоустойчивости. Однако репликация не защищает от **логических ошибок**: случайный `DROP TABLE`, ошибочный массовый `UPDATE` или `DELETE` будут воспроизведены на все реплики. Для защиты от таких сценариев необходимо **резервное копирование** — отдельная копия данных, из которой можно восстановиться.

Полное резервное копирование

Полная резервная копия — это снимок данных таблиц на определённый момент времени. YDB предоставляет несколько способов создания полных резервных копий, от простых к более функциональным.

Копирование таблиц внутри кластера

Самый простой способ — создать копию таблицы (или нескольких таблиц) внутри того же кластера с помощью команды `ydb tools copy`. Копия создаётся атомарно из согласованного снимка и использует механизм копирования при записи, поэтому операция выполняется быстро.

Подходит для:

- быстрого создания «страховочной» копии перед опасной операцией;
- клонирования данных для тестирования.

> **Важно**
>
> Копия хранится в том же кластере, что и исходные данные. Она защищает от логических ошибок, но не от потери кластера.

Дамп в файловую систему

Команды `ydb tools dump` и `ydb tools restore` позволяют выгрузить данные в локальную файловую систему и восстановить их обратно.

Подходит для:

- локальной разработки и тестирования;
- небольших баз данных;
- создания копии на отдельном носителе.

Экспорт в S3-совместимое хранилище

Команды `ydb export s3` и `ydb import s3` позволяют выгружать и загружать данные во внешнее S3-совместимое хранилище.

Подходит для:

- аварийного восстановления (данные хранятся вне кластера);
- миграции данных между кластерами;
- долгосрочного архивирования.

Экспорт в NFS

Команды `ydb export nfs` и `ydb import nfs` позволяют выгружать и загружать данные в сетевую файловую систему (NFS), смонтированную на всех хостах кластера YDB. В отличие от дампа в файловую систему, выгрузка выполняется на стороне сервера и не требует передачи данных через CLI.

Подходит для:

- аварийного восстановления (данные хранятся вне кластера);
- миграции данных между кластерами;
- долгосрочного архивирования.

Подробнее про настройку NFS для резервного копирования и восстановления см. рецепт Резервное копирование и восстановление через NFS.

Инкрементальное резервное копирование

При работе с большими таблицами повторное создание полных резервных копий может быть слишком затратным. Инкрементальное резервное копирование решает эту задачу: после начальной полной копии каждый последующий инкремент захватывает только изменения (вставки, обновления, удаления), произошедшие с момента предыдущей резервной копии.

Инкрементальные резервные копии организованы в **цепочку**:

```
Полная копия → Инкремент₁ → Инкремент₂ → ... → Инкрементₙ
```

Для восстановления необходима вся цепочка: сначала применяется полная копия, затем последовательно все инкременты. Восстановление выполняется к состоянию на момент последнего инкремента в цепочке.

Инкрементальное резервное копирование реализуется с помощью коллекций резервных копий.

> **Примечание**
>
> В настоящее время поддерживаются только строковые таблицы.

Резервное копирование системных таблеток

> **Примечание**
>
> На текущий момент поддерживается резервное копирование только кластерных системных таблеток. Резервное копирование системных таблеток баз данных не поддерживается.

Механизм резервного копирования системных таблеток обеспечивает инкрементальное копирование метаданных кластера — таких как Hive, BSController и SchemeShard — на локальную файловую систему хостов кластера.

Этот механизм используют для восстановления метаданных кластера, когда восстановление из резервных копий баз данных технически возможно, но не подходит по времени или объёму работ. Типичный сценарий — совокупный объём баз данных в кластере велик из-за их количества, размера отдельных баз или сочетания обоих факторов; полный `import/restore` всех данных в новый кластер в таком случае приводит к длительному простою. В этом сценарии можно восстановить системные таблетки и вернуть кластер к работоспособному состоянию, не выполняя массовое восстановление пользовательских данных на новом кластере.

Если объём баз данных позволяет выполнить штатное восстановление, используйте в первую очередь export/import или dump/restore. Резервное копирование системных таблеток следует применять как специальный механизм для ситуаций, когда нужно восстановить именно метаданные кластера и сократить объём восстановительных операций.

> **Примечание**
>
> Практические инструкции по включению и восстановлению см. в рецептах для резервного копирования системных таблеток.

> **Важно**
>
> Резервные копии разных системных таблеток создаются независимо друг от друга и не согласованы между собой. После восстановления состояние таблеток может оказаться несогласованным, что может негативно влиять на работу кластера.

Принцип работы

Резервное копирование состоит из двух компонентов:

- **Снимок состояния (snapshot)** — при каждом запуске таблетка сканирует все свои таблицы и записывает свое полное состояние в резервную копию, включая схему данных. Сканирование выполняется на основе снимка состояния и не блокирует работу таблетки.
- **Журнал изменений (changelog)** — при каждом изменении данных или схемы таблетка асинхронно записывает изменение в журнал параллельно с записью в распределённое хранилище. Когда размер журнала превышает размер снимка состояния, таблетка автоматически делает новый снимок.

> **Важно**
>
> Из-за асинхронной записи возможна потеря последних изменений, не успевших попасть в резервную копию до момента сбоя.

Резервные копии создаются **локально на хосте, где в данный момент работает таблетка**. Поэтому наиболее актуальная копия находится на том хосте, где таблетка работала непосредственно перед сбоем.

Количество хранимых резервных копий на хосте ограничено в конфигурации. После успешного снятия снапшота самая старая копия автоматически удаляется при превышении лимита. Незавершённые копии (без полностью записанного снапшота) удаляются при создании новой резервной копии.

Сравнение подходов

Способ	Где хранится	Инкременты	Сценарии
Копирование таблиц внутри кластера	В кластере	Нет	Быстрая копия перед опасной операцией
Дамп в файловую систему	Файловая система	Нет	Разработка, тестирование, небольшие БД
Экспорт в S3-совместимое хранилище	S3-совместимое хранилище	Нет	Аварийное восстановление, миграция, архивирование
Экспорт в NFS	Сетевая файловая система (NFS)	Нет	Аварийное восстановление, миграция, архивирование
Инкрементальное резервное копирование	В кластере (экспортируются в S3 или файловую систему)	Да	Регулярные бэкапы больших производственных БД
Резервное копирование системных таблеток	Локальная ФС хостов кластера	Да	Восстановление метаданных кластера в аварийных ситуациях

См. также

- Резервное копирование и восстановление — практическое руководство
- Коллекции резервных копий — архитектура и ограничения
- Рецепты для резервного копирования системных таблеток — включение и восстановление
- Резервное копирование и восстановление — практическое руководство
- Справочник export/import — команды CLI
- Справочник YQL:

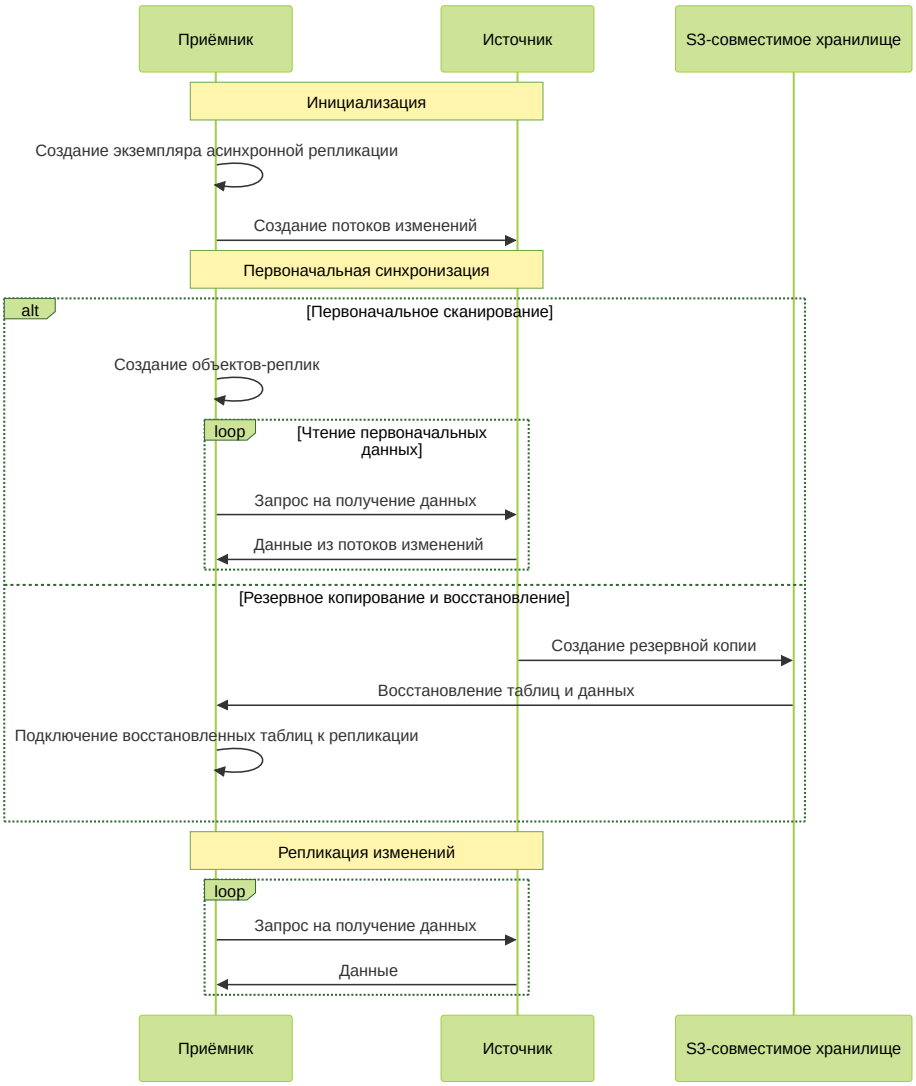
- `CREATE BACKUP COLLECTION`
- `BACKUP`
- `RESTORE`
- `DROP BACKUP COLLECTION`

Асинхронная репликация

Асинхронная репликация позволяет синхронизировать данные между базами данных YDB почти в реальном времени. Также она может быть использована для миграции данных между базами данных с минимальным простоем работающих с ними приложений. Базы данных могут располагаться как в одном кластере YDB, так и на разных.

Принцип работы

Асинхронная репликация реализована поверх Change Data Capture и оперирует логическими данными. Схема процесса репликации:



Как следует из схемы выше, в асинхронной репликации участвуют две базы:

1. **Источник** — база данных с исходными объектами.
2. **Приёмник** — база данных, в которой будут созданы экземпляр асинхронной репликации и объекты-реплики.

Процесс асинхронной репликации состоит из следующих стадий:

- Инициализация.
- Первоначальная синхронизация.
- Репликация изменений.

Инициализация

Инициализация асинхронной репликации состоит из:

- Создания экземпляра асинхронной репликации в приемнике с помощью выражения CREATE ASYNC REPLICATION.
- Установки соединения с источником. Для соединения с источником приемник использует параметры подключения, указанные при создании экземпляра асинхронной репликации.

> **Примечание**
>
> Пользователь, от имени которого производится подключение к источнику, должен обладать следующими правами:
>
> - просмотр объектов схемы и содержимого каталогов;
> - создание, изменение, удаление и чтение потоков изменений.

- Для заданного набора исходных объектов автоматически создаются потоки изменений в источнике.
- При включённом первоначальном сканировании в приёмнике автоматически создаются объекты-реплики. При синхронизации с помощью резервного копирования и восстановления целевые таблицы сначала восстанавливаются из резервной копии, затем подключаются к репликации.

Примечание

Объекты-реплики создаются или подключаются от имени пользователя, создавшего экземпляр асинхронной репликации.

Первоначальная синхронизация

Первоначальная синхронизация переносит существующие данные источника в приёмник перед переходом к репликации изменений. Способ синхронизации задаётся параметром INITIAL_SCAN при создании экземпляра репликации.

Первоначальное сканирование

При `INITIAL_SCAN = TRUE` репликация автоматически создаёт целевые таблицы и переносит в них исходные данные через потоки изменений. Это стандартный способ синхронизации. Если параметр не указан, используется настройка сервера; по умолчанию первоначальное сканирование включено.

Первоначальные данные исходных таблиц отгружаются в потоки изменений с использованием первоначального сканирования. В это время в приемнике запускаются читатели, которые записывают прочитанные данные в объекты-реплики.

Прогресс первоначального сканирования можно получить из описания экземпляра асинхронной репликации.

Синхронизация с помощью резервного копирования и восстановления

При `INITIAL_SCAN = FALSE` первоначальные данные переносятся серверными операциями резервного копирования в S3 и восстановления из S3. Этот способ позволяет использовать механизм резервного копирования для переноса большого объёма данных.

Репликация создаётся в состоянии `PAUSED` и подготавливает потоки изменений без первоначального сканирования. После готовности всех потоков пользователь создаёт резервную копию исходных таблиц и восстанавливает их в приёмнике. Пока идёт перенос, изменения источника накапливаются в потоках. Целевые таблицы на этом этапе не подключены к репликации.

После завершения восстановления пользователь переводит репликацию в `STANDBY` с помощью ALTER ASYNC REPLICATION. Приёмник проверяет схемы восстановленных таблиц, подключает их как объекты-реплики и начинает применять накопленные изменения. Подключённые таблицы становятся доступны только для чтения.

Для полноты данных потоки изменений должны быть готовы до создания резервной копии, а накопленные записи должны оставаться доступными до подтверждения их обработки. Период доступности задаётся параметром AVAILABILITY_PERIOD. Пошаговая процедура и проверки приведены в руководстве по первоначальной синхронизации с помощью резервного копирования и восстановления.

Выбор способа первоначальной синхронизации

Оба способа требуют чтения исходных данных и записи в приёмник. Дополнительные затраты различаются:

Затраты	Первоначальное сканирование	Резервное копирование и восстановление
Обработка данных	Каждая исходная строка сериализуется в CDC-сообщение, записывается в поток, читается и применяется в приёмнике	Данные выгружаются и загружаются серверными операциями; через CDC передаются только изменения
Временное хранение	Первоначальные данные и изменения хранятся в потоках с учётом периода доступности	Требуются согласованная копия таблиц в источнике и резервная копия в S3; в потоках хранятся изменения с учётом периода доступности
Передача по сети	Первоначальные данные передаются из потоков изменений источника в приёмник	Данные передаются из источника в S3, затем из S3 в приёмник; сжатие выгрузки позволяет уменьшить объём передачи
Настройка и эксплуатация	Синхронизация запускается автоматически при создании репликации	Требуются S3-хранилище, отдельные операции резервного копирования и восстановления, проверка их завершения и запуск репликации

Первоначальное сканирование проще в настройке и подходит для небольших объёмов данных. Для больших объёмов резервное копирование и восстановление позволяют сократить затраты на обработку первоначальных данных через CDC, но требуют дополнительного хранения и операций с S3. Итоговые затраты и длительность зависят от объёма данных, интенсивности изменений, ресурсов кластеров, сжатия и условий использования S3.

Репликация изменений

После первоначальной синхронизации читатели продолжают получать изменения источника и записывать их в объекты-реплики.

Для каждого изменения известно *время его возникновения* (). В то же время читатели отслеживают *время получения* изменения (). Таким образом, *отставание репликации* может быть вычислено по формуле:

Отставание репликации также можно получить из описания экземпляра асинхронной репликации с параметром `--stats`. При остановке читателя, потере его сессии или отсутствии свежего отчёта в течение 60 секунд значение отставания становится недоступным. Отсутствие значения означает, что отставание неизвестно, а не равно нулю.

Хранение изменений в источнике

Параметр AVAILABILITY_PERIOD определяет, как долго потоки изменений сохраняют сообщения, обработку которых репликация ещё не подтвердила. Он задаётся при создании репликации и не может быть изменён с помощью `ALTER ASYNC REPLICATION`.

По умолчанию используется значение `'UNLIMITED'`: неподтверждённые сообщения сохраняются до подтверждения их обработки. Для ограничения периода доступности укажите положительный интервал, например `Interval('PT48H')`.

При любом выбранном периоде доступности сообщения в топике удаляются практически сразу после подтверждения их обработки читателем, чтобы освобождать место в источнике и сокращать затраты на хранение уже обработанных изменений. Параметр `AVAILABILITY_PERIOD` защищает от удаления только неподтверждённые сообщения.

Условие	Без ограничения: `'UNLIMITED'`	Конечный период: например, `Interval('PT48H')`
Длительная остановка или отставание	Неподтверждённые сообщения сохраняются независимо от их возраста	По истечении заданного периода неподтверждённые сообщения могут быть удалены
Расход места в источнике	Объём накопленных сообщений может расти всё время, пока репликация не подтверждает обработку	Возраст удерживаемых для репликации сообщений ограничен заданным периодом
Первоначальный перенос через резервную копию	Нет ограничения по возрасту накопленных изменений; необходимо обеспечить место для их хранения	Экспорт, импорт и обработка накопленных изменений должны укладываться в выбранный период
Что контролировать	Свободное место в источнике и отставание репликации	Отставание репликации и запас времени до истечения периода доступности

Период доступности относится к возрасту сообщений, а не к длительности текущего простоя. Состояние `PAUSED` не останавливает отсчёт этого периода. Если нужные сообщения уже удалены, для восстановления полноты данных потребуется повторная первоначальная синхронизация.

При `'UNLIMITED'` используется важный читатель, а при конечном периоде — читатель с заданным временем доступности сообщений. Благодаря этому неподтверждённые сообщения могут храниться дольше времени хранения сообщений в топике.

Ограничения

- Набор исходных объектов задается при создании экземпляра асинхронной репликации и не может быть изменен.
- В качестве исходных объектов поддерживаются:
  - строковые таблицы, в том числе с синхронными вторичными индексами, включая уникальные (кроме векторных);
  - директории — будут реплицироваться все строковые таблицы, содержащиеся в них (включая содержимое поддиректорий) на момент создания экземпляра асинхронной репликации.
- Во время работы асинхронной репликации в исходных таблицах блокируется возможность изменения состава столбцов.
- Во время работы асинхронной репликации объекты-реплики доступны только для чтения.

Обработка ошибок в процессе асинхронной репликации

В процессе асинхронной репликации возможно возникновение разных классов ошибок:

- **Временные сбои**. Например, транспортные ошибки, перегрузка системы и т.д. Запросы будут повторяться до успешного выполнения.
- **Критичные ошибки**. Например, ошибки прав доступа, ошибки схемы и т.д. Процесс репликации будет остановлен, и в описании экземпляра репликации будет указан текст ошибки.

> **Важно**
>
> На данный момент остановленный из-за критичной ошибки процесс асинхронной репликации не может быть перезапущен. Его нужно удалить и создать заново.

Подробнее про классы ошибок и политики их обработки см. в разделе Обработка ошибок.

Уровни согласованности реплицируемых данных

Согласованность данных уровня строк

Механизм Change Data Capture, используемый для асинхронной репликации, гарантирует, что изменения по одному и тому же первичному ключу доставляются в том же порядке, в котором они происходили в источнике. Таким образом обеспечивается согласованность данных уровня строк.

Данные, записанные в объекты-реплики в этом режиме, сразу становятся доступными для чтения.

Глобальная согласованность данных

При выполнении транзакций в источнике могут быть атомарно изменены несколько строк таблицы или, более того, строки в разных партициях таблиц(ы). Глобальная согласованность данных в приемнике подразумевает сохранение свойства атомарности — изменения "проявляются" (становятся доступными для чтения) согласованно.

Для обеспечения глобальной согласованности данных необходимо создавать потоки изменений с включенными барьерами. Между барьерами данные буферизируются непосредственно на партициях объектов-реплик. При получении очередной порции барьеров от всех топиков записанные изменения фиксируются и становятся доступными для чтения. При этом данные в приемнике оказываются согласованными на момент времени, указанном в барьере. Если считать, что в источнике барьеры создаются каждые 10 секунд (значение по умолчанию), то в нормальном режиме приемник коммитит и публикует набор изменений от источника также каждые 10 секунд.

Периодичность фиксации изменений

По умолчанию изменения фиксируются (коммитятся) не чаще, чем раз в 10 секунд. Периодичность фиксации изменений можно переопределить, указав опцию COMMIT_INTERVAL при создании экземпляра асинхронной репликации.

> **Примечание**
>
> Периодичность фиксации изменений напрямую влияет на периодичность выгрузки барьеров в потоки изменений — значения параметров синхронизированы. Таким образом, обычно, изменения фиксируются с той же частотой, с которой в потоках изменений появляются барьеры. Однако, в некоторых случаях, например, в случае неравномерной нагрузки между таблицами, барьеры в потоках изменений могут появляться с существенной разницей во времени, что приведёт к увеличению периодичности фиксации изменений. Отставание можно получить из описания экземпляра асинхронной репликации.

Завершение асинхронной репликации

Завершение асинхронной репликации может быть конечной целью процедуры миграции данных из одной базы данных в другую. В таком случае клиент останавливает пишущую нагрузку в источнике, дожидается нулевого отставания репликации и завершает её. При завершении объекты-реплики становятся доступными как для чтения, так и для записи. После этого можно переключить нагрузку с источника на приемник и завершить процедуру миграции.

Перед завершением получите актуальное значение отставания с помощью `scheme describe --stats`. Если значение недоступно, дождитесь возобновления отчётов: отсутствие значения или ранее показанный ноль не подтверждают, что все изменения перенесены.

Примечание

Завершённый процесс асинхронной репликации не может быть возобновлён.

Важно

На данный момент поддерживается только **принудительное** завершение асинхронной репликации, при котором не производится никаких дополнительных проверок (на согласованность данных, на отставание и т.д.).

Для завершения асинхронной репликации используйте выражение ALTER ASYNC REPLICATION.

Удаление асинхронной репликации

При удалении асинхронной репликации:

- В исходных таблицах удаляются потоки изменений.
- В исходных таблицах разблокируется возможность изменения состава столбцов.
- Опционально могут быть удалены все объекты-реплики.
- Удаляется экземпляр асинхронной репликации.

Важно

Если удалить асинхронную репликацию без предварительного завершения и не удалять объекты-реплики, то они останутся доступными только для чтения.

Для удаления асинхронной репликации используйте выражение DROP ASYNC REPLICATION.

Режим работы кластера Bridge

> **Функциональность Корпоративной СУБД Яндекса**
>
> Данная функциональность доступна только в Корпоративной СУБД Яндекса. В open-source версии YDB она отсутствует.

Эта статья описывает особый режим работы кластера YDB, при котором хранение данных осуществляется с синхронной репликацией между несколькими частями кластера, называемыми pile.

Режим bridge является надстройкой над базовыми режимами распределённого хранилища: каждый pile строится по одному из поддерживаемых YDB режимов работы (например, `mirror-3-dc` или `block-4-2`) и сам по себе отказоустойчив; bridge добавляет поверх этого синхронную репликацию между pile и управляемый failover/switchover нагрузки между ними. Это обеспечивает дополнительный слой отказоустойчивости поверх выбранных топологий. Режим bridge требует больше оборудования и сложнее в эксплуатации; он особенно полезен для кластеров, размещённых в двух дата-центрах, и систем с повышенными требованиями к доступности — например, когда необходимо сохранять доступность при отказе трёх дата-центров из четырёх.

Описание режима

В режиме bridge узлы кластера разделяются на несколько pile (обычно — по одному pile в датацентре). При отказе одного из pile кластер становится недоступным до тех пор, пока не будет выполнена команда отключения этого pile. После отключения кластер возобновляет работу.

YDB поддерживается произвольное количество pile, однако для простоты изложения далее рассматривается случай с двумя pile.

На уровне распределённого хранилища в режиме bridge реализуется синхронная репликация по схеме Active / Active между pile. На уровне узлов баз данных возможно резервирование по схеме Active / Passive.

Для хранения данных используются пары групп хранения из разных pile. При этом для хранения данных в каждом из pile могут применяться любые поддерживаемые YDB режимы работы.

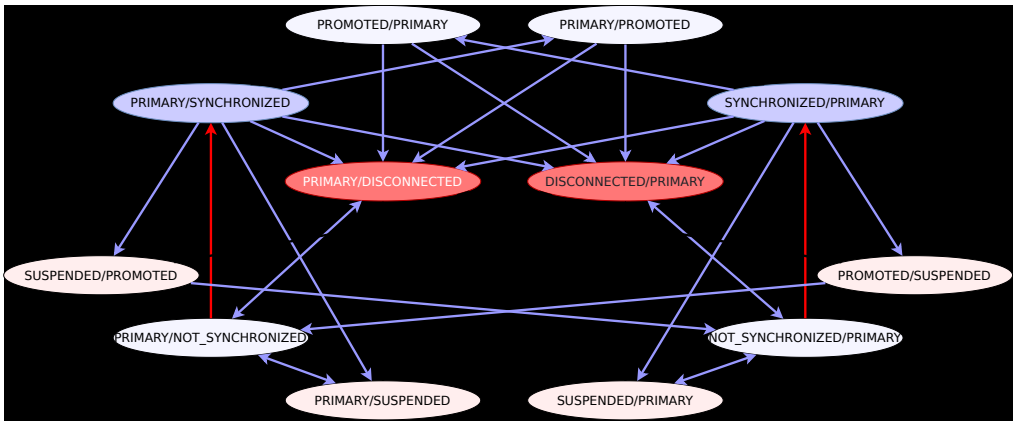
В режиме bridge может быть достигнут zero-downtime switchover. При failover кластер требует явного отключения репликации в отказавший pile для возобновления работы. При отключении любого pile происходит приостановка обслуживания запросов до выполнения команды отключения репликации в этот pile.

Pile не являются самостоятельными кластерами YDB, а представляют собой части одного кластера со сложной топологией:

- таблетки в режиме bridge запускаются в единственном экземпляре;
- в каждом pile работает отдельная статическая группа и набор независимых групп хранения с обычными VDisk. Доступ к группам хранения осуществляется через DS-proxy-proxy, который предоставляет таблеткам интерфейс DS-proxy и выполняет операции с использованием двух DS-proxy — по одной для групп в каждом pile;
- в каждом pile работает свой набор реплик StateStorage, SchemeBoard и Board. Доступ к StateStorage, SchemeBoard и Board осуществляется через аналогичные proxy-proxy.

Состояния Pile

Режим работы кластера определяется состояниями всех его pile. Например, для кластера из pile А и pile Б состояния pile указываются как (А, Б), где порядок имеет значение.



Каждый pile может находиться в одном из следующих состояний:

- `PRIMARY`. Основной pile, в котором работают таблетки. Только один pile может находиться в состоянии `PRIMARY`.
- `SYNCHRONIZED`. Ведомый pile, в котором все группы хранения синхронизированы с `PRIMARY`.
- `DISCONNECTED`. Отключённый от `PRIMARY`, не участвует в выполнении операций.
- `NOT_SYNCHRONIZED`. Pile, в котором группы хранения ещё не синхронизированы с `PRIMARY`. При завершении синхронизации YDB автоматически переключает pile в состояние `SYNCHRONIZED`.
- `PROMOTED`. Pile, который необходимо плавно переключить из состояния `SYNCHRONIZED` в состояние `PRIMARY`. Pile пребывает в этом состоянии до завершения плавного переключения, после чего YDB автоматически переключает его в состояние `PRIMARY`.
- `SUSPENDED`. Pile, который необходимо плавно переключить в состояние `DISCONNECTED`. По завершении переключения YDB автоматически переводит pile в состояние `DISCONNECTED`.

В норме пара pile работает в режиме `PRIMARY/SYNCHRONIZED` или `SYNCHRONIZED/PRIMARY`. То есть один из pile — `PRIMARY`, в нём работают таблетки, второй pile — `SYNCHRONIZED`, все его группы хранения синхронизированы с `PRIMARY`. Операция записи считается завершённой при таком состоянии pile только после успешного выполнения записи в группы хранения каждого pile с необходимой избыточностью.

Переходы между состояниями

Состояние до	Состояние после	Как происходит	Пояснение
`PRIMARY`	`SYNCHRONIZED`	Автоматически	Завершение планового переключения.
`PRIMARY`	`DISCONNECTED`	Вручную — failover	Аварийное отключение недоступного `PRIMARY` с выбором нового `PRIMARY`.
`PRIMARY`	`SUSPENDED`	Вручную — takedown	Плановое отключение текущего `PRIMARY` с выбором нового `PRIMARY`.
`SYNCHRONIZED`	`PRIMARY`	Вручную — failover	Аварийное отключение недоступного `PRIMARY` с выбором нового `PRIMARY`.
`SYNCHRONIZED`	`DISCONNECTED`	Вручную — failover	Аварийное отключение недоступного `SYNCHRONIZED`.
`SYNCHRONIZED`	`PROMOTED`	Вручную — switchover	Старт плановой смены `PRIMARY`.
`SYNCHRONIZED`	`SUSPENDED`	Вручную — takedown	Старт планового отключения `SYNCHRONIZED` pile.
`DISCONNECTED`	`NOT_SYNCHRONIZED`	Вручную — rejoin	Возврат pile в кластер после обслуживания/ восстановления.
`NOT_SYNCHRONIZED`	`SYNCHRONIZED`	Автоматически	Завершение синхронизации данных.
`NOT_SYNCHRONIZED`	`DISCONNECTED`	Вручную — failover	Аварийное отключение недоступного `NOT_SYNCHRONIZED`.
`PROMOTED`	`PRIMARY`	Автоматически	Завершение планового переключения.
`PROMOTED`	`PRIMARY`	Вручную — failover	Аварийное отключение недоступного `PRIMARY` с выбором нового `PRIMARY`.
`PROMOTED`	`DISCONNECTED`	Вручную — failover	Аварийное отключение недоступного `PROMOTED`.
`SUSPENDED`	`NOT_SYNCHRONIZED`	Вручную — rejoin	Отмена планового отключения pile.
`SUSPENDED`	`DISCONNECTED`	Автоматически	Завершение планового отключения.
`SUSPENDED`	`DISCONNECTED`	Вручную — failover	Аварийное отключение недоступного `SUSPENDED`.

Сценарии изменения состояния

Отказ pile

При отказе pile кластер становится недоступным. Для возобновления работы кластера необходимо отключить отказавший pile, переведя его в состояние `DISCONNECTED`.

Администратор может подать команду переключения из состояния `PRIMARY/SYNCHRONIZED` или `SYNCHRONIZED/PRIMARY` в одно из состояний `PRIMARY/DISCONNECTED` или `DISCONNECTED/PRIMARY`. Команда изменяет конфигурацию групп хранения и приводит к изменению способа записи в эти группы. Interconnect разрывает сессии со всеми узлами pile в состоянии `DISCONNECTED`; при этом сессии для обмена данными не устанавливаются (TCP-соединения для обмена метаданными о состояниях pile могут оставаться активными). Дальнейшие операции записи и чтения выполняются без участия pile в состоянии `DISCONNECTED`. Если изменился `PRIMARY` pile, таблетки перезапускаются в нём.

Аналогичным образом выполняется переключение, если отказавший pile находился в любом состоянии, а назначаемый `PRIMARY` pile — в одном из допустимых состояний: `PRIMARY`, `SYNCHRONIZED`, `PROMOTED`.

Если назначаемый `PRIMARY` pile находился в состоянии `DISCONNECTED`, `NOT_SYNCHRONIZED` или `SUSPENDED`, штатное переключение невозможно, поскольку pile может не содержать полной и актуальной реплики данных.

Восстановление pile

После восстановления работоспособности узлов отказавшего pile его необходимо повторно подключить к кластеру, переведя в состояние `NOT_SYNCHRONIZED`.

Администратор может подать команду переключения из состояния `PRIMARY/DISCONNECTED` в `PRIMARY/NOT_SYNCHRONIZED` или из `DISCONNECTED/PRIMARY` в `NOT_SYNCHRONIZED/PRIMARY`. Произойдёт обмен конфигурацией между узлами; сессии interconnect устанавливаются только между узлами с одинаковой конфигурацией. Команда запускает синхронизацию групп хранения. По завершении синхронизации происходит автоматическое переключение из состояния `PRIMARY/NOT_SYNCHRONIZED` в `PRIMARY/SYNCHRONIZED` или из `NOT_SYNCHRONIZED/PRIMARY` в `SYNCHRONIZED/PRIMARY`.

Плановое переключение PRIMARY pile

Для плановой смены `PRIMARY` pile необходимо перевести назначаемый новым `PRIMARY` pile в состояние `PROMOTED`.

Администратор может подать команду переключения из состояния `PRIMARY/SYNCHRONIZED` в `PRIMARY/PROMOTED` или из `SYNCHRONIZED/PRIMARY` в `PROMOTED/PRIMARY`. Команда не изменяет способ записи в группы хранения, но обновляет их конфигурацию и инициирует смену `PRIMARY` pile с плавным переносом таблеток в новый `PRIMARY`. По завершении переключения состояние `PRIMARY` сменяется на `SYNCHRONIZED`, а `PROMOTED` - на `PRIMARY`.

Плановое отключение pile

Для планового отключения pile его необходимо перевести в состояние `SUSPENDED`.

Администратор может подать команду переключения из состояния `PRIMARY/SYNCHRONIZED` в `PRIMARY/SUSPENDED` или из `SYNCHRONIZED/PRIMARY` в `SUSPENDED/PRIMARY`. Будет выполнено плановое отключение узлов pile, переведённого в состояние `SUSPENDED`, с переводом их в режим `DISCONNECTED`. Система стремится минимизировать влияние этого процесса на работу

кластера. После завершения перевода pile в состояние `DISCONNECTED` его узлы могут быть выключены для технического обслуживания.

После этого необходимо выполнить штатное восстановление pile, переключив его в состояние `NOT_SYNCHRONIZED`.

Восстановление при разделении кластера на две части с несовместимой конфигурацией (split brain)

Может возникнуть ситуация, когда администратор привёл кластер в состояние, при котором pile А и Б оказались изолированы друг от друга, а затем pile А был переконфигурирован так, что стал `PRIMARY`, а pile Б — `SYNCHRONIZED`, и одновременно pile Б был переконфигурирован так, что стал `PRIMARY`, а pile А — `SYNCHRONIZED`. Это приводит к разделению кластера на две части с несовместимой конфигурацией (split-brain). В таком состоянии каждая часть кластера может оставаться работоспособной.

Для восстановления единого кластера необходимо выбрать, какой из pile будет очищен. После этого нужно остановить все узлы этого pile, дождаться их полной остановки, очистить все диски на всех узлах, а затем снова включить узлы очищаемого pile.
После этого следует выполнить штатное восстановление pile, переведя его в состояние `NOT_SYNCHRONIZED`.

Нештатное восстановление pile

В сложных ситуациях, например при последовательных отказах pile, может возникнуть следующий сценарий: сначала отказал pile А, он был переведён в состояние `DISCONNECTED`, и кластер продолжил работу; затем произошёл необратимый отказ pile Б; после этого удалось восстановить работоспособность pile А. В таких случаях возможно восстановление кластера на основе pile А.

Если необходимо возобновить работу кластера, когда работоспособность сохранил только pile, находящийся в состоянии `DISCONNECTED`, `NOT_SYNCHRONIZED` или `SUSPENDED`, администратор может выполнить команду переключения из `PRIMARY/DISCONNECTED` в `DISCONNECTED/PRIMARY` (или аналогично для `NOT_SYNCHRONIZED` или `SUSPENDED`) с указанием специального параметра `force`. Это приведёт к разделению кластера на две части с несовместимой конфигурацией. В зависимости от фактического состояния данных в pile, который переводится в состояние `PRIMARY`, кластер может быть восстановлен в корректное или внутренне неконсистентное состояние. Работоспособность такого кластера не гарантируется.

Если кластер оказался работоспособным, можно продолжить восстановление до нормального состояния. Перед восстановлением отключённого pile необходимо полностью очистить данные и метаданные на всех его узлах, после чего перевести pile в состояние `NOT_SYNCHRONIZED`.

Особенности реализации режима bridge

Подробнее об устройстве режима bridge см. в Устройство режима bridge.

Трансфер данных

Трансфер в YDB — асинхронный механизм переноса данных из топика в таблицу. Создание экземпляра трансфера, его изменение и удаление осуществляется с использованием YQL. Трансфер запускается внутри базы данных, работает в фоновом режиме. Трансфер используется для решения задачи поставки данных из топика в таблицу.

Часто на практике удобнее записывать данные не напрямую в таблицу, а в топик, чтобы затем в асинхронном режиме переписывать их из топика в таблицу. Такой подход позволяет равномерно распределять нагрузку и справляться со всплесками, поскольку операция записи в очередь сообщений более лёгкая. В зависимости от количества записываемых сообщений в топик, задержка доступности данных для чтения из таблицы после добавления в топик может составлять от нескольких секунд до нескольких минут.

Трансфер осуществляет чтение сообщений из указанного топика, их преобразование в формат, пригодный для записи в таблицу, и последующую запись в целевую таблицу. Если в таблице уже есть строка с указанным первичным ключом, то строка будет обновлена. Преобразование данных осуществляется с помощью lambda-функции, которая принимает сообщение в качестве параметра и возвращает список структур, каждая из которых соответствует строке, подлежащей добавлению или замене в таблице. Например, если таблица содержит колонку `example_column` типа `Int32`, структура должна включать именованное поле `example_column` того же типа.
Если lambda-функция возвращает пустой список, ни одна строка в таблице не будет изменена. Из lambda-функции не допускается обращение к таблицам и другим объектам YDB.
Используемую lambda-функцию можно получить из описания экземпляра трансфера.

На работу трансфера тратятся дополнительные ресурсы сервера, в первую очередь CPU. Потребление CPU зависит от сложности преобразования записываемых в топик данных.

Трансфер способен читать данные из топиков, расположенных как в базе данных, где он создаётся, так и в другой базе YDB или кластере YDB. Для чтения топика из другой базы данных при создании трансфера необходимо указать параметры подключения к этой базе. Целевая таблица должна находиться в той же базе данных, где создаётся трансфер.

Чтение из топика осуществляется с использованием читателя. Трансфер автоматически добавляет в топик читателя с уникальным именем для чтения сообщений. При удалении трансфера созданный читатель удаляется автоматически. Также возможно создать читателя вручную и указать его имя при создании трансфера. Читатель, созданный вручную, не удаляется автоматически при удалении трансфера. Имя автоматически созданного читателя можно получить из описания экземпляра трансфера.

Если требуется передача данных из одного топика в несколько таблиц, для каждой таблицы необходимо создать отдельный трансфер. Аналогично, данные из нескольких топиков могут быть направлены в одну таблицу, для чего каждый топик должен быть ассоциирован с отдельным трансфером. Если имя читателя задаётся вручную, то при наличии нескольких трансферов, читающих из одного топика, необходимо назначить каждому трансферу уникального читателя, иначе данные будут обработаны только одним из них.

> **Примечание**
>
> При обновлении строки в таблице производится полная перезапись её значений данными из возвращаемой lambda-функцией структуры. Если структура не содержит именованного поля, соответствующего колонке таблицы, значение этой колонки будет установлено в `NULL`. Именованные поля, не имеющие соответствия в таблице, игнорируются.
>
> При попытке записать в `NOT NULL` колонку таблицы значение `NULL` трансфер остановится с критичной ошибкой.

Гарантии

- Гарантируется, что записи в таблицу будут происходить в том же порядке, в которую они были записаны в партицию топика. Рекомендуется группировать сообщения, относящиеся к одной строке таблицы, в одну партицию топика.
- Запись в таблицу осуществляется с использованием пакетной вставки без гарантий атомарности. Запись данных разбивается на несколько независимых транзакций, каждая их которых затрагивает единственную партицию таблицы, с параллельным исполнением.

Начало работы трансфера

Если трансфер создаётся без явного указания имени читателя, то в топик будет добавлен новый читатель, и обработка сообщений топика начнётся с первого сообщения в топике.
Если трансфер создаётся с указанием созданного ранее читателя, то обработка сообщений топика начнётся с первого необработанного этим читателем сообщения.

Необходимые права для работы трансфера

Для создания и выполнения трансфера пользователь должен обладать правами на запись в целевую таблицу и чтение из исходного топика. Если топик расположен в другой базе данных YDB и читатель создаётся автоматически, дополнительно требуется право на модификацию топика, что позволяет трансферу автоматически создавать читателя и удалять его при удалении трансфера.

Диагностика

Текущие параметры трансфера, включая текст lambda-функции, можно просмотреть в пользовательском интерфейсе YDB UI и в описании экземпляра трансфера.

Скорость обработки данных и задержки можно контролировать по метрикам читателя, который используется для чтения из топика.

Временная остановка трансфера

Работа трансфера может быть временно приостановлена, а затем возобновлена. После возобновления работы трансфера начнут обрабатываться сообщения, следующие за последним обработанным сообщением в топике.

Для остановки трансфера следует изменить статус трансфера на `PAUSED`, а для возобновления — на `ACTIVE`.

Важно

Следует учесть, что в топике есть время хранения сообщений, по истечении которого сообщения удаляются. Если с момента остановки трансфера пройдёт больше времени, чем время хранения сообщений, то сообщения будут удалены и не будут обработаны после возобновления работы трансфера.

Чтобы гарантировать, что необработанные сообщения не будут удалены, следует сделать читателя важным.

Обработка ошибок в процессе трансфера

В процессе трансфера возможно возникновение разных типов ошибок:

- **Временные сбои**. Транспортные ошибки, перегрузка системы и другие временные проблемы. Запросы будут повторяться до успешного выполнения.
- **Критичные ошибки**. Ошибки, связанные с правами доступа, схемой данных и другими критическими аспектами. В случае возникновения таких ошибок трансфер будет остановлен, и в пользовательском интерфейсе YDB UI на странице трансфера будет указан текст ошибки. Также текст ошибки можно получить из описания экземпляра трансфера.

Для возобновления работы трансфера необходимо выполнить команду `ALTER TRANSFER` с устранением причины ошибки. Например, если ошибка содержалась в тексте lambda-функции, то следует изменить lambda-функцию. Если ошибка не связана с конфигурацией трансфера, например, отсутствовали права на чтение из топика, то после устранения причины ошибки трансфер необходимо перезапустить сначала временно остановив, а потом возобновив его работу.

См. также

- CREATE TRANSFER
- ALTER TRANSFER
- DROP TRANSFER
- Рецепты работы с трансфером данных из топика в таблицу

Администрирование кластеров

В этом разделе документации YDB описано всё, что нужно знать для работы с кластерами YDB.

Перед началом работы рекомендуется ознакомиться с системными требованиями YDB.

Основные подразделы:

- Концепции администрирования кластеров — дополнения к общему разделу концепций, актуальные для администрирования кластеров.
- Конфигурация кластеров YDB — управление конфигурацией кластеров YDB.
- Способы развёртывания YDB — способы развёртывания кластеров YDB.
  - **Ansible**: для развертываний на физическом оборудовании и виртуальных машинах.
  - **Kubernetes**: для развёртываний в контейнерах.
  - **Вручную**: развертывание кластера вручную.
- Обзор наблюдаемости — инструменты для наблюдения за кластерами YDB.
- Резервное копирование и восстановление — резервное копирование и восстановление кластеров YDB.
- Первоначальная синхронизация асинхронной репликации с помощью резервного копирования и восстановления: перенос первоначальных данных с помощью резервного копирования и восстановления и запуск асинхронной репликации.

Концепции администрирования кластеров

Данный раздел дополняет общий раздел Концепции YDB теоретическими материалами, актуальными в первую очередь для администрирования кластеров.

Основные темы:

- Системные требования и рекомендации для YDB
- Настройки CPU для production
- Планирование ёмкости хранилища
- Версионирование YDB
- Обслуживание кластера без потери доступности
- SelfHeal
- Настройка аутентификации и авторизации узлов баз данных

Смотрите также:

- Способы развёртывания YDB
- Конфигурация кластеров YDB
- Обзор наблюдаемости
- Резервное копирование и восстановление

Системные требования и рекомендации для YDB

В данном разделе приводятся рекомендации для развёртывания кластеров YDB, применимые вне зависимости от выбранного способа управления инфраструктурой (Ansible, Kubernetes или вручную).

Аппаратная конфигурация

Необходимое количество серверов и дисков определяется требованиями к отказоустойчивости. Подробнее читайте в разделе Топология кластера YDB.

- **Процессор (CPU)**

  YDB сервер может быть запущен только на процессорах с архитектурой x86-64 с поддержкой инструкций AVX2: Intel Haswell (4-е поколение) и более поздние, AMD EPYC и более поздние.

  Архитектура ARM в настоящее время не поддерживается для серверных развёртываний YDB. Для локальной разработки на компьютерах с процессорами Apple Silicon можно использовать Docker с эмуляцией набора инструкций x86_64 (Rosetta); см. вкладку Docker в Быстрый старт с YDB.

  > **Примечание**
  >
  > Энергосберегающие режимы CPU (C-states, P-states, scaling governor) могут увеличивать задержки за счёт задержек выхода из спящих состояний.
  >
  > Для production-нагрузок рекомендуется использовать режим performance.
  >
  > Подробнее см. в разделе Настройки CPU для production.

- **Оперативная память**

  Рекомендуется использовать оперативную память с поддержкой коррекции ошибок (ECC) для защиты от аппаратных сбоев.

- **Дисковая подсистема**

  YDB сервер может быть запущен на серверах с дисками любого типа (HDD/SSD/NVMe). Однако рекомендуется использовать диски SSD/NVMe для достижения большей производительности.

  Для эффективной работы YDB рекомендуется использовать физические (не виртуальные) диски объемом более 800 ГБ как блочные устройства.

  Минимальный объем диска должен быть не менее 80 ГБ, при меньшем объеме узел YDB не сможет использовать устройство. Корректная и бесперебойная работа с дисками минимального объема не гарантируется. Использовать такие диски рекомендуется исключительно в ознакомительных целях.

  > **Важно**
  >
  > Конфигурации с дисками объемом меньше 800 ГБ или с любыми видами виртуализации системы хранения нельзя использовать для сервисов, находящихся в промышленной эксплуатации, а также для тестирования производительности системы.
  >
  > Мы не рекомендуем использовать для хранения данных YDB диски, которые используются другими процессами (в том числе операционной системой).

  YDB не использует файловую систему для хранения данных и работает с диском напрямую. Поэтому не следует монтировать файловую систему или выполнять другие операции с разделом, который использует YDB. Также не рекомендуется делить блочное устройство с другими процессами — это может привести к существенному снижению производительности.

  Работоспособность и производительность YDB не тестировались ни на каких видах виртуальных или сетевых устройств хранения.

  При планировании объёма хранилища следует учитывать, что YDB использует часть дискового пространства для своих внутренних нужд. Так, например, на кластере среднего размера из 8 узлов можно ожидать потребления порядка 100 ГБ на весь кластер под статическую группу. На большом кластере с >1500 узлов — около 200 ГБ. Также имеются системные логи объёмом 25.6 ГБ на каждом Pdisk и системная область на каждом Pdisk. Её размер зависит от объёма Pdisk, но составляет не менее 0.2 ГБ.

  Также диск используется для спиллинга — механизма управления памятью, который позволяет временно сохранять промежуточные результаты выполнения запросов на диск при нехватке оперативной памяти. Это также важно учитывать при планировании объёма диска. Подробная конфигурация спиллинга описана в разделе Конфигурация спиллинга.

Программная конфигурация

YDB сервер может быть запущен на серверах с операционной системой Linux с ядром версии 4.19 и выше и libc версии 2.30. Например, Ubuntu 20.04, Debian 11, Fedora 34 или более новые версии. При этом для достижения наилучшей производительности рекомендуется использовать более новые версии ядра Linux (6.6 и выше), так как они содержат значительные улучшения в подсистемах ввода-вывода, планировщика и управления памятью, которые положительно влияют на работу СУБД.

YDB использует аллокатор памяти TCMalloc, для его эффективной работы рекомендуется включить Transparent Huge Pages и Memory overcommitment.

Для повышения производительности дискового и сетевого ввода-вывода в доверенной (trusted) среде можно отключить IOMMU, указав загрузочный параметр Linux `intel_iommu=off` или `amd_iommu=off`. В недоверенной (untrusted) среде, а также при жёстких требованиях безопасности или активном использовании виртуализации (например, PCI passthrough и изоляции устройств), отключение IOMMU не рекомендуется. В таких случаях используйте `intel_iommu=on iommu=pt` или `amd_iommu=on iommu=pt`.

Среду можно считать доверенной, если на сервере выполняется только YDB и приложения, находящиеся под контролем пользователя. К недоверенным следует относить конфигурации, в которых выполняются сторонние приложения или виртуальные машины.

Если сервер имеет более 32 ядер CPU, для повышения производительности YDB динамические узлы следует запускать в раздельных taskset/cpuset, размером от 10 до 32 ядер. Например, в случае 128 ядер оптимально запустить 4 динамических узла: каждый в своём taskset, размером 32 ядра. Ядра одного taskset/cpuset должны быть в одной NUMA-ноде.

Операционные системы macOS и Windows в данный момент не поддерживаются для запуска серверной части YDB. Тем не менее, для целей разработки и функционального тестирования допустимо запускать YDB на них в Docker-контейнере.

Настройки CPU для production

Проблема

Энергосберегающие режимы процессора (такие как Intel SpeedStep, AMD Cool'n'Quiet, C-states, P-states) могут негативно влиять на производительность YDB по нескольким причинам:

1. **Нестабильная частота процессора** — YDB ожидает стабильной производительности CPU для предсказуемого выполнения операций.
2. **Проблемы с affinity ядер** — при изменении частоты отдельных ядер нарушается балансировка нагрузки между потоками.
3. **Задержки переключения режимов** — переходы между энергосберегающими состояниями вводят дополнительные задержки.
4. **Непредсказуемая производительность** — операции могут выполняться с разной скоростью в зависимости от текущего режима процессора.

Симптомы проблемы:

- Нестабильное время выполнения запросов.
- Необъяснимые просадки производительности.
- Проблемы с балансировкой нагрузки между узлами кластера.
- Непредсказуемое поведение при высокой нагрузке.

Решение

Рекомендуемые настройки BIOS/UEFI

Для серверов YDB рекомендуется настроить следующие параметры BIOS/UEFI:

1. **CPU Power Management** — перевести в режим «Performance» или «Max Performance».
2. **Intel Turbo Boost** — рекомендуется включить для максимальной производительности.
3. **C-states** — отключить или ограничить до C1/C1E.
4. **P-states** — использовать фиксированную максимальную частоту.
5. **Intel SpeedStep / AMD Cool'n'Quiet** — отключить.
6. **Package C-State Limit** — установить в C0 или C1.

Настройки операционной системы

Для Linux-систем при **ручном** развёртывании (при использовании Ansible-роли это настраивается автоматически) выполните следующие команды с правами администратора:

```
# Установить governor в режим performance для всех ядер
echo 'performance' | sudo tee /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor >/dev/null

# Установить минимальную частоту равной максимальной
cat /sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_max_freq | \
sudo tee /sys/devices/system/cpu/cpu*/cpufreq/scaling_min_freq >/dev/null

# Отключить энергосбережение для PCIe
echo 'performance' | sudo tee /sys/module/pcie_aspm/parameters/policy >/dev/null
```

Проверка текущих настроек

Для проверки текущего состояния можно использовать следующие команды. После применения настроек первая команда должна выводить `performance` для каждого ядра.

```
# Проверить текущий governor
cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor

# Проверить доступные частоты
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies

# Проверить C-states
cat /sys/devices/system/cpu/cpu0/cpuidle/state*/name
```

Обоснование

YDB как высокопроизводительная распределённая СУБД требует стабильной и предсказуемой производительности процессора. Энергосберегающие режимы, разработанные для клиентских устройств, не подходят для серверных рабочих нагрузок, где важна постоянная доступность и минимальная задержка.

Стабильная частота процессора обеспечивает:

- Предсказуемое время выполнения операций.
- Корректную работу механизмов балансировки нагрузки.
- Стабильную производительность под высокой нагрузкой.
- Минимальные задержки при обработке транзакций.

Исключения

В некоторых сценариях можно рассмотреть компромиссные решения:

- **Тестовые среды** — где энергоэффективность важнее производительности.
- **Резервные узлы** — не участвующие в активной обработке.
- **Среды с низкой нагрузкой** — где экономия энергии имеет приоритет.

Однако для production-сред с высокой нагрузкой рекомендуется строго следовать приведённым рекомендациям.

Планирование ёмкости хранилища

Планирование ёмкости хранилища — это определение требуемого количества физических дисков для хранения заданного объёма данных с учётом репликации и накладных расходов, а также обратная задача: оценка полезного объёма данных для заданного количества оборудования.

Модель распределённого хранилища

На уровне оборудования кластер YDB включает в себя некоторое количество физических дисков (HDD / SSD / NVMe). Остальные компоненты (CPU, RAM, сеть) в данной статье не рассматриваются.

Кластеры YDB являются мультитенантными, и один физический диск может совместно использоваться несколькими потребителями — изолированными базами данных. Поверх каждого физического диска запущено по одному компоненту для работы с ним — PDisk. Ресурсы одного PDisk (IOPS, полоса на чтение и на запись, ёмкость) делятся на равные доли, именуемые слотами. Слоты резервируются для создания логических (виртуальных) дисков VDisk. За каждым VDisk закреплён один слот.

Для управления ресурсами пользователю доступна сущность группы хранения. Группа хранения представляет собой совокупность из нескольких VDisk на разных серверах, обеспечивающих отказоустойчивость наподобие RAID. Количество VDisk в группе хранения определяется режимом работы кластера:

Режим	Дата-центры	Стоек в дата-центре (минимум)	VDisk в группе хранения	Фактор избыточности `RF`
`block-4-2`	1	8* (рекомендуется 10+)	8	1.5
`mirror-3-dc`	3	3* (рекомендуется 4+)	9	3 (вплоть до 6**)
`mirror-3of4`	1	8* (рекомендуется 10+)	8	4
`none`	1	1	1	1

* Значения 8 для `block-4-2` и `mirror-3of4`, а также 3 для `mirror-3-dc` — это минимально необходимое количество стоек для обеспечения работоспособности кластера. При такой конфигурации кластер запускается и работает, но не имеет запаса для самовосстановления: при отказе стойки её VDisk'и некуда переселить в рамках модели отказа, и любой дополнительный сбой может привести к недоступности данных. Дополнительные стойки нужны механизму SelfHeal, чтобы перенести VDisk'и с отказавшей стойки на оставшееся оборудование с сохранением гарантий отказоустойчивости. Поэтому для практической эксплуатации рекомендуется использовать не менее 10 стоек для `block-4-2` и `mirror-3of4`, и не менее 4 стоек для `mirror-3-dc`. Подробнее о режимах хранения и гарантиях отказоустойчивости см. Топология кластера YDB.

** Для режима `mirror-3-dc` в штатном режиме `RF = 3` (3 копии данных). При недоступности одного дата-центра запись идёт в 4 копии (по 2 в оставшихся дата-центрах), и нагрузка на оставшиеся ДЦ удваивается. Если ДЦ может быть недоступен длительно (или дата-центры выводятся в обслуживание по очереди), необходимо закладывать `RF = 6`. Если допустимы только краткосрочные отказы (восстановление быстрее, чем система успевает перезаписать существенную долю данных), достаточно `RF = 3`. Подробнее о режимах хранения и гарантиях отказоустойчивости см. Топология кластера YDB.

С точки зрения пользователя группу хранения можно воспринимать как фиксированный набор ресурсов: IOPS, полоса на чтение и на запись, ёмкость. Каждой базе данных принадлежит одна или более групп хранения. Каждая группа хранения принадлежит одной определённой базе данных.

Пользовательские таблицы состоят из таблеток. Каждая таблетка управляет фрагментом пользовательских или системных данных. Таблетки хранят данные в группах, каждая таблетка может записывать данные в несколько групп хранения, в каждую группу хранения пишут множество таблеток.

Используемые величины

Tablet Storage — это суммарный объём данных, хранимый таблетками базы.

Database Storage — это суммарный размер, аллоцированный VDisk'ами базы. Этот размер включает в себя:

- собственно данные, записанные таблетками;
- избыточность при репликации или кодировании с исправлением ошибок;
- служебные метаданные и журналы;
- аллоцированное, но реально не используемое место.

Для существующих баз значения Tablet Storage и Database Storage можно посмотреть в YDB UI на странице базы на вкладке Info → Storage.

Распределённое хранилище реализует сложную отложенную сборку мусора, поэтому в любой момент времени часть занятого места приходится на устаревшие данные, ещё не удалённые сборщиком.

Другие накладные расходы могут существенно варьироваться в зависимости от характера хранимых данных и нагрузки. Для грубой оценки можно считать `Overhead = 2` — это медианное значение, полученное эмпирически на выборке продуктивных баз в режиме `mirror-3-dc`. В зависимости от режима, структуры данных и характера нагрузки накладные расходы могут существенно отличаться. Для более точной оценки рекомендуется проведение пилотного эксперимента.

Связать эти величины можно приблизительно через соотношение:

```
Database Storage = Tablet Storage × RF × Overhead
```

Во всех расчётах применяются обозначения согласно ГОСТ IEC 60027-2-2015:
1 кБ = 1000 байт, 1 МБ = 1000 кБ, 1 ГБ = 1000 МБ, 1 ТБ = 1000 ГБ;
1 КиБ = 1024 байт, 1 МиБ = 1024 КиБ, 1 ГиБ = 1024 МиБ, 1 ТиБ = 1024 ГиБ.

Оценка требуемого оборудования

Задача сводится к тому, чтобы подобрать конфигурацию кластера так, чтобы данные гарантированно поместились в группы хранения и в кластере остался рабочий запас пустых слотов для обеспечения отказоустойчивости.

Входные параметры:

- `Tablet Storage` — полезный объём данных, которые необходимо разместить.

- `DriveSize` — ёмкость используемых дисков.

Искомые параметры (конфигурация кластера):

- `NumRacks` — количество стоек / серверов (доменов отказа)
- `DisksPerRack` — количество дисков в каждом домене отказа.

Референсные значения:

- `RF` — фактор избыточности, определяется режимом работы кластера (см. таблицу выше).
- `Overhead` — накладные расходы хранилища (см. раздел выше); для грубой оценки принять `2`.
- `ExpectedSlotCount` — количество слотов на PDisk; определяется характеристиками диска — IOPS и полосой на чтение и на запись. Типовые значения: `8` для HDD, `16` для SSD/NVMe.
- `VDisksInGroup` — количество VDisk в группе хранения, определяется режимом работы кластера (см. таблицу выше).

Для оценки требуемого оборудования выполните следующие шаги.

1. Оцените требуемый объём Database Storage:

   ```
   Database Storage = Tablet Storage × RF × Overhead
   ```

2. Определите размер слота:

   ```
   SlotSize = (DriveSize - 28.08 ГБ) / ExpectedSlotCount
   ```

   Около 28.08 ГБ ёмкости PDisk зарезервировано для системных нужд, остальное место поровну распределяется между слотами.

   Данная формула применима для дисков объёмом от ~859 ГБ (800 ГиБ). Использование дисков меньшего размера не рекомендуется, если необходимо достижение оптимальной производительности. Подробнее о требованиях к дисковой подсистеме читайте в разделе Системные требования и рекомендации для YDB.

3. Оцените количество групп хранения:

   ```
   TotalGroups = ceil( Database Storage / (SlotSize × VDisksInGroup × 0.85) )
   ```

   Коэффициент 0.85 отражает рекомендуемый процент заполнения VDisk (VDisk Raw Usage = 85%).

   При мониторинге рекомендуется ориентироваться на значения VDisk Slot Usage и Capacity Alert. Для существующих групп значения VDisk Raw Usage, VDisk Slot Usage и Capacity Alert можно посмотреть в YDB UI на вкладке **Info** → **Storage** страницы базы.

   Порог, при котором группа считается переполненной, соответствует значениям VDisk Slot Usage = 100%, Capacity Alert = LightYellow, VDisk Raw Usage ≈ 90% — тогда как формула выше оставляет запас до этого порога.

4. Оцените количество занятых слотов:

   ```
   UsedSlots = TotalGroups × VDisksInGroup
   ```

5. Получите нижнюю оценку требуемого числа физических дисков (без учёта запаса на отказоустойчивость):

   ```
   TotalPDisks ≥ ceil( UsedSlots / ExpectedSlotCount )
   ```

6. Выберите конфигурацию кластера — параметры `NumRacks`, `DisksPerRack`, и определите минимально необходимый общий запас пустых слотов в кластере:

   ```
   TotalSlots    = NumRacks × DisksPerRack × ExpectedSlotCount
   MinEmptySlots = ceil( MaxSlotsInRack + 0.027 × TotalSlots )
   ```

   Запас пустых слотов необходим для нормальной работы механизма SelfHeal, выполняющего автоматическую реконфигурацию групп хранения для замены вышедших из строя или длительное время недоступных дисков.

   Первое слагаемое `MaxSlotsInRack` — максимальное количество слотов в одном домене отказа. Для однородного кластера `MaxSlotsInRack = DisksPerRack × ExpectedSlotCount`, для неоднородного — `MaxSlotsInRack = max_i(DisksPerRack_i × ExpectedSlotCount)`. Этот запас необходим, чтобы при отказе наиболее ёмкого домена его VDisk'и могли уместиться на оставшемся оборудовании.

   Второе слагаемое `0.027 × TotalSlots` — эмпирически подобранный операционный запас (~1 диск на 37), покрывающий внеплановую замену отдельных дисков.

7. Оцените соответствие конфигурации минимальным требованиям:

   ```
   EmptySlots = TotalSlots - UsedSlots
   ```

   Должно выполняться условие `EmptySlots ≥ MinEmptySlots`. Если условие не выполняется, увеличьте `NumRacks` или `DisksPerRack` и вернитесь к шагу 6. См. Пример расчёта.

Оценка полезной ёмкости хранилища

Обратная задача — оценить, какой полезный объём данных можно разместить на заданном количестве оборудования.

Пусть имеется `TotalPDisks` физических дисков ёмкостью `DriveSize` каждый. Тогда:

1. Общее количество слотов:

   ```
   TotalSlots = TotalPDisks × ExpectedSlotCount
   ```

2. Часть слотов остаются пустыми для запаса на SelfHeal, остальные доступны для групп хранения:

```
MinEmptySlots = ceil( MaxSlotsInRack + 0.027 × TotalSlots )
UsableSlots = TotalSlots - MinEmptySlots
```

   Здесь `MaxSlotsInRack` — максимальное количество слотов в одном домене отказа (стойке или сервере): для однородного кластера `DisksPerRack × ExpectedSlotCount`, для неоднородного — `max_i(DisksPerRack_i × ExpectedSlotCount)`.

3. Количество групп хранения:

```
TotalGroups = floor( UsableSlots / VDisksInGroup )
```

4. Полезная ёмкость распределённого хранилища с учётом порога заполнения 85%:

```
Database Storage = TotalGroups × VDisksInGroup × SlotSize × 0.85
```

5. Оценка полезного объёма данных:

```
Tablet Storage = Database Storage / (RF × Overhead)
```

Пример расчёта

Допустим, необходимо разместить 100 ТБ данных (Tablet Storage) в кластере с режимом отказоустойчивости `block-4-2`, используя SSD-диски ёмкостью 3.2 ТБ.

Исходные параметры:

Параметр	Значение
Tablet Storage	100 000 ГБ
Режим	`block-4-2`
RF	1.5
Overhead	2
DriveSize	3 200 ГБ
ExpectedSlotCount	16
VDisksInGroup	8

Расчёт:

1. Database Storage = 100 000 × 1.5 × 2 = **300 000 ГБ**
2. SlotSize = (3 200 − 28.08) / 16 ≈ **198.25 ГБ**
3. TotalGroups = ⌈300 000 / (198.25 × 8 × 0.85)⌉ = **223 группы хранения**
4. UsedSlots = 223 × 8 = **1 784 слота**
5. TotalPDisks (без запаса) = ⌈1 784 / 16⌉ = **112 дисков**
6. Далее выберем конкретную конфигурацию кластера, предусмотрев запас пустых слотов для обеспечения отказоустойчивости. Для этого рассмотрим различные варианты оборудования и оценим соответствие минимальным требованиям, учитывая что для режима `block-4-2` требуется минимум 8 доменов отказа (серверов или стоек), а для устойчивой практической эксплуатации рекомендуется 10 и более:
   - Возьмем 10 стоек по 12 дисков. В одной стойке 12 × 16 = 192 слота, TotalSlots = 1 920. MinEmptySlots = ⌈192 + 0.027 × 1 920⌉ = 244. EmptySlots = 1 920 − 1 784 = 136. 136 < 244 — минимальные требования не соблюдены. Несложно убедиться, что 10 стоек по 14 дисков оказывается достаточно: TotalSlots = 2 240, MinEmptySlots = ⌈224 + 0.027 × 2 240⌉ = 285, EmptySlots = 2 240 − 1 784 = 456 > 285.
   - В качестве домена отказа можно выбрать сервер и рассмотреть 30 серверов по 4 диска. В одном домене отказа 4 × 16 = 64 слота, TotalSlots = 1 920. MinEmptySlots = ⌈64 + 0.027 × 1 920⌉ = 116. EmptySlots = 1 920 − 1 784 = 136 > 116 — минимальные требования соблюдены.

Таким образом, для хранения 100 ТБ данных в режиме `block-4-2` на SSD-дисках ёмкостью 3.2 ТБ потребуется 10 стоек по 14 дисков (140 дисков) либо 30 серверов по 4 диска (120 дисков).

Обратный расчёт:

Далее возьмём конфигурацию из предыдущего примера — 30 серверов по 4 SSD-диска ёмкостью 3.2 ТБ, и оценим, какой полезный объём данных поместится в таком кластере.

1. TotalSlots = 120 × 16 = **1 920 слотов**
2. MinEmptySlots = ⌈MaxSlotsInRack + 0.027 × TotalSlots⌉ = ⌈4 × 16 + 0.027 × 1 920⌉ = ⌈115.84⌉ = **116 слотов**
3. UsableSlots = 1 920 − 116 = **1 804 слота**
4. TotalGroups = ⌊1 804 / 8⌋ = **225 групп хранения**
5. SlotSize = (3 200 − 28.08) / 16 ≈ **198.25 ГБ**
6. Database Storage = 225 × 8 × 198.25 × 0.85 ≈ **303 323 ГБ**
7. Tablet Storage = 303 323 / (1.5 × 2) ≈ **101 108 ГБ**

Таким образом, кластер из 30 серверов и 120 SSD-дисков по 3.2 ТБ в режиме `block-4-2` позволяет разместить около 101.1 ТБ данных (Tablet Storage).

Дополнительная информация

- Топология кластера YDB
- Системные требования и рекомендации для YDB

- SelfHeal хранилища

Версионирование YDB

Релизы YDB именуются версией, представляющей собой строку, состоящую из нескольких компонентов. В зависимости от контекста часть находящихся справа компонентов может быть опущена.

Упорядоченный список компонентов:

1. Две последние цифры года выпуска;
2. Порядковый номер мажорной версии внутри года;
3. Порядковый номер минорной версии внутри мажорной;
4. Номер патча внутри минорной версии;
5. Тип релиза, например `testing`, `stable` или `lts`.

Первые три компонента образуют базовую версию релиза, например `25.2.1`. Четвёртый компонент и тип уточняют конкретную сборку, например `25.1.4.7` или `25.1.4.7-stable`. Номер патча и тип не влияют на правила совместимости версий.

Таким образом, мажорные релизы обычно идентифицируются двумя компонентами, например `25.1`. Минорные релизы — тремя компонентами, например `25.1.4`. Полная версия включает номер патча и при необходимости тип релиза.

Список доступных версий можно получить на странице загрузки. Релизная политика YDB описана более детально в статье Управление релизами в разделе документации для разработчиков YDB. Схема релизных веток — в разделе Общая схема работы с ветками.

Совместимость версий

Совместимыми для обновления являются все минорные версии в рамках одной мажорной версии. Мажорные версии совместимы последовательно. Для обновления на следующую мажорную версию предварительно следует обновиться до последнего доступного минорного релиза текущей мажорной версии. Например:

- `X.Y.*.* → X.Y.*.*` — обновление возможно, все минорные версии в рамках одной мажорной совместимы.
- `X.Y.Z.*` (последняя доступная `X.Y.*.*`) → `X.Y+1.*.*` - обновление возможно, мажорные версии последовательны.
- `X.Y.*.*` → `X.Y+2.*.*` — обновление невозможно, мажорные версии непоследовательны.
- `X.Y.*.* → X.Y-2.*.*` — обновление невозможно, мажорные версии непоследовательны.

> **Важно**
>
> Также в любом случае нельзя откатываться более чем на 2 мажорные версии назад относительно хотя бы раз установленной версии, так как настолько старая версия может не знать, как работать с данными на дисках, записанными текущей.

Примеры совместимости версий

- `v.25.1.3.2` -> `v.25.1.5.5` - обновление возможно
- `v.25.1.5.5` -> `v.25.2.3.1` - обновление возможно (где `v25.1.5.*` последняя доступная минорная версия в `v.25.1`)
- `v.25.1.4.1` -> `v.25.2.3.1` - обновление невозможно, необходимо сначала обновиться до последней минорной версии (`v.25.1.5.*`)
- `v.25.1.5.5` -> `v.25.3.5.3` - обновление невозможно, необходимо сначала обновиться до следующей мажорной версии (`v.25.2.*.*`).

Формальное описание возможных версий

```
<valid-version> ::= <version-core> "-" <version-type>

<version-core> ::= <year> "." <major> "." <minor> "." <patch>

<version-type> ::= "testing" | "stable" | "lts"

<year> ::= <positive digit> <digit>

<major> ::= <positive digit>

<minor> ::= <digits>

<patch> ::= <digits>

<digit> ::= "0" | <positive digit>

<positive digit> ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

<digits> ::= <digit> | <digit> <digits>
```

Примеры полных версий

- Тестовая версия: `25.1.4.7-testing`
- Стабильная версия: `25.1.4.7-stable`
- LTS версия: `25.1.4.7-lts`

Обслуживание кластера без потери доступности

Периодически кластер YDB необходимо обслуживать, например, обновлять его версию или заменять сломавшиеся диски. Работы по обслуживанию могут привести к недоступности кластера или имеющихся баз данных из-за:

- Превышения модели отказа затронутых групп хранения.
- Превышения модели отказа State Storage.
- Недостатка вычислительных ресурсов вследствие остановки слишком большого количества динамических узлов.

Для избежания таких ситуаций в YDB есть системная таблетка, которая следит за состоянием кластера — *Cluster Management System (CMS)*. CMS позволяет ответить на вопрос, можно ли безопасно вывести в обслуживание узел YDB или хост, на котором работают узлы YDB. Для этого необходимо создать задачу обслуживания в CMS и указать в ней взятие эксклюзивных блокировок на узлы или хосты, которые будут задействованы в обслуживании. Компоненты кластера, на которые взяты блокировки, считаются недоступными с точки зрения CMS, и их можно безопасно обслуживать. CMS проверит текущее состояние кластера и возьмёт блокировки, только если работы по обслуживанию соответствуют ограничениям режима доступности и лимитам недоступных узлов.

Поломки во время проведения работ

Во время проведения работ по обслуживанию, безопасность которых гарантирована CMS, в кластере могут произойти поломки, не связанные с этими работами. Если в результате поломок доступность кластера оказалась под угрозой, то завершение работ в срочном порядке может помочь снизить риск потери доступности.

Задача обслуживания

Задача обслуживания представляет собой набор *действий*, которые пользователь просит выполнить CMS для возможности проведения безопасного обслуживания.

Поддерживаемые действия:

- Взятие эксклюзивной блокировки на компонент кластера (узел, хост или диск).

В задаче действия делятся на группы. Действия из одной группы выполняются атомарно. На данный момент группы могут состоять только из одного действия.

Если в момент запроса действие выполнить нельзя, CMS сообщает причину и время, когда стоит *обновить* задачу, и переводит действие в состояние *ожидания (pending)*. При обновлении задачи CMS повторно пытается выполнить ожидающие действия.

У *выполненных (performed)* действий есть дедлайн, после которого они считаются *завершёнными (completed)* и прекращают оказывать эффект на кластер. Например, снимается эксклюзивная блокировка. Действие можно завершить досрочно.

Затянувшиеся работы

Если работы по обслуживанию кластера продолжаются после завершения действий, которые были выполнены для безопасного проведения этих работ, то это расценивается как поломка в кластере.

Завершённые действия автоматически удаляются из задачи.

Режим доступности

В задаче обслуживания необходимо указать режим доступности кластера, который должен соблюдаться при проверке возможности выполнения действий. Поддерживаются следующие режимы:

- **Strong**: режим, минимизирующий риск потери доступности.
  - Допускается не более одного недоступного VDisk в каждой из затрагиваемых групп хранения.
  - Допускается не более одного недоступного кольца State Storage.
- **Weak**: режим, не позволяющий превысить модель отказа.
  - Допускается не более двух недоступных VDisk-ов для затрагиваемых групп хранения со схемой block-4-2.
  - Допускается не более четырёх недоступных VDisk-ов, три из которых должны находиться в одном датацентре, для затрагиваемых групп хранения со схемой mirror-3-dc.
  - Допускается не более `(nto_select - 1) / 2` недоступных колец State Storage.
- **Smart**: режим, который по умолчанию ведёт себя как **Strong**, но автоматически переключается на **Weak**, если уже есть недоступные VDisk-и в затрагиваемых группах хранения или недоступные кольца State Storage, не связанные с текущими работами по обслуживанию.
- **Force**: принудительный режим, модель отказа игнорируется. *Не рекомендуется к использованию*.

Приоритет

У задачи обслуживания можно указать приоритет. Меньшее значение означает более высокий приоритет, поддерживаются отрицательные приоритеты (по аналогии с `nice` в Linux).

Приоритет определяет порядок выполнения задач обслуживания. Задачи с более высоким приоритетом имеют преимущество перед задачами с более низким приоритетом: они блокируют их выполнение и могут выполняться поверх. Задачи с одинаковым приоритетом обрабатываются равноправно — ни одна из них не имеет преимущества перед другой.

Лимиты недоступных узлов

В CMS есть абсолютные и относительные лимиты на количество недоступных узлов для каждой базы данных (тенанта) и для кластера в целом. Лимиты недоступных узлов настраиваются в конфигурации CMS.

Алгоритм проверки

Для того чтобы проверить, можно ли выполнить действия задачи обслуживания, CMS последовательно идёт по каждой группе действий в задаче и проверяет действие из группы:

- Если объектом действия является хост, то CMS проверяет, можно ли выполнить действие со всеми узлами, запущенными на хосте.
- Если объектом действия является узел, то CMS проверяет:

    - Есть ли блокировка на данном узле.
    - Можно ли взять блокировку на данный узел в соответствии с лимитами недоступных узлов.
    - Можно ли взять блокировки на все VDisk-и данного узла в соответствии с режимом доступности.
    - Можно ли взять блокировку на кольцо State Storage данного узла в соответствии с режимом доступности.
    - Можно ли взять блокировку на данный узел в соответствии с лимитом недоступных узлов, на которых могут запускаться кластерные системные таблетки.
- Если объектом действия является диск, то CMS проверяет:
    - Есть ли блокировка на данном диске.
    - Можно ли взять блокировки на все VDisk-и данного диска в соответствии с режимом доступности.

Если проверки прошли успешно, то действие можно выполнить, и на проверенные узлы, хосты или диски берётся временная блокировка. После этого CMS рассматривает следующую группу действий. Временные блокировки позволяют понять, конфликтуют ли запрошенные в разных группах действия между собой. После полного завершения проверки временные блокировки снимаются.

Режим Bridge

Если кластер работает в режиме brigde, то для каждого pile независимо устанавливаются лимиты и выполняются проверки доступности.

Приостановка обслуживания

> **Важно**
>
> Длительная приостановка обслуживания кластера может привести к потере доступности из-за поломок оборудования.

Во внештатных ситуациях, например при возникновении проблем с инфраструктурой, можно временно приостановить новые работы по обслуживанию кластера с помощью параметра `disable_maintenance` в конфигурации CMS.

Во время приостановки обслуживания:

- Новые работы по обслуживанию кластера будут приостановлены.
- Уже начатые работы по обслуживанию кластера продолжат выполняться.
- Остается возможность выполнять экстренные работы по обслуживанию вручную без использования CMS.

Практика

- Для кластеров, развёрнутых вручную

SelfHeal

SelfHeal автоматически восстанавливает отказоустойчивость пяти типов объектов кластера YDB:

1. Динамических групп хранения.
2. Статической группы хранения.
3. Реплик State Storage.
4. Реплик Board.
5. Реплик SchemeBoard.

Внутри этих объектов работают три механизма: SelfHeal для динамических групп хранения, отдельный SelfHeal для статической группы хранения и общий SelfHeal для State Storage, Board и SchemeBoard. В документации эти механизмы сгруппированы в два раздела: SelfHeal хранилища (для динамических и статической групп) и SelfHeal подсистем распространения метаданных (для State Storage, Board и SchemeBoard).

При отказе узла или диска SelfHeal ожидает, что неисправность сохранится достаточно долго, прежде чем начать перенос. Если узел или диск восстановлен до срабатывания механизма, перенос не начинается. Для дисков время ожидания по умолчанию составляет около часа.

> **Примечание**
>
> SelfHeal динамических групп хранения от версии конфигурации не зависит. SelfHeal статической группы и подсистем распространения метаданных доступен только при конфигурации V2 и включённой распределённой конфигурации. Для статической группы также должно быть разрешено автоматическое управление с помощью параметра `automatic_static_group_management`.

Подробнее о механизмах:

- SelfHeal хранилища
- SelfHeal подсистем распространения метаданных

SelfHeal хранилища

SelfHeal хранилища — механизм автоматического восстановления отказоустойчивости хранилища YDB.

Общий обзор механизмов и условий их работы приведён в разделе SelfHeal. Восстановление реплик State Storage, Board и SchemeBoard описано в разделе SelfHeal подсистем распространения метаданных.

Как работает SelfHeal хранилища

Компонент Sentinel, входящий в CMS, постоянно следит за состоянием PDisk и узлов. Если неисправность сохраняется достаточно долго (по умолчанию около часа), Sentinel инициирует перенос затронутых VDisk на исправное оборудование, чтобы снова соблюдалась модель отказа.

Команду исполняет Blob Storage Controller: данные реплицируются в фоне. Сам перенос может занять от минут до суток в зависимости от объёма данных и оборудования. После принятия команды для CMS задача уже поставлена; завершение репликации обеспечивает распределённое хранилище.

SelfHeal хранилища включён по умолчанию для динамических групп. В кластерах с конфигурацией V2 можно также включить SelfHeal статической группы. На конфигурации V1 SelfHeal статической группы включить нельзя.

Включение и выключение SelfHeal

Вы можете включать и выключать SelfHeal с помощью утилиты YDB DSTool.

Чтобы включить SelfHeal выполните команду:

```
ydb-dstool -e <bs_endpoint> cluster set --enable-self-heal
```

`<bs_endpoint>` — эндпоинт произвольного узла хранения кластера.

Чтобы выключить SelfHeal выполните команду:

```
ydb-dstool -e <bs_endpoint> cluster set --disable-self-heal
```

Когда отключать SelfHeal

Обычно SelfHeal оставляют включённым. Временно отключить его имеет смысл только тогда, когда автоматический перенос опаснее ожидания, например, если:

- обнаружена ошибка в SelfHeal, из-за которой перенос создаёт риск потери данных;
- одновременно отказало много узлов, кластер перегружен, а дополнительная фоновая репликация увеличит нагрузку и помешает оператору восстановить работоспособность кластера.

> **Важно**
>
> Пока SelfHeal отключён, VDisk с неисправных PDisk автоматически не переносятся. Контролируйте состояние хранилища и снова включите SelfHeal сразу после стабилизации кластера.

Настройки SelfHeal

Параметры ниже управляют разными этапами работы Sentinel: опросом состояния PDisk, подтверждением устойчивого состояния и повторами при передаче нового статуса в Blob Storage Controller. Для каждого состояния время до его подтверждения определяется произведением **State update interval** и лимита циклов для этого состояния. Например, для большинства отказных состояний значения по умолчанию составляют 60 секунд и 60 циклов, поэтому переход в `FAULTY` начинается примерно через час. Интервалы обновления конфигурации и повторов смены статуса в это произведение не входят.

> **Важно**
>
> В штатной эксплуатации менять эти параметры не требуется. Значения по умолчанию подобраны для типичных кластеров. Меняйте их только если понимаете, как сдвиг задержки повлияет на скорость реакции SelfHeal, например по рекомендации разработчиков YDB.

Вы можете настроить SelfHeal в **Viewer** → **Cluster Management System** → **CmsConfigItems**.

Чтобы создать настройки впервые, нажмите кнопку **Create**. Если вам нужно изменить существующие настройки, нажмите кнопку ✎.

Доступны следующие настройки:

Параметр	Описание
Status	Включение и выключение SelfHeal в CMS.
Dry run	Включение и выключение режима, в котором CMS не изменяет настройку BSC.
Config update interval (sec.)	Период обновления конфигурации из BSC.
Retry interval (sec.)	Период повторных попыток обновления конфигурации.
State update interval (sec.)	Период обновления состояний PDisk'ов.
Timeout (sec.)	Таймаут обновления состояний PDisk'ов.
Change status retries	Количество повторных попыток на изменение статуса PDisk в BSC (`ACTIVE`, `FAULTY`, `BROKEN` и др.).

Change status retry interval (sec.)	Задержка между повторными попытками передать новый статус PDisk в BSC.
Default state limit	Для состояний, для которых не указана настройка, может использоваться это значение «по умолчанию». Для неизвестных состояний PDisk, для которых нет настройки, тоже используется это значение. Это значение используется, если значение не задано для состояний `Initial`, `InitialFormatRead`, `InitialSysLogRead`, `InitialCommonLogRead`, `Normal`.
Initial	PDisk начинает инициализацию. Переход в `FAULTY`.
InitialFormatRead	PDisk читает свою запись формата. Переход в `FAULTY`.
InitialFormatReadError	PDisk получил ошибку при чтении своей записи формата. Переход в `FAULTY`.
InitialSysLogRead	PDisk читает системный лог. Переход в `FAULTY`.
InitialSysLogReadError	PDisk получил ошибку при чтении системного лога. Переход в `FAULTY`.
InitialSysLogParseError	PDisk получил ошибку при парсинге или проверке консистентности системного лога. Переход в `FAULTY`.
InitialCommonLogRead	PDisk читает общий лог VDisk'ов. Переход в `FAULTY`.
InitialCommonLogReadError	PDisk получил ошибку при чтении общего лога VDisk'ов. Переход в `FAULTY`.
InitialCommonLogParseError	PDisk получил ошибку при парсинге или проверке консистентности общего лога. Переход в `FAULTY`.
CommonLoggerInitError	PDisk получил ошибку при инициализации внутренних структур предназначенных для записи в общий лог. Переход в `FAULTY`.
Normal	PDisk завершил инициализацию и работает нормально. Переход в `ACTIVE` произойдет через указанное количество циклов (например, если `Normal` держится 5 минут, диск переходит в состояние `ACTIVE`).
OpenFileError	PDisk получил ошибку при открытии файла диска. Переход в `FAULTY`.
Missing	Нода отвечает, но в её списке нет данного PDisk. Переход в `FAULTY`.
Timeout	Нода не ответила за отведенный таймаут. Переход в `FAULTY`.
NodeDisconnected	Отключение ноды. Переход в `FAULTY`.
Stopped	PDisk остановлен. Переход в `FAULTY`.
Unknown	Неожиданный ответ, например, ответ `TEvUndelivered` на запрос состояния. Переход в `FAULTY`.

Работа с дисками-донорами

Диск-донор — это предыдущий VDisk после переноса данных, который продолжает хранить свои данные и отвечает только на запросы чтения от нового VDisk'а. При переносе с включенными дисками-донорами предыдущие VDisk'и продолжают функционировать до тех пор, пока данные не будут полностью перенесены на новые диски. Чтобы предотвратить потерю данных при переносе VDisk'а, включите возможность использования дисков-доноров:

```
ydb-dstool -e <bs_endpoint> cluster set --enable-donor-mode
```

Чтобы выключить диски-доноры, введите команду:

```
ydb-dstool -e <bs_endpoint> cluster set --disable-donor-mode
```

SelfHeal подсистем распространения метаданных

SelfHeal подсистем распространения метаданных — механизм автоматического управления конфигурациями реплик State Storage, Board и SchemeBoard.

В конфигурации параметры механизма используют историческое имя `state_storage_self_heal_config`, которое относится ко всем трём подсистемам.

> **Важно**
>
> Инструкция относится только к кластерам YDB с **конфигурацией V2**. На кластерах с **конфигурацией V1** эти шаги и команды (в том числе получение конфигурации через `ydb admin cluster config fetch`) недоступны или не дадут ожидаемого результата. Альтернатив для V1 здесь не приводится — см. Миграция на конфигурацию V2.

Механизм обнаруживает неисправности узлов и, если их нельзя быстро восстановить, переносит затронутые реплики на другие узлы. При росте кластера он также может автоматически увеличивать число реплик с учётом конфигурации и доступных узлов.

Механизм запускается Sentinel — компонентом системы управления кластером CMS.

Включение и выключение SelfHeal подсистем распространения метаданных

Вы можете включать и выключать SelfHeal подсистем распространения метаданных с помощью изменения конфигурации:

1. Получите текущую конфигурацию кластера с помощью команды ydb admin cluster config fetch:

   ```
   ydb [global options...] admin cluster config fetch > config.yaml
   ```

2. Измените конфигурационный файл `config.yaml`. Для этого поменяйте значение параметра `state_storage_self_heal_config.enable` на `true` или на `false`.

   Для работы SelfHeal необходимо включить Sentinel: установите параметр `cms_config.sentinel_config.enable` в значение `true` в секции `config` файла `config.yaml`. Примените изменения на шаге 3.

   Пример конфигурации с включёнными Sentinel и SelfHeal:

   ```
   config:
       self_management_config:
           enabled: true # Включение распределённой конфигурации
       cms_config:
           sentinel_config:
               enable: true # Включение Sentinel
               state_storage_self_heal_config:
                   enable: true # Включение SelfHeal подсистем распространения метаданных
   ```

   > **Примечание**
   >
   > Для работы механизма требуется конфигурация V2 и включённый CMS Sentinel.
   >
   > Наличие `self_management_config.enabled: true` косвенно указывает на использование конфигурации V2. Установка этого флага сама по себе не переводит кластер на конфигурацию V2. Для перехода выполните шаги инструкции Миграция на конфигурацию V2.

   При значении `true` у параметра `state_storage_self_heal_config.enable` включается механизм сохранения работоспособности и отказоустойчивости State Storage, Board, SchemeBoard.
3. Обновите конфигурацию кластера с учетом выполненных изменений с помощью ydb admin cluster config replace:

   ```
   ydb [global options...] admin cluster config replace -f config.yaml
   ```

Управление автоматическим изменением конфигурации

Помимо общего включения/выключения SelfHeal подсистем распространения метаданных (параметр `state_storage_self_heal_config.enable`), в секции `self_management_config` конфигурационного файла `config.yaml` вы можете по отдельности управлять автоматическим изменением конфигурации каждой из подсистем распространения метаданных, а также ограничивать множество узлов, на которые SelfHeal может переносить реплики.

```
config:
    self_management_config:
        enabled: true
        automatic_state_storage_management: true
        automatic_state_storage_board_management: true
        automatic_scheme_board_management: true
        state_storage_self_heal_allowed_nodes: [1, 2, 3, 4, 5, 6, 7, 8]
        state_storage_board_self_heal_allowed_nodes: [1, 2, 3, 4, 5, 6, 7, 8]
        scheme_board_self_heal_allowed_nodes: [1, 2, 3, 4, 5, 6, 7, 8]
```

Параметр	Значение по умолчанию	Описание
`automatic_state_storage_management`	`true`	Разрешает SelfHeal автоматически изменять конфигурацию State Storage. При значении `false` SelfHeal не изменяет текущую конфигурацию State Storage.

`automatic_state_storage_board_management`	`true`	То же самое для Board: разрешает или запрещает SelfHeal автоматически изменять его конфигурацию.
`automatic_scheme_board_management`	`true`	То же самое для SchemeBoard: разрешает или запрещает SelfHeal автоматически изменять его конфигурацию.
`state_storage_self_heal_allowed_nodes`	`[]` (без ограничений)	Список идентификаторов узлов, на которые SelfHeal может переносить или на которых может добавлять реплики State Storage. Пустой список означает, что ограничений нет и могут быть использованы любые узлы кластера.
`state_storage_board_self_heal_allowed_nodes`	`[]` (без ограничений)	То же самое для реплик Board.
`scheme_board_self_heal_allowed_nodes`	`[]` (без ограничений)	То же самое для реплик SchemeBoard.

Дополнительные параметры SelfHeal подсистем распространения метаданных

В секции `cms_config.sentinel_config.state_storage_self_heal_config` конфигурационного файла `config.yaml` вы можете настроить дополнительные параметры работы механизма SelfHeal подсистем распространения метаданных. Они влияют на то, как быстро реагирует механизм на изменения и сколько реплик подсистем распространения метаданных создаётся. В примере ниже все параметры показаны со значениями по умолчанию:

```yaml
config:
    cms_config:
        sentinel_config:
            enable: true
            state_storage_self_heal_config:
                enable: true
                wait_for_config_step: 60000000
                relax_time: 600000000
                pileup_replicas: false
                override_replicas_in_ring_count: 0
                override_rings_count: 0
                replicas_specific_volume: 200
```

Параметр	Значение по умолчанию	Описание
`wait_for_config_step`	`60000000` (микросекунды, 60 секунд)	Время ожидания между промежуточными шагами применения новой конфигурации подсистем распространения метаданных (добавление/удаление групп колец, снятие флага `WriteOnly`, см. Конфигурирование State Storage). Значение задаётся в микросекундах.
`relax_time`	`600000000` (микросекунды, 600 секунд)	Минимальный интервал между двумя последовательными срабатываниями SelfHeal подсистем распространения метаданных. Пока не прошло указанное время с момента предыдущего срабатывания, повторное изменение конфигурации не запускается, даже если обнаружены неисправные узлы. Значение задаётся в микросекундах.
`pileup_replicas`	`false`	Разрешает размещать реплики разных подсистем (State Storage, Board, SchemeBoard) на одном и том же наборе узлов. При значении `false` SelfHeal старается использовать разные узлы для реплик разных подсистем там, где это возможно; при значении `true` узлы, уже занятые под одну подсистему, могут повторно использоваться для остальных.
`override_replicas_in_ring_count`	`0` (рассчитывается автоматически)	Принудительно задаёт количество реплик в одном кольце. Если значение `0`, количество реплик в кольце вычисляется автоматически на основе `replicas_specific_volume` и числа доступных узлов.
`override_rings_count`	`0` (рассчитывается автоматически)	Принудительно задаёт количество колец в конфигурации. Если значение `0`, количество колец вычисляется автоматически на основе числа доступных узлов и топологии кластера.
`replicas_specific_volume`	`200`	Определяет, сколько узлов кластера должно приходиться на одну дополнительную реплику в кольце: одна дополнительная реплика добавляется на каждые `replicas_specific_volume` узлов в кластере. Используется при автоматическом расчёте количества реплик, если `override_replicas_in_ring_count` не задан (равен `0`).

Проверка результата

Проверить, что изменения применились, можно в разделе `CMS` в YDB UI кластера (доступен на порту 8765): перейдите на вкладку `Sentinel` для просмотра статуса Sentinel и SelfHeal подсистем распространения метаданных.

Настройка аутентификации и авторизации узлов баз данных

Аутентификация узлов в кластере YDB обеспечивает проверку подлинности узлов баз данных при выполнении служебных вызовов к другим узлам по протоколу gRPC. Авторизация узлов обеспечивает проверку и предоставление необходимых полномочий при обработке служебных вызовов, включая операции регистрации запускаемых узлов в кластере и доступа к управлению конфигурацией. Использование аутентификации и авторизации узлов рекомендовано для всех кластеров YDB, поскольку позволяет избежать ситуаций несанкционированного доступа к данным через включение в кластер контролируемых злоумышленниками узлов.

Прежде чем динамический узел базы данных войдёт в состав кластера и начнёт обмен с другими узлами по внутреннему протоколу Interconnect, он должен зарегистрироваться у уже работающих узлов хранения. Для этого запускаемый узел устанавливает gRPC-соединение с одним из узлов хранения, указанных в опции `--node-broker`, и по этому соединению выполняет служебные вызовы — в том числе запрос на регистрацию. Именно такие gRPC-вызовы и защищаются аутентификацией и авторизацией узлов: без успешной проверки узел не получает право зарегистрироваться и не может войти в кластер.

Что нужно знать перед тем как начать

Идентификаторы узлов:

- **NodeId** — уникальный числовой идентификатор узла в кластере (от 1 до 1048575).
- **SID** (Subject IDentifier) — уникальный идентификатор субъекта доступа, определяющий, какие операции может выполнять узел (например, `registerNode@cert` означает «узел с правом регистрации»).

Цель аутентификации. Убедиться, что новый узел БД - это авторизованный узел. Для этого используются сертификаты.

Как проходит проверка узла

1. Запускаемый узел базы данных открывает gRPC-подключение к одному из узлов хранения, указанных в опции `--node-broker`. Подключение выполняется по TLS, а в качестве клиентского сертификата используется сертификат запускаемого узла, заданный в опции `--grpc-cert`.
2. Узел хранения и узел базы данных выполняют mTLS : каждый узел проверяет сертификат другого. Проверяется цепочка доверия (сертификат подписан известным центром сертификации) и соответствие имени хоста значению поля "Subject Name" сертификата.
3. Узел хранения проверяет поле "Subject" сертификата на соответствие требованиям, заданным в статической конфигурации в секции client_certificate_authorization. Таким образом узел хранения убеждается, что сертификат издан именно для узлов вашего кластера. Например, вы можете требовать, чтобы все сертификаты узлов имели организацию (O) = "MyCompany", что затруднит подделку сертификата злоумышленникам.
4. Если все проверки успешны (сертификат подлинный, цепочка доверия в порядке, поле Subject соответствует требованиям), узел хранения доверяет подключению и присваивает ему идентификатор субъекта доступа (SID) — строку типа registerNode@cert, указанную в конфигурации в параметре member_groups. SID определяет, какие операции может выполнять узел (в данном случае — регистрироваться в кластере).
5. Узел базы данных использует доверенное gRPC-подключение, чтобы «представиться» узлу хранения. Узел БД посылает служебный запрос, в котором передаёт IP-адрес и порт, по которому будут принимать соединения от других узлов кластера. Узел хранения запоминает полученную информацию — это и называется регистрацией в кластере.
6. Узел хранения проверяет: имеет ли SID, присвоенный подключению на этапе mTLS, право на регистрацию узлов. Для этого узел хранения смотрит на параметр конфигурации `register_dynamic_node_allowed_sids` — это список SID, которым разрешена регистрация новых узлов БД в кластере. Если SID узла (например, `registerNode@cert`) содержится в этом списке — регистрация разрешена. Если нет — узел отвергается как неавторизованный.

   После успешной проверки узел хранения выполняет три действия:

   - Присваивает узлу БД уникальный числовой идентификатор — **NodeId** (например, NodeId=50001).
   - Записывает соответствие: NodeId=50001 → IP-адрес 192.168.1.100:19002 в служебный реестр узлов кластера.
   - Этот реестр используется другими узлами кластера для маршрутизации (куда направить запрос узлу с NodeId=50001).
7. Теперь узел БД готов к полноценной работе в кластере. Узел БД подключается к другим узлам в кластере через служебный протокол Interconnect и передаёт свой назначенный NodeId (например, 50001) и свой сетевой адрес (host:port). Другие узлы кластера используют эту информацию для маршрутизации запросов. Если какой-то узел попытается подключиться с неизвестным NodeId или IP-адресом (которого нет в реестре), другие узлы его отвергают.

Режим аутентификации узлов при регистрации задаётся параметром node_registration_token в секции `auth_config`. По умолчанию аутентификация выключена. Такой режим допустим только в полностью изолированных тестовых окружениях.

Настройка аутентификации и авторизации узлов баз данных

Чтобы включить аутентификацию и авторизацию узлов, перед изменением конфигурации кластера необходимо подготовить инфраструктуру сертификатов и настроить запуск узлов с использованием TLS-сертификатов.

1. Убедитесь, что в развернутом кластере YDB настроено шифрование gRPC-трафика с использованием TLS.
2. При выпуске сертификатов для узлов задайте единые правила заполнения поля "Subject", чтобы можно было отличать сертификаты узлов кластера. Подробности — в документации по правилам проверки сертификатов.

> **Примечание**
>
> Пример скрипта генерации самоподписанных сертификатов для узлов YDB заполняет поле "Subject" значением `O=YDB` для всех узлов. Примеры в этой статье рассчитаны именно на такой вариант заполнения поля "Subject".

3. В параметры командной строки для запуска узлов баз данных добавьте опции с путями к сертификатам и ключу:

Опция командной строки	Описание
`--grpc-ca`	Путь к файлу сертификатов доверенных центров сертификации `ca.crt`.
`--grpc-cert`	Путь к файлу сертификата узла `node.crt`.
`--grpc-key`	Путь к файлу секретного ключа узла `node.key`.

Пример запуска узла базы данных:

```
/opt/ydb/bin/ydbd server --yaml-config  /opt/ydb/cfg/config.yaml --tenant /Root/testdb \
    --grpcs-port 2136 --grpc-ca /opt/ydb/certs/ca.crt \
    --grpc-cert /opt/ydb/certs/node.crt --grpc-key /opt/ydb/certs/node.key \
    --ic-port 19002 --ca /opt/ydb/certs/ca.crt \
    --mon-port 8766 --mon-cert /opt/ydb/certs/web.pem \
    --node-broker grpcs://<ydb1>:2135 \
    --node-broker grpcs://<ydb2>:2135 \
    --node-broker grpcs://<ydb3>:2135
```

Включение режима аутентификации и авторизации узлов

Чтобы включить обязательную авторизацию узлов баз данных, обновите статическую конфигурацию кластера:

1. На корневом уровне добавьте блок `client_certificate_authorization` с требованиями к полю "Subject" сертификатов подключаемых узлов.

   ```
   client_certificate_authorization:
     request_client_certificate: true
     client_certificate_definitions:
       - member_groups: ["registerNode@cert"]
         subject_terms:
           - short_name: "O"
             values: ["YDB"]
   ```

   При необходимости добавьте дополнительные проверки сертификатов согласно документации по `client_certificate_authorization`.

   В случае успешной проверки сертификата и соответствия компонентов его поля "Subject" требованиям, заданным в `subject_terms`, подключению будут присвоены субъекты доступа (SID), перечисленные в `member_groups`. Чтобы отделять такие субъекты от других групп и учётных записей, необходимо их именовать с добавлением суффикса `@cert`.
2. В блоке `security_config` добавьте параметр `register_dynamic_node_allowed_sids` со списком SID, которым разрешена регистрация узлов баз данных. Пример:

   ```
   domains_config:
     ...
     security_config:
       enforce_user_token_requirement: true
       ...
       register_dynamic_node_allowed_sids:
         - "root@builtin"
         - "registerNode@cert"
   ```

   Включение `root@builtin` в этот список обязательно по техническим причинам: некоторые внутренние процессы кластера используют его независимо от режима аутентификации узлов.

   Подробности о параметрах аутентификации кластера см. в документации по `security_config`.
3. В блоке `auth_config` присвойте параметру `node_registration_token` пустую строку:

   ```
   auth_config:
     ...
     node_registration_token: ""
   ```

   Подробности о параметре `node_registration_token` см. в документации по `auth_config`.
4. Обновите статическую конфигурацию на всех узлах кластера вручную или через плейбук Ansible.
5. Выполните поэтапный перезапуск узлов хранения с помощью ydbops или через плейбук Ansible.
6. Выполните поэтапный перезапуск узлов баз данных кластера с помощью ydbops или плейбука Ansible.

Конфигурация кластеров YDB

В данном разделе представлены материалы по управлению конфигурацией кластеров YDB. Вы узнаете о двух версиях механизма конфигурации (V1 и V2), их различиях, а также о том, как проверить, какая версия используется в вашем кластере.

Основные подразделы:

- Конфигурация V1 — раздел о конфигурации V1
- Конфигурация V2 — раздел о конфигурации V2, экспериментальной функциональности для версий YDB 25.1 и выше.
- Миграция конфигурации кластера — миграция между конфигурациями V1 и V2.
- Проверка версии конфигурации — проверка, какая версия конфигурации используется на кластере.
- Сравнение конфигураций кластера YDB: V1 и V2 — детальное сравнение конфигураций V1 и V2.

Совет

Новые кластеры YDB рекомендуется разворачивать сразу с использованием конфигурации V2. Если кластер был развёрнут на конфигурации V1, то она продолжит использоваться даже после обновления на версию YDB 25.1 или выше. После такого обновления рекомендуется запланировать и провести миграцию на V2, так как в будущих версиях YDB поддержка V1 будет прекращена. Узнать, какая версия конфигурации используется на кластере, можно по инструкции.

Конфигурация V1

В этом разделе документации YDB описана Конфигурация V1, являющаяся основным способом конфигурирования кластеров YDB.

Конфигурация V1 — двухуровневая система конфигурации кластера YDB, состоящая из статической конфигурации и динамической конфигурации:

1. **Статическая конфигурация**: файл в формате YAML, который располагается локально на каждом статическом узле и используется при запуске процесса `ydbd server`. Эта конфигурация содержит, в том числе, настройки статической группы и State Storage.
2. **Динамическая конфигурация**: файл в формате YAML, являющийся расширенной версией статической конфигурации. Загружается через CLI и надёжно сохраняется в таблетке Console, которая затем распространяет конфигурацию на все динамические узлы кластера. Использование динамической конфигурации опционально.

Подробнее о Конфигурации V1 можно узнать в разделе Обзор конфигурации V1.

Основные материалы:

- Обзор конфигурации V1
- Статическая конфигурация
- Динамическая конфигурация кластера
- Временные конфигурации
- DSL конфигурация кластера
- Изменение конфигураций через CMS
- Изменение конфигурации актор-системы
- Расширение кластера
- Конфигурирование подсистем распространения метаданных State Storage, Board, Scheme Board
- Перемещение State Storage
- Перемещение статической группы
- Замена FQDN узла
- Аутентификация и авторизация узлов баз данных

Обзор конфигурации V1

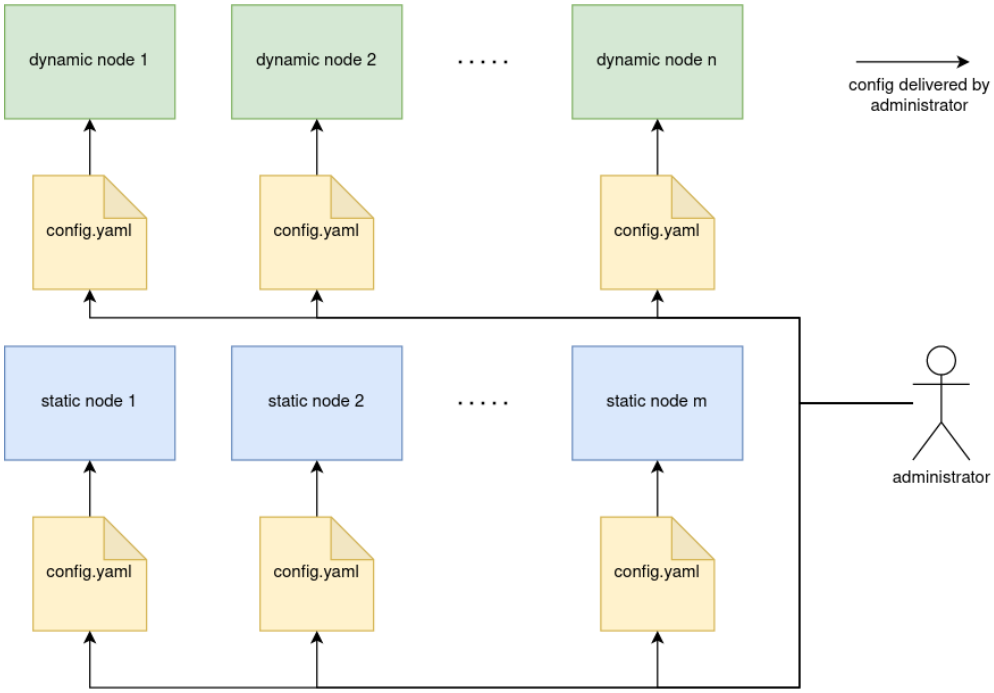
Для запуска узла YDB требуется конфигурация. Существуют два типа конфигурации:

- **Статическая** — файл в формате YAML, хранящийся на локальном диске узла.
- **Динамическая** — документ в формате YAML, хранящийся в хранилище конфигурации YDB .

Статические узлы кластера используют статическую конфигурацию. Динамические узлы могут использовать статическую конфигурацию, динамическую конфигурацию или их комбинацию.

Статическая конфигурация

Статическая конфигурация представляет собой YAML файл, хранимый на узлах кластера. В этом файле перечислены все настройки системы. Путь к файлу передается на вход процессу `ydbd` при запуске через параметр командной строки. Распространение статической конфигурации по кластеру и поддержка её в консистентном состоянии на всех узлах — ответственность администратора кластера. Подробности по использованию статической конфигурации можно найти в разделе Параметры конфигурации кластера. Эта конфигурация **необходима** для запуска статических узлов.

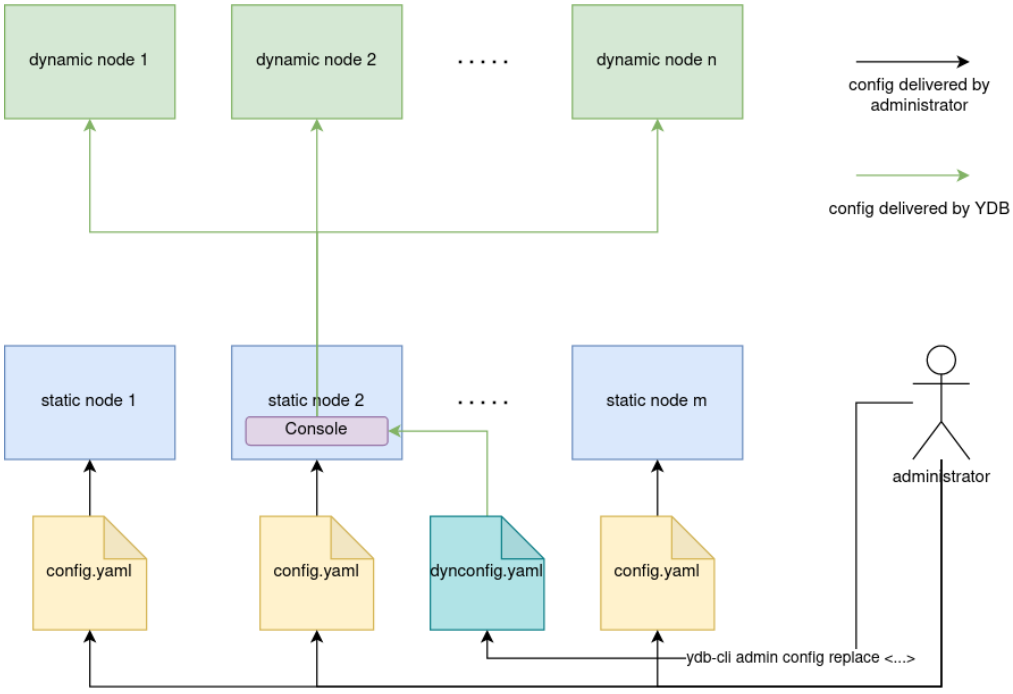


Базовый сценарий использования

1. Скопировать стандартную конфигурацию из GitHub.
2. Изменить конфигурацию в соответствии с вашими требованиями.
3. Разместить идентичные файлы конфигурации на всех узлах кластера.
4. Запустить все узлы кластера, указав путь к файлу конфигурации явно, используя аргумент командной строки `--yaml-config`.

Динамическая конфигурация

Динамическая конфигурация является YAML-документом, надёжно сохранённом в кластере в таблетке Console. В отличие от статической её достаточно загрузить в кластер, так как за её распространение и поддержание в консистентном состоянии будет отвечать YDB. При этом динамическая конфигурация при помощи селекторов позволяет обрабатывать, в том числе, сложные сценарии, оставаясь при этом в рамках одного файла конфигурации. Описание динамической конфигурации представлено в разделе Динамическая конфигурация кластера.



Базовый сценарий использования

1. Скопировать стандартную конфигурацию из GitHub.
2. Изменить конфигурацию в соответствии с вашими требованиями.
3. Разместить идентичные файлы конфигурации на всех статических узлах кластера.
4. Запустить все статические узлы кластера, указав путь к файлу конфигурации явно, используя аргумент командной строки `--yaml-config`.
5. Дополнить файл конфигурации до формата динамической конфигурации.
6. Загрузить на кластер полученную конфигурацию при помощи `ydb admin config replace -f dynconfig.yaml`.

Параметры конфигурации кластера

Конфигурация кластера задается в YAML-файле, который передается в параметре `--yaml-config` при запуске узлов кластера. В данной статье приведено описание основных разделов конфигурации и ссылки на подробную документацию по каждому разделу.

Каждый раздел конфигурации служит определенной цели в настройке работы кластера YDB, от распределения аппаратных ресурсов до настроек безопасности и функциональных флагов. Конфигурация организована в логические группы, соответствующие различным аспектам управления кластером и его работы.

Разделы конфигурации

Доступны следующие разделы конфигурации, расположенные в алфавитном порядке:

Раздел	Обязателен	Описание
actor_system_config	Да	Распределение CPU-ресурсов по пулам акторной системы
auth_config	Нет	Настройки аутентификации и авторизации
aws_client_config	Нет	Настройки по умолчанию клиента AWS
blob_storage_config	Нет	Конфигурация статической группы кластера для системных таблеток
bridge_config	Нет	Конфигурация режима bridge
client_certificate_authorization	Нет	Аутентификация с помощью клиентских сертификатов
cms_config	Нет	Конфигурация Cluster Management System (CMS)
domains_config	Нет	Конфигурация домена кластера, включая Blob Storage и State Storage
feature_flags	Нет	Функциональные флаги для включения или отключения определённых возможностей YDB
health_check_config	Нет	Пороговые значения и таймауты сервиса Health Check
hive_config	Нет	Конфигурация запуска таблеток
host_configs	Нет	Типовые конфигурации хостов для узлов кластера
hosts	Да	Конфигурация статических узлов кластера
immediate_controls_config	Нет	Конфигурация динамических настроек кластера
kafka_proxy_config	Нет	Конфигурация Kafka Proxy
log_config	Нет	Конфигурация и параметры логирования
memory_controller_config	Нет	Распределение памяти и лимиты для компонентов базы данных
monitoring_config	Нет	Параметры YDB Monitoring
node_broker_config	Нет	Конфигурация стабильных имен узлов
query_service_config	Нет	Конфигурация внешних источников для федеративных запросов
resource_broker_config	Нет	Брокер ресурсов для контроля потребления CPU и памяти
security_config	Нет	Настройки конфигурации безопасности
self_management_config	Нет	Настройки распределённой конфигурации V2 и автоматического управления компонентами кластера
system_tablet_backup_config	Нет	Конфигурация резервного копирования системных таблеток
table_service_config	Нет	Настройки конфигурации выполнения запросов
tli_config	Нет	Параметры диагностики инвалидации блокировок транзакций (TLI)
tls	Нет	Конфигурация TLS для безопасных соединений

Практические рекомендации

Этот раздел документации посвящён полному описанию доступных настроек, а практические рекомендации по тому, что и когда настраивать, можно найти в следующих местах:

- В рамках первоначального развёртывания кластера YDB:
- Ansible
- Kubernetes
- Вручную
- В рамках поиска и устранения неисправностей
- В рамках усиления безопасности

Примеры конфигураций кластеров

Модельные конфигурации кластера для развертывания можно найти в репозитории. Изучите их перед развертыванием кластера.

Динамическая конфигурация кластера

Динамическая конфигурация позволяет запускать динамические узлы, сконфигурировав их централизованно, без необходимости раскладывать файлы по узлам вручную. YDB выступает в роли системы управления конфигурациями, предоставляя инструменты для надёжного хранения, версионирования и доставки конфигурации, а так же DSL (Domain Specific Language) для переопределения её частей для определённых групп узлов. Конфигурация представляет собой YAML документ. Он является расширенной версией статической конфигурации:

- описание конфигурации вынесено в поле `config`;
- добавлено поле `metadata`, необходимое для валидации и версионирования;
- добавлены поля `allowed_labels` и `selector_config` для гранулярного переопределения настроек.

Эта конфигурация загружается в кластер, где надёжно сохраняется и доставляется до каждого динамического узла при его старте. Некоторые настройки обновляются на лету, без перезапуска узлов. С помощью динамической конфигурации можно централизованно решить следующие задачи:

- переключать настройки журналирования компонентов как для всего кластера, так и для отдельных баз данных или групп узлов;
- включать экспериментальную функциональность (фича-флаги) на отдельных базах данных;
- изменять настройки акторной системы на конкретной базе данных, отдельном узле или на группе узлов.

Подготовка к использованию динамической конфигурации

Перед началом использования динамической конфигурации в кластере необходимо провести подготовительные работы:

1. Включите в кластере аутентификацию и авторизацию узлов баз данных.
2. Если ранее в кластере использовалось управление конфигурацией через CMS, выгрузите существующие настройки в формате YAML. Для этого выполните следующую команду:

   ```
   ./ydbd -s grpcs://<node1.ydb.tech>:2135 --ca-file ca.crt --token-file ydbd-token \
        admin console configs dump-yaml > dynconfig.yaml
   ```

   Предварительно необходимо получить токен аутентификации с помощью команды `ydb auth get-token`, аналогично процедуре инициализации кластера.
3. Сформируйте первоначальный файл динамической конфигурации:
   - Если ранее в CMS выполнялись настройки (экспортированные на предыдущем шаге), используйте полученный файл в качестве основы и:
     - добавьте секцию `metadata` по образцу из примера конфигурации;
     - в секцию `config` добавьте параметр `yaml_config_enabled: true`.
   - Если ранее настройки через CMS не выполнялись, используйте минимальное наполнение файла динамической конфигурации.
   - Если в кластере используется шифрование интерконнекта акторной системы, добавьте в секцию `config` соответствующие настройки TLS для интерконнекта.
4. Примените сформированный файл динамической конфигурации на кластер:

   ```
   # Применить конфигурационный файл dynconfig.yaml на кластер
   ydb admin config replace -f dynconfig.yaml
   ```

> **Примечание**
>
> После включения поддержки динамической конфигурации в кластере YDB устаревшая функция управления конфигурацией через CMS станет недоступной.

Примеры конфигурации

Пример минимальной динамической конфигурации для однодатацентрового кластера:

```
# Метаданные конфигурации.
# Поле управляется сервером
metadata:
  # Имя кластера из параметра cluster_uuid, выставляемом при установке кластера, или "", если параметр не выс
тавлен
  cluster: ""
  # Идентификатор конфигурационного файла, всегда возрастает на 1 и начинается с 0.
  # Увеличивается автоматически при загрузке новой конфигурации на сервер.
  version: 0
# Основная конфигурация кластера, все значения из него применяются по-умолчанию, пока не переопределены селек
торами.
# Содержание аналогично статической конфигурации кластера
config:
  # должен быть всегда выставлен в true для использования yaml конфигурации
  yaml_config_enabled: true
  # конфигурация актор-системы - поскольку по-умолчанию данная секция используется
  # только узлами БД, выставлена конфигурация именно для них
  actor_system_config:
    # автоматический подбор конфигурации для ноды на основе типа и количества доступных ядер
    use_auto_config: true
    # HYBRID || COMPUTE || STORAGE — тип ноды, для узлов баз данных всегда COMPUTE
    node_type: COMPUTE
    # количество выделенных ядер
    cpu_count: 14
allowed_labels: {}
selector_config: []
```

Подробнее параметры конфигурации описаны на странице Параметры конфигурации кластера.

Первоначально установленная динамическая конфигурация кластера получает номер версии 1. При применении новой конфигурации версия хранимой конфигурации сравнивается с указанной в YAML-документе и автоматически увеличивается на единицу.

Пример более сложной динамической конфигурации с установкой типовых глобальных параметров, а также особых параметров для одной из баз данных:

```
---
metadata:
  kind: MainConfig
  cluster: ""
  version: 1
config:
  yaml_config_enabled: true
  table_profiles_config:
    table_profiles:
    - name: default
      compaction_policy: default
      execution_policy: default
      partitioning_policy: default
      storage_policy: default
      replication_policy: default
      caching_policy: default
    compaction_policies:
    - name: default
    execution_policies:
    - name: default
    partitioning_policies:
    - name: default
      auto_split: true
      auto_merge: true
      size_to_split: 2147483648
    storage_policies:
    - name: default
      column_families:
      - storage_config:
          sys_log:
            preferred_pool_kind: ssd
          log:
            preferred_pool_kind: ssd
          data:
            preferred_pool_kind: ssd
    replication_policies:
    - name: default
    caching_policies:
    - name: default
  interconnect_config:
    encryption_mode: REQUIRED
    path_to_certificate_file: "/opt/ydb/certs/node.crt"
    path_to_private_key_file: "/opt/ydb/certs/node.key"
    path_to_ca_file: "/opt/ydb/certs/ca.crt"
allowed_labels:
  node_id:
    type: string
  host:
    type: string
  tenant:
    type: string
selector_config:
- description: Custom settings for testdb
  selector:
    tenant: /cluster1/testdb
  config:
    shared_cache_config:
      memory_limit: 34359738368
    feature_flags: !inherit
      enable_views: true
    actor_system_config:
      use_auto_config: true
      node_type: COMPUTE
      cpu_count: 14
```

Обновление динамической конфигурации

```
# Получить конфигурацию кластера
ydb admin config fetch > dynconfig.yaml
# Отредактировать при помощи любого текстового редактора
vim dynconfig.yaml
# Применить конфигурационный файл dynconfig.yaml на кластер
ydb admin config replace -f dynconfig.yaml
```

Дополнительные возможности конфигурирования описаны на страницах селекторы и временная конфигурация.
Все команды для работы с конфигурацией описаны в разделе Получить конфигурацию кластера.

Механизм работы

Обновление конфигурации с точки зрения администратора

1. Конфигурационный файл загружается пользователем при помощи grpc-вызова или YDB CLI в кластер.
2. Файл проверяется на валидность, проверяются базовые ограничения, корректность версии, корректность имени кластера, корректность конфигураций полученных после преобразования DSL.
3. Версия конфигурации в файле увеличивается на единицу.
4. Файл надёжно сохраняется в кластере таблеткой Console.
5. Обновления файла рассылаются по узлам кластера.

Обновление конфигурации с точки зрения узла кластера

1. Каждый узел при старте запрашивает полную конфигурацию.
2. Получив конфигурацию, узел генерирует конечную конфигурацию для своего набора лейблов.
3. Узел подписывается на обновления конфигурации, регистрируясь в таблетке Console.
4. В случае обновления конфигурации локальный сервис получает его и преобразует для лейблов узла.
5. Все локальные сервисы, подписанные на обновления, получают обновлённую конфигурацию.

Пункты 1,2 выполняются только для динамических узлов кластера.

Версионирование конфигурации

Данный механизм позволяет избежать конкурентной модификации конфигурации и сделать её обновление идемпотентным. При получении запроса на модификацию конфигурации сервер сравнивает версию полученной модификации с сохраненной. Если версия меньше на единицу, то конфигурации сравниваются — если они одинаковы, значит пользователь пытается загрузить конфигурацию повторно, пользователь получает OK, а конфигурация на кластере не обновляется. Если версия равна текущей на кластере, то конфигурация заменяется на новую, при этом поле версии увеличивается на единицу. Во всех остальных случаях пользователь получает ошибку.

Динамически обновляемые настройки

Часть настроек системы обновляется без перезапуска узлов. Для их изменения достаточно загрузить новую конфигурацию и дождаться её распространения по кластеру.

Список динамически обновляемых настроек:

- `immediate_controls_config`;
- `log_config`;
- `memory_controller_config`;
- `monitoring_config`;
- `table_service_config`;
- `tracing_config.external_throttling`;
- `tracing_config.sampling`.

В будущем список может быть расширен.

Ограничения

- Использование более 30 различных лейблов в селекторах может привести к задержкам при валидации конфигурации в десятки секунд, т.к. YDB необходимо проверить валидность каждой возможной конечной конфигурации. При этом количество значений одного лейбла влияет намного меньше.
- Использование объемных файлов (более 500KiB для кластера в 1000 узлов) конфигурации может привести к росту сетевого трафика в кластере при обновлении конфигурации. Объем трафика прямо пропорционален количеству нод и объему конфигурации.

Временные конфигурации

Временные конфигурации — это специальный вид конфигураций, которые дополняют динамическую и при этом не являются персистентными. Т.е. данные конфигурации сбрасываются при переезде или рестарте таблетки Console, а так же при обновлении основной конфигурации.

Основные сценарии использования:

- временное изменение конфигурации для отладки или тестирования;
- пробное включение потенциально опасных настроек. В случае падения или рестарта кластера данные настройки будут автоматически отключены.

Данные конфигурации добавляются в конец набора селекторов, синтаксис описания идентичен синтаксису селекторов.

```
# Получить все временные конфигурации загруженные на кластер
ydb admin volatile-config fetch --all --output-directory <dir>
# Получить временную конфигурацию с id=1
ydb admin volatile-config fetch --id 1
# Применить временную конфигурацию volatile.yaml на кластер
ydb admin volatile-config add -f volatile.yaml
# Удалить временные конфигурации с id=1 и id=3 на кластере
ydb admin volatile-config drop --id 1 --id 3
# Удалить все временные конфигурации на кластере
ydb admin volatile-config drop --all
```

Пример работы с временной конфигурацией

Временное включение настроек журналирования компонента `blobstorage` в `DEBUG` на узле `host1.example.com`:

```
# Запрос текущих метаданных, чтобы сформировать корректный заголовок временной конфигурации
$ ydb admin config fetch --all
---
kind: MainConfig
cluster: "example-cluster-name"
version: 2
config:
  # ...
---
kind: VolatileConfig
cluster: "example-cluster-name"
version: 2
id: 1
selector_config:
  # ...
# Загрузка конфигурации с версией 2, именем кластера example-cluster-name и идентификатором 2
$ ydb admin volatile-config add -f - <<<EOF
metadata:
  kind: VolatileConfig
  cluster: "example-cluster-name"
  version: 2
  id: 2
selector_config:
- description: Set blobstorage logging level to DEBUG
  selector:
    node_host: host1.example.com
  config:
    log_config: !inherit
      entry: !inherit_key:component
      - component: BLOBSTORAGE
        level: 8
EOF
# ...
# анализ журнала
# ...
# Удаление конфигурации
$ ydb admin volatile-config drop --id 2
```

Domain-specific language (DSL) конфигурации кластера

YDB предоставляет DSL конфигурации кластера для более гибкого управления применением настроек к узлам кластера.

Селекторы

Основной сущностью DSL являются **селекторы**. Они позволяют переопределять часть конфигурации или конфигурацию целиком для определённых узлов или групп узлов. Например, они могут быть использованы для включения экспериментальной функциональности узлов определённой базы данных.
Каждый селектор представляет собой массив переопределений и расширений основной конфигурации. У каждого селектора есть поле `description`, которое может использоваться для хранения произвольной строки описания. Поле `selector` представляет собой набор правил, по которым определяется, должен ли данный селектор применяться для определённого узла, то есть для какого-либо набора лейблов. Поле `config` описывает правила переопределения.
Селекторы применяются в порядке их описания.

> **Примечание**
>
> Механизм селекторов поддерживается только для узлов баз данных.

Лейблы

Лейблы — это особые метки, с помощью которых можно пометить узлы или группы узлов. Каждый узел имеет набор автоматически задаваемых лейблов:

- `node_id` — внутренний идентификатор узла в системе;
- `node_host` — `hostname` узла, получаемый при старте;
- `tenant` — база данных, обслуживаемая данным узлом;
- `dynamic` — является ли данный узел динамическим (true/false).

И, опционально, какие-либо дополнительные лейблы, явно определённые пользователем при запуске процесса `ydbd` на узле при помощи аргументов командной строки. Например: `--label example=test`.

Пример использования селекторов

В примере ниже определяется общая конфигурация актор-системы и конфигурация для тенанта `large_tenant`. По умолчанию при такой конфигурации актор-система считает, что на узле 4 ядра, а на узлах тенанта `large_tenant` — по 16 ядер, при этом также переопределяется тип узлов для актор-системы на `COMPUTE`.

```
metadata:
  cluster: ""
  version: 8
config:
  actor_system_config:
    use_auto_config: true
    node_type: STORAGE
    cpu_count: 4

# Раздел используется как хинт при генерации возможных конфигураций при помощи команды resolve
allowed_labels:
  dynamic:
    type: string

selector_config:
- description: large_tenant has bigger nodes with 16 cpu # произвольная строка с описанием
  selector: # селектор для всех узлов тенанта large_tenant
    tenant: large_tenant
  config:
    actor_system_config: !inherit # переиспользуем исходный actor_system_config, семантика !inherit описана в
разделе ниже
      # в данном случае !inherit позволяет управлять параметром actor_system_config.use_auto_config одновреме
нно для всего кластера изменяя лишь базовую настройку
      cpu_count: 16
      node_type: COMPUTE
```

Допустимые лейблы

Отображение, в котором можно задать допустимые значения для лейблов. Раздел используется как хинт при генерации возможных конфигураций при помощи команды `resolve`. При запуске узлов значения не валидируются.

Доступны два типа лейблов:

- `string`;
- `enum`.

string

Может принимать любые значения или быть не задан.

Пример:

```
dynamic:
  type: string
host_name:
  type: string
```

enum

Может принимать значения из списка `values` или быть не задан.

Пример:

```
flavour:
  type: enum
  values:
    ? small
    ? medium
    ? big
```

Поведение селекторов

Селекторы представляют собой простой язык предикатов. Селекторы по каждому лейблу объединяются через условие **И**.

Простой селектор

Следующий селектор выберет узлы, на которых значение лейбла `label1` равно `value1` **и** лейбла `label2` — `value2`:

```
selector:
  label1: value1
  label2: value2
```

Следующий селектор выберет **ВСЕ** узлы в кластере, т.к. никаких условий не задано:

```
selector: {}
```

In

Данный оператор позволяет выбирать узлы с значением лейблов из списка.

Следующий селектор выберет все узлы, где лейбл `label1` равен `value1` **или** `value2`:

```
selector:
  label1:
    in:
    - value1
    - value2
```

NotIn

Данный оператор позволяет выбирать узлы, у которых выбранный лейбл не равен значению из списка.

Следующий селектор выберет все узлы, где лейбл `label1` равен `value1` **и** `label2` не равен `value2` **и** `value3`:

```
selector:
  label1: value1
  label2:
    not_in:
    - value2
    - value3
```

Дополнительные теги в YAML

Теги необходимы для частичного или полного переиспользования конфигураций предыдущих селекторов. С помощью них можно объеденить, расширить, удалить и полностью переопределить параметры, заданные в предыдущих селекторах и основной конфигурации.

!inherit

Область применения: YAML mapping
Действие: аналогично merge-тегу в YAML, скопировать все дочерние элементы из родительского отображения и объеденить с текущими с перезаписью
Пример:

Изначальная конфигурация	Переопределение	Результирующая конфигурация

`config:`
`  some_config:`
`  first_entry: 1`
`  second_entry: 2`
`  third_entry: 3`
	`config:`
	`  some_config: !inherit`
	`  second_entry: 100`
		`config:`
		`  some_config:`
		`  first_entry: 1`
		`  second_entry: 100`
		`  third_entry: 3`

!inherit:<key>

Область применения: YAML sequence
Действие: скопировать элементы родительского массива и переписать, трактуя объект key в элементах как ключ, дописывая новые ключи в конец
Пример:

Изначальная конфигурация	Переопределение	Результирующая конфигурация

```
config:
  some_config:
    array:
    - abc: 2
      value: 10
    - abc: 1
      value: 20
      another_value: test
```

```
config:
  some_config: !inherit
    array: !inherit:abc
    - abc: 1
      value: 30
    - abc: 3
      value: 40
```

```
config:
  some_config:
    array:
    - abc: 2
      value: 10
    - abc: 1
      value: 30
    - abc: 3
      value: 40
```

!remove

Область применения: YAML sequence element под `!inherit:<key>`
Действие: удалить элемент с соотвествующим ключом.
Пример:

Изначальная конфигурация	Переопределение	Результирующая конфигурация

`config:`
`  some_config:`
`  array:`
`  - abc: 2`
`  value: 10`
`  - abc: 1`
`  value: 20`
`  another_value: test`
	`config:`
	`  some_config: !inherit`
	`  array: !inherit:abc`
	`  - !remove`
	`  abc: 1`
		`config:`
		`  some_config:`
		`  array:`
		`  - abc: 2`
		`  value: 10`

!append

Область применения: YAML sequence
Действие: скопировать элементы родительского массива и дописать новые в конец
Пример:

Изначальная конфигурация	Переопределение	Результирующая конфигурация

`config:`
`  some_config:`
`  array:`
`  - abc: 2`
`  value: 10`
`  - abc: 1`
`  value: 20`
`  another_value: test`
	`config:`
	`  some_config: !inherit`
	`  array: !append`
	`  - abc: 1`
	`  value: 30`
	`  - abc: 3`
	`  value: 40`
		`config:`
		`  some_config:`
		`  array:`
		`  - abc: 2`
		`  value: 10`
		`  - abc: 1`
		`  value: 20`
		`  another_value: test`
		`  - abc: 1`
		`  value: 30`
		`  - abc: 3`
		`  value: 40`

Генерация конечных конфигураций

Конфигурации могут содержать сложные наборы переопределений. С помощью YDB CLI существует возможность посмотреть конечные конфигурации для:

- конкретных узлов;
- наборов лейблов;
- все возможные комбинации для текущей конфигурации.

```
# Сгенерировать все возможные конечные конфигурации для cluster.yaml
ydb admin config resolve --all -f cluster.yaml
# Сгенерировать конфигурацию для cluster.yaml при лейблах tenant=/Root/test и canary=true
ydb admin config resolve -f cluster.yaml --label tenant=/Root/test --label canary=true
# Сгенерировать конфигурацию для cluster.yaml при лейблах аналогичных текущим на узле 1001
ydb admin config resolve -f cluster.yaml --node_id 1001
# Взять текущую конфигурацию кластера и сгенерировать для него финальную конфигурацию при лейблах аналогичных текущим на узле 1001
ydb admin config resolve --from-cluster --node_id 1001
```

Более подробно команда преобразования конфигурации описана в разделе Получить конфигурацию кластера.

Пример вывода `ydb admin config resolve --all -f cluster.yaml` для следующего файла конфигурации:

```
metadata:
  cluster: ""
  version: 8
config:
  actor_system_config:
    use_auto_config: true
    node_type: STORAGE
    cpu_count: 4
allowed_labels:
  dynamic:
    type: string
selector_config:
```

```
- description: Actorsystem for dynnodes # произвольная строка с описанием
  selector: # селектор для всех узлов с лейблом dynamic = true
    dynamic: true
  config:
    actor_system_config: !inherit # переиспользуем исходный actor_system_config, семантика !inherit описана в
разделе ниже
      node_type: COMPUTE
      cpu_count: 8
```

Вывод:

```
---
label_sets: # наборы лейблов, для которых сгенерирована конфигурация
- dynamic:
    type: NOT_SET # один из трех типов лейбла: NOT_SET | COMMON | EMPTY
config: # сгенерированная конфигурация
  invalid: 1
  actor_system_config:
    use_auto_config: true
    node_type: STORAGE
    cpu_count: 4
---
label_sets:
- dynamic:
    type: COMMON
    value: true # значение лейбла
config:
  invalid: 1
  actor_system_config:
    use_auto_config: true
    node_type: COMPUTE
    cpu_count: 8
```

Изменение конфигураций через CMS

> **Примечание**
>
> Данный способ изменения конфигурации является устаревшим. Рекомендуемый способ конфигурирования описан в разделе динамическая конфигурация кластера.

Получить текущие настройки

Следующая команда позволит получить текущие настройки по кластеру или по тенанту.

```
ydbd -s <endpoint> admin console configs load --out-dir <config-folder>
```

```
ydbd -s <endpoint> admin console configs load --out-dir <config-folder> --tenant <tenant-name>
```

Обновить настройки

Сначала надо выкачать нужный конфиг как указано выше, после чего требуется подготовить protobuf файл с запросом на изменение.

```
Actions {
  AddConfigItem {
    ConfigItem {
      Cookie: "<cookie>"
      UsageScope {
        TenantAndNodeTypeFilter {
          Tenant: "<tenant-name>"
        }
      }
      Config {
          <config-name> {
              <full-config>
          }
      }
    }
  }
}
```

Поле UsageScope необязательно, и нужно для применения настроек для определенного тенанта.

```
ydbd -s <endpoint> admin console configs update <protobuf-file>
```

Изменение конфигурации актор-системы

Актор-система — основа YDB, каждый компонент системы представляется одним или несколькими акторами.
Каждый актор распределен в определенный ExecutorPool, соответствующий задаче актора.
Изменение конфигурации поможет более точно распределить количество зарезервированных ядер для каждого типа задач.

Описание конфига актор-системы

Конфиг актор-системы состоит из перечисления ExecutorPool'ов, сопоставления ExecutorPool с видами задач и конфигураций шедулера актор-системы.

На данный момент есть следующие типы задач и соответствующие им пулы:

- System - предназначен для выполнения быстрых внутренних операций YDB;
- User - включает в себя всю пользовательскую нагрузку по обработке и выполнению входящих запросов;
- Batch - задачи, которые не имеют строгого лимита на время выполнения, в основном выполнение фоновых операций;
- IO - отвечает за выполнение всех задач с блокирующими операциями (как пример запись логов в файл);
- IC - Interconnect, включает всю нагрузку связанную с коммуникацией между узлами.

Каждый пул описывается полем Executor, как в примере ниже.

```
Executor {
  Type: BASIC
  Threads: 9
  SpinThreshold: 1
  Name: "System"
}
```

Описание основных полей:

- **Type** - на данный момент может иметь два типа: **BASIC** и **IO**. Все пулы, кроме **IO**, имеют тип **BASIC**;
- **Threads** - количество потоков (количество параллельно работающих акторов) в данном пуле;
- **SpinThreshold** - количество тактов процессора перед уходом в сон при отсутствии задач, которое будет исполнять поток, выполняющий работу акторов (влияет на потребление cpu и latency запросов во время слабой нагрузки);
- **Name** - название пула, которое будет отображаться у узла в мониторинге.

Сопоставление пулов с типами задач происходит заданием порядкового номера пула в специальные поля. Пулы нумеруются с нуля, также на один пул может повесить несколько типов задач.

Список полей с их задачами:

- **SysExecutor** - System
- **UserExecutor** - User
- **BatchExecutor** - Batch
- **IoExecutor** - IO

Пример:

```
SysExecutor: 0
UserExecutor: 1
BatchExecutor: 2
IoExecutor: 3
```

IC пул задается другим способом, через ServiceExecutor, как в примере ниже.

```
ServiceExecutor {
  ServiceName: "Interconnect"
  ExecutorId: 4
}
```

Шедулер актор-системы отвечает за доставку отложенных сообщений между акторами и задается следующими параметрами:

- **Resolution** - минимальный шаг смещения времени в микросекундах;
- **SpinThreshold** - аналогично с параметром пула, количество тактов процессора перед уходом в сон при отсутствии сообщений;
- **ProgressThreshold** - максимальный шаг смещения времени в микросекундах.

В случае если по неизвестным причинам поток шедулера залип, то он будет отправлять сообщения по отстающему времени, смещая его каждый раз на **ProgressThreshold**.

Не рекомендуется менять конфиг шедулера. В конфигах пулов рекомендуется менять только количество потоков.

Пример конфига актор-системы по умолчанию:

```
Executor {
  Type: BASIC
  Threads: 9
  SpinThreshold: 1
  Name: "System"
}
Executor {
  Type: BASIC
  Threads: 16
  SpinThreshold: 1
  Name: "User"
}
Executor {
  Type: BASIC
```

```
  Threads: 7
  SpinThreshold: 1
  Name: "Batch"
}
Executor {
  Type: IO
  Threads: 1
  Name: "IO"
}
Executor {
  Type: BASIC
  Threads: 3
  SpinThreshold: 10
  Name: "IC"
  TimePerMailboxMicroSecs: 100
}
SysExecutor: 0
UserExecutor: 1
IoExecutor: 3
BatchExecutor: 2
ServiceExecutor {
  ServiceName: "Interconnect"
  ExecutorId: 4
}
```

На статических узлах

Статические узлы берут конфигурацию актор-системы из файла `/opt/ydb/cfg/config.yaml`.

После замены конфигурации требуется перезапустить узел.

На динамических узлах

Динамические узлы берут конфигурацию из CMS, чтобы изменить ее, можно воспользоваться следующей командой:

```
ConfigureRequest {
  Actions {
    AddConfigItem {
      ConfigItem {
        // UsageScope: { ... }
        Config {
          ActorSystemConfig {
            <конфиг актор-системы>
          }
        }
        MergeStrategy: 3
      }
    }
  }
}

```bash
ydbd -s <endpoint> admin console execute --domain=<domain> --retry=10 actorsystem.txt
```
```

Расширение кластера

Вы можете расширить кластер YDB, добавив новые узлы в конфигурацию кластера. Ниже приведены необходимые действия по расширению кластера YDB, установленного вручную на виртуальные машины или физические сервера. Расширение кластера в среде Kubernetes осуществляется путём корректировки настроек контроллера YDB для Kubernetes.

Расширение кластера YDB не требует приостановки доступа пользователей к базам данных. При расширении кластера выполняется перезапуск его компонент для применения изменений в конфигурации, что, в свою очередь, может привести к необходимости повтора выполняемых на кластере транзакций. Повторы транзакций выполняются автоматически за счет использования приложениями возможностей SDK YDB по контролю ошибок и повтору операций.

Подготовка новых серверов

В случае размещения новых статических или динамических узлов кластера на новых серверах, не входивших ранее в состав расширяемого кластера YDB, на каждом новом сервере необходимо выполнить установку программного обеспечения YDB в соответствии с процедурами, описанными в инструкции по развертыванию кластеров. В частности, необходимо:

1. создать учетную запись и группу в операционной системе для работы сервиса YDB;
2. установить программное обеспечение YDB;
3. подготовить и разместить на сервере соответствующий ему ключ и сертификат TLS;
4. скопировать на сервер актуальный конфигурационный файл кластера YDB.

Используемые на новых серверах сертификаты TLS должны соответствовать требованиям к заполнению полей, и быть подписаны доверенным центром регистрации, используемым на уже существующих серверах расширяемого кластера YDB.

Добавление динамических узлов

Добавление динамических узлов позволяет увеличить доступные вычислительные ресурсы (процессорные ядра и оперативную память) для выполнения пользовательских запросов кластером YDB.

Для добавления динамического узла в кластер достаточно запустить процесс, обслуживающий этот узел, передав ему в параметрах командной строки путь к конфигурационному файлу кластера, имя обслуживаемой базы данных и адреса любых трех статических узлов кластера кластера YDB, как показано в инструкции по развертыванию кластеров.

После успешного добавления динамического узла в кластер информация о нем будет доступна на странице мониторинга кластера в YDB UI.

Для вывода динамического узла из кластера достаточно выполнить остановку процесса динамического узла.

Добавление статических узлов

Добавление статических узлов позволяет увеличить пропускную способность при выполнении операций ввода-вывода и увеличить доступную ёмкость для хранения данных в кластере YDB.

Для добавления статических узлов в кластер необходимо выполнить следующую последовательность действий:

1. Отформатировать диски, которые будут использоваться для хранения данных YDB, с использованием процедуры, описанной для этапа развертывания кластера.
2. Скорректировать конфигурационный файл кластера, лежащий на нодах:
   - включить в конфигурацию описание добавляемых узлов (в секции `hosts`) и используемых на них дисков (в секции `host_configs`);
   - установить номер изменения конфигурации в виде параметра `storage_config_generation: K` на верхнем уровне, где `K` - целое число, номер изменения (при первоначальной установке значение `K=0` или не указано, при первом расширении кластера `K=1`, при втором `K=2`, и так далее).
3. Скопировать доработанный конфигурационный файл кластера на все существующие и на все добавляемые сервера кластера, заместив им старую версию конфигурационного файла.
4. Выполнить последовательный перезапуск всех существующих статических узлов кластера, с ожиданием инициализации и восстановления работы каждого перезапускаемого узла.
5. Выполнить последовательный перезапуск всех существующих динамических узлов кластера.
6. Запустить процессы, обслуживающие новые статические узлы кластера, на соответствующих серверах.
7. Убедиться в том, что новые статические узлы отображаются на странице мониторинга кластера в YDB UI.
8. Получить токен аутентификации для выполнения административных команд с помощью YDB CLI, например:

   ```
   ydb -e grpcs://<node1.ydb.tech>:2135 -d /Root --ca-file ca.crt \
       --user root auth get-token --force >token-file
   ```

   В примере команды выше используются следующие параметры:

   - `node1.ydb.tech` - FQDN любого из серверов, на которых размещены статические узлы кластера;
   - `2135` - номер порта grpcs сервиса статических узлов;
   - `ca.crt` - имя файла с сертификатом центра регистрации;
   - `root` - логин пользователя с административными правами;
   - `token-file` - имя файла, в который сохраняется токен аутентификации для последующего использования.

   При выполнении приведенной выше команды YDB CLI запросит пароль для аутентификации указанного пользователя.
9. Разрешить кластеру YDB использовать диски на новых статических узлах для хранения данных, выполнив следующую команду на любом из узлов кластера:

   ```
   export LD_LIBRARY_PATH=/opt/ydb/lib
   /opt/ydb/bin/ydbd -f ydbd-token-file --ca-file ca.crt -s grpcs://`hostname -f`:2135 \
       admin blobstorage config init --yaml-file  /opt/ydb/cfg/config.yaml
   echo $?
   ```

   В примере команды выше используются следующие параметры:

   - `ydbd-token-file` - имя файла ранее полученного токена аутентификации;
   - `2135` - номер порта grpcs сервиса статических узлов;

- `ca.crt` - имя файла с сертификатом центра регистрации.

Если при выполнении приведённой выше команды возвращается ошибка сверки номера конфигурации, это означает, что при корректировке конфигурационного файла кластера было неверно установлено поле `storage_config_generation`. Текст ошибки содержит ожидаемое значение номера конфигурации, которое можно использовать для корректировки файла настроек кластера. Пример сообщения об ошибке сверки номера конфигурации:

```
ErrorDescription: "ItemConfigGeneration mismatch ItemConfigGenerationProvided# 0 ItemConfigGenerationExp
ected# 1"
```

10. Добавить дополнительные группы хранения в одну или несколько баз данных, выполнив команды следующего вида на любом из узлов кластера:

```
export LD_LIBRARY_PATH=/opt/ydb/lib
/opt/ydb/bin/ydbd -f ydbd-token-file --ca-file ca.crt -s grpcs://`hostname -f`:2135 \
    admin database /Root/testdb pools add ssd:1
echo $?
```

В примере команды выше используются следующие параметры:

- `ydbd-token-file` - имя файла ранее полученного токена аутентификации;
- `2135` - номер порта grpcs сервиса статических узлов;
- `ca.crt` - имя файла с сертификатом центра регистрации;
- `/Root/testdb` - полный путь к базе данных;
- `ssd:1` - имя пула хранения и количество выделяемых групп хранения.

11. Убедиться, что добавленные группы хранения отображаются на странице мониторинга кластера в YDB UI.

Вывод статических узлов из кластера YDB производится в соответствии с документированной процедурой декомиссии.

В случае повреждения и невозможности ремонта сервера, на котором работает статический узел кластера, необходимо разместить недоступный статический узел на новом сервере, содержащем аналогичное или большее количество и объем дисков.

Конфигурирование подсистем распространения метаданных State Storage, Board, Scheme Board

Применяется если нужно изменить конфигурацию подсистем распространения метаданных состоящую из State Storage, Board, Scheme Board, на кластере YDB.

> **Важно**
>
> Ошибка в конфигурации подсистем распространения метаданных (в том числе в секции `domains_config`) или неправильная последовательность изменений могут привести к недоступности кластера YDB.

Для изменения конфигурации подсистем распространения метаданных в кластере YDB необходимо выполнить следующие шаги.

1. Внести требуемые изменения в секции `domains_config` конфигурационного файла `config.yaml` на каждом узле кластера YDB:
   Правила изменения секции `domains_config` см. в разделе Правила конфигурирования подсистем распространения метаданных.
2. С помощью процедуры rolling-restart последовательно перезапустите все узлы кластера YDB — статические и динамические: реплики подсистем метаданных размещаются на статических узлах, к ним обращаются таблетки на динамических. Подробнее об узлах — в разделе Топология кластера.
   Перед рестартом следующего хоста необходима задержка, чтобы перезапуск на предыдущем хосте завершился и узел снова участвовал в работе кластера.

Правила конфигурирования подсистем распространения метаданных

Перечисленные ниже правила применяются к `state_storage` и к раздельным полям `explicit_state_storage_config`, `explicit_state_storage_board_config`, `explicit_scheme_board_config` в секции `domains_config` файла `config.yaml` (см. Конфигурация State Storage). Ключи `explicit_*` соответствуют по отдельности State Storage, Board и Scheme Board.

Под кольцом в конфигурации понимается блок `ring` внутри элемента списка `ring_groups` (см. Конфигурация State Storage).

Изменение конфигурации производится в несколько шагов. Сначала добавляется новая группа колец, состоящая из правильно выбранных узлов (в соответствии с моделью отказа), а затем старая группа колец удаляется.

Чтобы избежать недоступности кластера, удаление и добавление групп колец выполняйте строго в последовательности шагов, описанной ниже.

1. Чтобы изменить конфигурацию подсистем распространения метаданных без недоступности кластера необходимо производить это путем добавления и удаления групп колец.
2. Добавлять и удалять можно только группы колец с параметром `WriteOnly: true`.
3. В новой конфигурации всегда должна присутствовать хотя бы одна группа колец из предыдущей конфигурации без параметра `WriteOnly`. Такая группа колец должна идти первой в списке.
4. Если в разных группах колец используются одни и те же узлы кластера, добавьте параметр `ring_group_actor_id_offset` к группе колец с уникальным значением (например, `1`, `2`, ...). Значение должно быть уникальным среди групп колец.

   Этот параметр сделает уникальными идентификаторы реплик в данной группе колец, они не будут совпадать с идентификаторами из других групп, и это позволит разместить на одном узле кластера несколько реплик одного типа.
5. Переход к новой конфигурации выполняется за 4 последовательных шага. На каждом шаге подготавливается новая конфигурация и применяется к кластеру.

   Только что созданные или готовые к удалению группы колец помечаются флагом `WriteOnly: true`. Это необходимо, чтобы запросы на чтение обрабатывались уже развернутой группой колец, пока новая конфигурация распространится на необходимое количество узлов, создадутся новые реплики или удалятся старые.

   Поэтому между шагами необходимо делать паузу как минимум в `1 минуту`.

```
- Добавляем новую группу колец с параметром `WriteOnly: true` соответствующую целевой конфигурации.
- Снимаем флаг `WriteOnly`.
- Выставляем флаг `WriteOnly: true` на группу колец соответствующую старой конфигурации, новую группу колец п
ереносим в начало списка групп колец.
- Удаляем старую группу колец.
```

Пример

Рассмотрим на примере текущей конфигурации:

```
config:
  domains_config:
    explicit_scheme_board_config:
      ring:
        nto_select: 5
        node: [1,2,3,4,5,6,7,8]
```

и целевой конфигурации:

```
config:
  domains_config:
    explicit_scheme_board_config:
      ring:
        nto_select: 5
        node: [10,20,30,40,5,6,7,8]
```

Мы хотим перенести часть реплик на другие узлы кластера.

Шаг 1
На первом шаге подготовьте `ring_groups`, следуя правилам конфигурирования: первая группа колец совпадает с **текущей** конфигурацией из листингов выше, вторая — с **целевой** и помечена `WriteOnly: true`. Параметр `ring_group_actor_id_offset` указывайте так, как описано в тех же правилах, если наборы узлов у групп совпадают.

```
config:
  domains_config:
    explicit_scheme_board_config:
      ring_groups:
        - ring:
          nto_select: 5
          node: [1,2,3,4,5,6,7,8]
        - ring:
          nto_select: 5
          node: [10,20,30,40,5,6,7,8]
          write_only: true
          ring_group_actor_id_offset: 1
```

Шаг 2
Снимаем флаг `WriteOnly`.

```
config:
  domains_config:
    explicit_scheme_board_config:
      ring_groups:
        - ring:
          nto_select: 5
          node: [1,2,3,4,5,6,7,8]
        - ring:
          nto_select: 5
          node: [10,20,30,40,5,6,7,8]
          ring_group_actor_id_offset: 1
```

Шаг 3
Делаем новую группу колец первой в списке. На старую конфигурацию выставляем флаг `WriteOnly: true`

```
config:
  domains_config:
    explicit_scheme_board_config:
      ring_groups:
        - ring:
          nto_select: 5
          node: [10,20,30,40,5,6,7,8]
          ring_group_actor_id_offset: 1
        - ring:
          nto_select: 5
          node: [1,2,3,4,5,6,7,8]
          write_only: true
```

Шаг 4
Применяем на кластере целевую конфигурацию:

```
config:
  domains_config:
    explicit_scheme_board_config:
      ring_groups:
        - ring:
          nto_select: 5
          node: [10,20,30,40,5,6,7,8]
          ring_group_actor_id_offset: 1
```

Проверка результата

Проверить, что изменения применились, можно в разделе `CMS` в YDB UI кластера (доступен на порту 8765): перейдите на вкладку `Tablets` и убедитесь по репликам таблеток подсистем метаданных, что конфигурация подхватилась.

Перемещение State Storage

Если нужно вывести из эксплуатации хост кластера YDB, на котором располагается часть State Storage, необходимо переместить её на другой хост.

> **Важно**
>
> Ошибка в конфигурации подсистем распространения метаданных (в том числе в секции `domains_config`) или неправильная последовательность изменений могут привести к недоступности кластера YDB.

В качестве примера рассмотрим кластер YDB со следующей конфигурацией State Storage:

```
...
domains_config:
  ...
  state_storage:
  - ring:
      node: [1, 2, 3, 4, 5, 6, 7, 8, 9]
      nto_select: 9
    ssid: 1
  ...
...
```

На хосте с `node_id:1` сконфигурирован и запущен статический узел кластера, который обслуживает часть State Storage. Предположим, нам нужно вывести из эксплуатации этот хост.

Для замены `node_id:1` мы добавили в кластер новый хост с `node_id:10` и развернули на нём статический узел.

Чтобы переместить State Storage с хоста `node_id:1` на `node_id:10`:

1. Остановите статические узлы кластера на хостах с `node_id:1` и `node_id:10`.

   > **Примечание**
   >
   > Кластер YDB является отказоустойчивым. Временное выключение узла не приводит к недоступности кластера. Подробнее см. Топология кластера YDB.

2. В конфигурационном файле `config.yaml` измените список хостов `node`, заменив идентификатор удаляемого хоста на идентификатор добавляемого:

   ```
   domains_config:
   ...
     state_storage:
     - ring:
         node: [2, 3, 4, 5, 6, 7, 8, 9, 10]
         nto_select: 9
       ssid: 1
   ...
   ```

3. Обновите конфигурационные файлы `config.yaml` для всех узлов кластера, в том числе и динамических.
4. С помощью процедуры rolling-restart перезапустите все узлы кластера, включая динамические, кроме статических узлов на хостах с `node_id:1` и `node_id:10`. Обратите внимание, что между рестартом хостов необходима задержка как минимум в 15 секунд.
5. Запустите статические узлы кластера на хостах `node_id:1` и `node_id:10`.

Перемещение статической группы

Если нужно вывести из эксплуатации хост кластера YDB, на котором располагается часть статической группы, необходимо переместить её на другой хост.

> **Важно**
>
> Ошибка в конфигурации подсистем распространения метаданных (в том числе в секции `domains_config`) или неправильная последовательность изменений могут привести к недоступности кластера YDB.

В качестве примера рассмотрим кластер YDB, в котором на хосте с `node_id:1` сконфигурирован и запущен статический узел. Этот узел обслуживает часть статической группы.

Фрагмент конфигурации статической группы:

```
...
blob_storage_config:
  ...
  service_set:
    ...
    groups:
      ...
      rings:
        ...
        fail_domains:
        - vdisk_locations:
          - node_id: 1
            path: /dev/vda
            pdisk_category: SSD
        ...
      ...
    ...
  ...
...
```

Для замены `node_id:1` мы добавили в кластер новый хост с `node_id:10` и развернули на нём статический узел.

Чтобы переместить часть статической группы с хоста `node_id:1` на `node_id:10`:

1. Остановите статический узел кластера на хосте с `node_id:1`.

   > **Примечание**
   >
   > Кластер YDB является отказоустойчивым. Временное выключение узла не приводит к недоступности кластера. Подробнее см. Топология кластера YDB.

2. В конфигурационном файле `config.yaml` измените значение `node_id`, заменив идентификатор удаляемого хоста на идентификатор добавляемого:

   ```
   ...
   blob_storage_config:
     ...
     service_set:
       ...
       groups:
         ...
         rings:
           ...
           fail_domains:
           - vdisk_locations:
             - node_id: 10
               path: /dev/vda
               pdisk_category: SSD
           ...
         ...
       ...
     ...
   ...
   ```

   Измените путь `path` и категорию `pdisk_category` диска, если на хосте с `node_id: 10` они отличаются.
3. Обновите конфигурационные файлы `config.yaml` для всех узлов кластера, в том числе и динамических.
4. С помощью процедуры rolling-restart перезапустите все статические узлы кластера.
5. Перейдите на страницу мониторинга YDB UI и убедитесь, что VDisk статической группы появился на целевом физическом диске и реплицируется. Подробнее см. Мониторинг статической группы.
6. С помощью процедуры rolling-restart перезапустите все динамические узлы кластера.

Аутентификация и авторизация узлов баз данных

Иногда возникает ситуация, когда меняется FQDN узла, при этом сам узел в системе остаётся, но под другим именем. Простая замена имени узла в разделе `hosts` не сработает, так как `BS_CONTROLLER` внутри хранит привязку ресурсов к парам `FQDN:IcPort`, где `IcPort` — номер порта Interconnect, на котором работает узел.

Процедура замены

1. Для заменяемого узла определить его `NodeId`.
2. Подготовить команду `DefineBox`, которая описывает ресурсы кластера, в которой для ресурсов заменяемого узла будет добавлен элемент `EnforcedNodeId: <NodeId>`.
3. Выполнить эту команду.
4. Заменить FQDN в списке `hosts` в `cluster.yaml`.
5. Выполнить rolling restart.
6. Убрать из `DefineBox` поле `EnforcedNodeId` и заменить `Fqdn` на новое название узла.
7. Выполнить `DefineBox` с новыми значениями.

Пример

Предположим кластер, состоящий из трёх узлов:

config.yaml:

```
- host: host1.my.sub.net
  node_id: 1
  location: {unit: 12345, data_center: MYDC, rack: r1}
- host: host2.my.sub.net
  node_id: 2
  location: {unit: 23456, data_center: MYDC, rack: r2}
- host: host3.my.sub.net
  node_id: 3
  location: {unit: 34567, data_center: MYDC, rack: r3}
```

DefineBox выглядит так:

```
DefineBox {
    BoxId: 1
    Host { Key { Fqdn: "host1.my.sub.net" IcPort: 19001 } HostConfigId: 1 }
    Host { Key { Fqdn: "host2.my.sub.net" IcPort: 19001 } HostConfigId: 1 }
    Host { Key { Fqdn: "host3.my.sub.net" IcPort: 19001 } HostConfigId: 1 }
}
```

Предположим, что мы хотим переименовать host1.my.sub.net в host4.my.sub.net. Для этого сначала делаем DefineBox следующего вида:

```
DefineBox {
    BoxId: 1
    Host { Key { Fqdn: "host1.my.sub.net" IcPort: 19001 } HostConfigId: 1 EnforcedNodeId: 1 }
    Host { Key { Fqdn: "host2.my.sub.net" IcPort: 19001 } HostConfigId: 1 }
    Host { Key { Fqdn: "host3.my.sub.net" IcPort: 19001 } HostConfigId: 1 }
}
```

Затем изменяем config.yaml:

```
- host: host4.my.sub.net
  node_id: 1
  location: {unit: 12345, data_center: MYDC, rack: r1}
- host: host2.my.sub.net
  node_id: 2
  location: {unit: 23456, data_center: MYDC, rack: r2}
- host: host3.my.sub.net
  node_id: 3
  location: {unit: 34567, data_center: MYDC, rack: r3}
```

Затем делаем rolling restart кластера.

И затем делаем второй раз скорректированный DefineBox:

```
DefineBox {
    BoxId: 1
    Host { Key { Fqdn: "host4.my.sub.net" IcPort: 19001 } HostConfigId: 1 }
    Host { Key { Fqdn: "host2.my.sub.net" IcPort: 19001 } HostConfigId: 1 }
    Host { Key { Fqdn: "host3.my.sub.net" IcPort: 19001 } HostConfigId: 1 }
}
```

Аутентификация и авторизация узлов баз данных

Аутентификация узлов в кластере YDB обеспечивает проверку подлинности узлов баз данных при выполнении служебных вызовов к другим узлам по протоколу gRPC. Авторизация узлов обеспечивает проверку и предоставление необходимых полномочий при обработке служебных вызовов, включая операции регистрации запускаемых узлов в кластере и доступа к конфигурации. Использование аутентификации и авторизации узлов рекомендуется для всех кластеров YDB, поскольку позволяет избежать ситуаций несанкционированного доступа к данным через включение в кластер контролируемых злоумышленниками узлов.

Аутентификация и авторизация узлов баз данных осуществляется в следующем порядке:

1. Запускаемый узел базы данных открывает gRPC-подключение к одному из узлов хранения кластера, указанных в опции командной строки `--node-broker`. При подключении используется протокол TLS, и в настройках подключения в качестве клиентского сертификата используется сертификат запускаемого узла.
2. Узел хранения и узел базы данных осуществляют взаимные проверки подлинности с использованием протокола TLS: производится проверка цепочки доверия сертификата и проверка соответствия имени хоста значению поля «Subject Name» сертификата.
3. Узел хранения проверяет заполнение поля «Subject» сертификата на соответствие требованиям, устанавливаемым настройками в статической конфигурации.
4. При успешном прохождении перечисленных выше проверок подключение со стороны узла базы данных считается аутентифицированным, и ему присваивается идентификатор субъекта доступа — SID, определяемый настройками.
5. Узел базы данных использует установленное gRPC-подключение для регистрации в составе кластера с использованием соответствующего служебного вызова. При регистрации узел базы данных передаёт свой сетевой адрес, предназначенный для взаимодействия с другими узлами кластера.
6. Узел хранения проверяет наличие SID, присвоенного gRPC-подключению, в списке допустимых. При успешном прохождении этой проверки узел хранения регистрирует узел базы данных в кластере, сопоставляя полученный сетевой адрес с идентификатором узла.
7. Узел базы данных входит в кластер, подключаясь через свой сетевой адрес и указывая идентификатор узла, полученный при регистрации. Попытки входа в кластер узлов с неизвестными сетевыми адресами или идентификаторами блокируются другими узлами.

Далее описаны настройки, необходимые для включения функции аутентификации и авторизации узлов.

Пререквизиты для настройки

1. В развёрнутом кластере YDB должно быть настроено шифрование трафика gRPC с использованием протокола TLS.
2. При подготовке сертификатов узлов для кластера, в котором планируется использовать функции аутентификации и авторизации узлов, необходимо обеспечить единые правила заполнения поля «Subject» сертификатов, позволяющие идентифицировать сертификаты, выпущенные для узлов кластера. Более подробная информация приведена в документации по настройке правил проверки сертификатов.

   Примечание

   Предлагаемый пример скрипта для генерации самоподписанных сертификатов узлов YDB заполняет поле «Subject» значением `O=YDB` для всех сертификатов узлов. Приведённые далее примеры настроек подготовлены для сертификатов с именно таким заполнением поля «Subject».

3. В параметры командной строки для запуска узлов баз данных необходимо добавить опции, задающие пути к файлам сертификатов доверенных центров сертификации, сертификата узла и ключа узла. Список дополнительных опций приведён в таблице ниже.

Опция командной строки	Описание
`--grpc-ca`	Путь к файлу сертификатов доверенных центров сертификации `ca.crt`.
`--grpc-cert`	Путь к файлу сертификата узла `node.crt`.
`--grpc-key`	Путь к файлу секретного ключа узла `node.key`.

   Пример команды для запуска узла базы данных с опциями, указывающими пути к ключам и сертификатам TLS для протокола gRPC:

   ```
   /opt/ydb/bin/ydbd server --config-dir /opt/ydb/cfg --tenant /Root/testdb \
       --grpcs-port 2136 --grpc-ca /opt/ydb/certs/ca.crt \
       --grpc-cert /opt/ydb/certs/node.crt --grpc-key /opt/ydb/certs/node.key \
       --ic-port 19002 --ca /opt/ydb/certs/ca.crt \
       --mon-port 8766 --mon-cert /opt/ydb/certs/web.pem \
       --node-broker grpcs://<ydb1>:2135 \
       --node-broker grpcs://<ydb2>:2135 \
       --node-broker grpcs://<ydb3>:2135
   ```

Включение режима аутентификации и авторизации узлов

Примечание

Начиная с версии 25.2, корректно настроенная аутентификация узлов баз данных является обязательной.

Регистрация узлов в кластере невозможна без выполнения аутентификации.

Для включения обязательной авторизации узлов баз данных в файл конфигурации кластера необходимо добавить следующие блоки настроек:

1. На корневом уровне конфигурации добавьте блок `client_certificate_authorization`, в котором укажите требования к заполнению поля "Subject" доверенных сертификатов подключаемых узлов, например:

```
client_certificate_authorization:
  request_client_certificate: true
  client_certificate_definitions:
    - member_groups: ["registerNode@cert"]
      subject_terms:
      - short_name: "O"
        values: ["YDB"]
```

При необходимости добавьте другие проверки сертификатов в соответствии с документацией.

В случае успешной проверки сертификата и соответствия компонентов поля "Subject" сертификата требованиям, установленным в блоке `subject_terms`, подключению будут присвоены субъекты доступа, перечисленные в параметре `member_groups`. Чтобы отделить такие субъекты доступа от других групп и учётных записей, к их наименованию добавляется суффикс `@cert`.

2. В блок настроек аутентификации кластера `security_config` добавьте элемент `register_dynamic_node_allowed_sids`, указав список субъектов доступа, которым разрешена регистрация узлов баз данных. Пример:

```
security_config:
  enforce_user_token_requirement: true
  ...
  register_dynamic_node_allowed_sids:
  - "registerNode@cert"
```

Более подробная информация по настройке параметров аутентификации кластера приведена в соответствующем разделе документации.

3. В блоке настроек `auth_config` присвойте параметру `node_registration_token` пустую строку. Пример:

```
auth_config:
  ...
  node_registration_token: ""
```

4. Обновите файлы статической конфигурации на всех узлах кластера вручную либо с помощью плейбука Ansible.
5. Выполните поэтапный перезапуск узлов хранения кластера с помощью ydbops либо с использованием плейбука Ansible.
6. Выполните поэтапный перезапуск узлов баз данных кластера с помощью ydbops либо с использованием плейбука Ansible.

Конфигурация V2

В этом разделе документации описан способ конфигурации кластеров YDB под названием V2, являющаяся экспериментальным способом конфигурирования кластеров YDB версий YDB v25.1 и выше.

Важно

Эта статья посвящена кластерам YDB, в которых используется конфигурация V2. Данный способ конфигурирования пока является экспериментальным и доступен только для версий YDB начиная с v25.1. Для использования в продакшене мы рекомендуем выбирать конфигурацию V1 — она является основной и официально поддерживаемой для всех кластеров YDB.

Основные материалы:

- Обзор конфигурации V2
- Обновление конфигурации кластеров YDB
- DSL конфигурации кластера
- Параметры конфигурации
- Расширение кластера
- Удаление узла из кластера
- Конфигурирование подсистем распространения метаданных State Storage, Board, Scheme Board
- Перемещение State Storage
- SelfHeal статической группы
- Перемещение статической группы
- Замена FQDN узла
- Аутентификация и авторизация узлов баз данных

Обзор конфигурации V2

Для развёртывания кластера YDB, добавления в кластер новых узлов и изменения параметров требуется конфигурация.

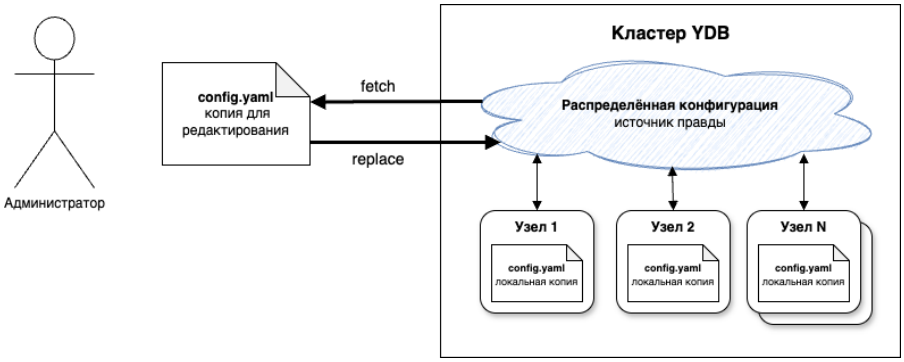
> **Важно**
>
> Эта статья посвящена кластерам YDB, в которых используется конфигурация V2. Данный способ конфигурирования пока является экспериментальным и доступен только для версий YDB начиная с v25.1. Для использования в продакшене мы рекомендуем выбирать конфигурацию V1 — она является основной и официально поддерживаемой для всех кластеров YDB.

Конфигурация кластера YDB V2 представляет собой текстовый файл в формате YAML. В минимальном варианте он содержит секцию `config` с различными параметрами, необходимыми для запуска и настройки узлов кластера, а также секцию с метаданными `metadata`. Расширенные возможности для гибкого конфигурирования описаны в статье Domain-specific language (DSL) конфигурации кластера. Подробнее о доступных параметрах можно узнать в справке по конфигурации.

Пример файла конфигурации V2

```yaml
metadata:
  cluster: ""
  version: 0
config:
  hosts:
    - host: localhost
  drive:
    - type: RAM
  grpc_config:
    port: 2136
  monitoring_config:
    monitoring_port: 8765
```

Управление конфигурацией



За управление состоянием конфигурационного файла отвечает сам кластер YDB, и он же является единственным источником правды о том, как он сейчас сконфигурирован. За надёжное сохранение текущего состояния, являющегося источником правды, отвечает механизм распределённой конфигурации; как это работает технически, подробнее описано в статье Устройство механизма конфигурации V2. Узнать текущее состояние конфигурации кластера можно с помощью консольной команды ydb admin cluster config fetch, а состояние каждого конкретного узла — через его YDB UI.

Изменение конфигурации кластера YDB осуществляется администратором следующим образом:

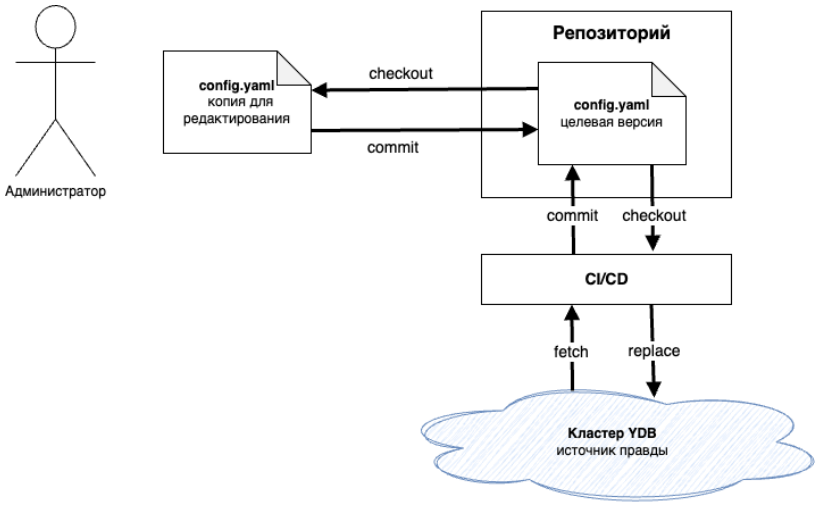
1. Сохранение текущего состояния конфигурации кластера в локальный файл через ydb admin cluster config fetch.
2. Редактирование нужных параметров в файле в текстовом редакторе или любым другим удобным способом.
3. Загрузка изменений обратно на кластер посредством вызова команды ydb admin cluster config replace.

Пример изменения конфигурации

```
$ ydb -e grpc://<ydb.example.com>:2135 admin cluster config fetch > config.yaml     # 1
$ vim config.yaml                                                                   # 2
$ ydb -e grpc://<ydb.example.com>:2135 admin cluster config replace -f config.yaml  # 3
```

Загрузка изменений обратно на кластер не всегда проходит успешно. Помимо базовой валидации корректности конфигурационного файла, у системы есть защита от конкурентного изменения несколькими администраторами. Система инкрементирует поле `metadata.version` при каждом изменении конфигурации и отказывается принимать новую версию, если её номер не совпадает с ожидаемым, так как это означает, что между `fetch` и `replace` было другое изменение, и `replace` его бы стёр. Чтобы минимизировать такие конфликты, можно использовать подход «Инфраструктура как код»: хранить копию конфигурационного файла в репозитории системы управления версиями (например, Git) и запускать команды `fetch` и `replace` не вручную, а только из привязанной к этому репозиторию системы непрерывной интеграции и доставки (CI/CD), реагирующей на изменения конфигурационного файла YDB в репозитории и обеспечивающей последовательную отправку всех изменений на кластер YDB.

Схема с подходом «Инфраструктура как код»



Каждый узел кластера YDB сохраняет локально копию конфигурации в директорию, указанную в аргументе запуска `ydbd --config-dir`. Эта локальная копия используется в следующих ситуациях:

1. Для применения настроек, которые нужны на самом старте работы узла, ещё до того, как у него появляется возможность начать общаться с другими узлами кластера. Изменение таких настроек может требовать перезапуска узла.
2. Для первоначального развёртывания и расширения кластера.
3. В случае форс-мажора, если с основным механизмом управления конфигурацией возникли проблемы, требующие ручного вмешательства.

Выше описан основной механизм управления конфигурацией V2 YDB. В зависимости от предпочитаемого способа управления инфраструктурой может предоставляться дополнительная автоматизация.

Базовые сценарии использования конфигурации

Первоначальное развёртывание кластера YDB

Для конфигурации кластера при первоначальном развёртывании рекомендуется использовать инструкции для выбранного способа управления инфраструктурой:

- Развёртывание YDB кластера с помощью Ansible;
- Начало работы с YDB в Kubernetes;
- Развёртывание YDB кластера вручную.

Обновление конфигурации

Для обновления конфигурации на уже развёрнутом кластере необходимо воспользоваться соответствующими командами в зависимости от способа развёртывания:

- Обновление конфигурации кластеров YDB, развёрнутых с Ansible;
- Обновление конфигурации кластеров YDB, развёрнутых вручную.

Если изменения конфигурации затрагивают параметры, требующие перезапуска узлов кластера, воспользуйтесь процедурой rolling restart. Подробнее о ней в зависимости от способа развёртывания:

- Перезапуск кластера, развёрнутого с помощью Ansible;
- Перезапуск кластера, развёрнутого вручную.

Расширение кластера

При расширении кластера конфигурация доставляется на старые и новые узлы по-разному:

- На узлы, существовавшие до расширения, изменения доставляются автоматически при вызове ydb admin cluster config replace.
- Перед первым запуском новых узлов локальная копия доставляется специальной командой ydb admin node config init, а не самим узлом.

Смотрите также

- Справка по параметрам конфигурации
- Сравнение конфигураций кластера YDB: V1 и V2

Обновление конфигурации кластеров YDB

В этой статье рассматривается изменение конфигурации кластера после первоначального развертывания.

> **Важно**
>
> Эта статья посвящена кластерам YDB, в которых используется конфигурация V2. Данный способ конфигурирования пока является экспериментальным и доступен только для версий YDB начиная с v25.1. Для использования в продакшене мы рекомендуем выбирать конфигурацию V1 — она является основной и официально поддерживаемой для всех кластеров YDB.

Управление конфигурацией кластера YDB осуществляется с помощью YDB CLI. Стандартный подход к обновлению включает получение текущей конфигурации от кластера с помощью YDB CLI, ее изменение локально и загрузку обновленной конфигурации обратно на кластер.

Чтобы предотвратить случайную перезапись изменений при одновременной работе нескольких администраторов, конфигурация YDB содержит метаданные с номером версии. При каждой успешной загрузке конфигурации этот номер автоматически увеличивается. Если другой администратор попытается загрузить конфигурацию, основанную на предыдущей (устаревшей) версии (т.е. с тем же или меньшим номером версии в метаданных), система отвергнет эту попытку. Это гарантирует, что изменения одного администратора не будут незаметно перезаписаны изменениями другого.

Для обновления конфигурации необходимо получить текущую конфигурацию, выполнить ее модификацию и применить обновленную конфигурацию. В процессе применения конфигурации она будет надежно сохранена кластером и распространена по всем его узлам. По мере доставки конфигурации на узлы кластера, конфигурация начинает применяться этими узлами. Некоторые параметры конфигурации изменяют поведение узла сразу после доставки на него конфигурации, другие же оказывают влияние только при запуске узла, и при изменении таких параметров для вступления их в силу необходим перезапуск кластера.

Базовые операции с конфигурацией

Получение текущей конфигурации

Для получения текущей конфигурации кластера используется команда:

```
ydb -e grpcs://<endpoint>:2135 admin cluster config fetch > config.yaml
```

В качестве `<endpoint>` указывается адрес любого из узлов кластера.

Применение новой конфигурации

Для загрузки обновленной конфигурации на кластер используется следующая команда:

```
ydb -e grpcs://<endpoint>:2135 admin cluster config replace -f config.yaml
```

В качестве `<endpoint>` указывается адрес любого из узлов кластера.

Некоторые параметры конфигурации применяются на ходу после выполнения команды, однако для некоторых требуется выполнение процедуры перезапуска кластера.

Domain-specific language (DSL) конфигурации кластера

YDB предоставляет DSL конфигурации кластера для более гибкого управления применением настроек к узлам кластера.

Селекторы

Основной сущностью DSL являются **селекторы**. Они позволяют переопределять часть конфигурации или конфигурацию целиком для определённых узлов или групп узлов. Например, они могут быть использованы для включения экспериментальной функциональности узлов определённой базы данных.
Каждый селектор представляет собой массив переопределений и расширений основной конфигурации. У каждого селектора есть поле `description`, которое может использоваться для хранения произвольной строки описания. Поле `selector` представляет собой набор правил, по которым определяется, должен ли данный селектор применяться для определённого узла, то есть для какого-либо набора лейблов. Поле `config` описывает правила переопределения.
Селекторы применяются в порядке их описания.

> **Примечание**
>
> Механизм селекторов поддерживается только для узлов баз данных.

Лейблы

Лейблы — это особые метки, с помощью которых можно пометить узлы или группы узлов. Каждый узел имеет набор автоматически задаваемых лейблов:

- `node_id` — внутренний идентификатор узла в системе;
- `node_host` — `hostname` узла, получаемый при старте;
- `tenant` — база данных, обслуживаемая данным узлом;
- `dynamic` — является ли данный узел динамическим (true/false).

И, опционально, какие-либо дополнительные лейблы, явно определённые пользователем при запуске процесса `ydbd` на узле при помощи аргументов командной строки. Например: `--label example=test`.

Пример использования селекторов

В примере ниже определяется общая конфигурация актор-системы и конфигурация для тенанта `large_tenant`. По умолчанию при такой конфигурации актор-система считает, что на узле 4 ядра, а на узлах тенанта `large_tenant` — по 16 ядер, при этом также переопределяется тип узлов для актор-системы на `COMPUTE`.

```
metadata:
  cluster: ""
  version: 8
config:
  actor_system_config:
    use_auto_config: true
    node_type: STORAGE
    cpu_count: 4

# Раздел используется как хинт при генерации возможных конфигураций при помощи команды resolve
allowed_labels:
  dynamic:
    type: string

selector_config:
- description: large_tenant has bigger nodes with 16 cpu # произвольная строка с описанием
  selector: # селектор для всех узлов тенанта large_tenant
    tenant: large_tenant
  config:
    actor_system_config: !inherit # переиспользуем исходный actor_system_config, семантика !inherit описана в
разделе ниже
      # в данном случае !inherit позволяет управлять параметром actor_system_config.use_auto_config одновреме
нно для всего кластера изменяя лишь базовую настройку
      cpu_count: 16
      node_type: COMPUTE
```

Допустимые лейблы

Отображение, в котором можно задать допустимые значения для лейблов. Раздел используется как хинт при генерации возможных конфигураций при помощи команды `resolve`. При запуске узлов значения не валидируются.

Доступны два типа лейблов:

- `string`;
- `enum`.

string

Может принимать любые значения или быть не задан.

Пример:

```
dynamic:
  type: string
host_name:
  type: string
```

enum

Может принимать значения из списка `values` или быть не задан.

Пример:

```
flavour:
  type: enum
  values:
    ? small
    ? medium
    ? big
```

Поведение селекторов

Селекторы представляют собой простой язык предикатов. Селекторы по каждому лейблу объединяются через условие **И**.

Простой селектор

Следующий селектор выберет узлы, на которых значение лейбла `label1` равно `value1` **и** лейбла `label2` — `value2`:

```
selector:
  label1: value1
  label2: value2
```

Следующий селектор выберет **ВСЕ** узлы в кластере, т.к. никаких условий не задано:

```
selector: {}
```

In

Данный оператор позволяет выбирать узлы с значением лейблов из списка.

Следующий селектор выберет все узлы, где лейбл `label1` равен `value1` **или** `value2`:

```
selector:
  label1:
    in:
    - value1
    - value2
```

NotIn

Данный оператор позволяет выбирать узлы, у которых выбранный лейбл не равен значению из списка.

Следующий селектор выберет все узлы, где лейбл `label1` равен `value1` **и** `label2` не равен `value2` **и** `value3`:

```
selector:
  label1: value1
  label2:
    not_in:
    - value2
    - value3
```

Дополнительные теги в YAML

Теги необходимы для частичного или полного переиспользования конфигураций предыдущих селекторов. С помощью них можно объеденить, расширить, удалить и полностью переопределить параметры, заданные в предыдущих селекторах и основной конфигурации.

!inherit

Область применения: YAML mapping
Действие: аналогично merge-тегу в YAML, скопировать все дочерние элементы из родительского отображения и объеденить с текущими с перезаписью
Пример:

Изначальная конфигурация	Переопределение	Результирующая конфигурация

`config:`
`  some_config:`
`  first_entry: 1`
`  second_entry: 2`
`  third_entry: 3`
	`config:`
	`  some_config: !inherit`
	`  second_entry: 100`
		`config:`
		`  some_config:`
		`  first_entry: 1`
		`  second_entry: 100`
		`  third_entry: 3`

!inherit:<key>

Область применения: YAML sequence
Действие: скопировать элементы родительского массива и переписать, трактуя объект key в элементах как ключ, дописывая новые ключи в конец
Пример:

Изначальная конфигурация	Переопределение	Результирующая конфигурация

```
config:
  some_config:
    array:
    - abc: 2
      value: 10
    - abc: 1
      value: 20
      another_value: test
```

```
config:
  some_config: !inherit
    array: !inherit:abc
    - abc: 1
      value: 30
    - abc: 3
      value: 40
```

```
config:
  some_config:
    array:
    - abc: 2
      value: 10
    - abc: 1
      value: 30
    - abc: 3
      value: 40
```

!remove

Область применения: YAML sequence element под `!inherit:<key>`
Действие: удалить элемент с соотвествующим ключом.
Пример:

Изначальная конфигурация	Переопределение	Результирующая конфигурация

`config:`
`  some_config:`
`  array:`
`  - abc: 2`
`  value: 10`
`  - abc: 1`
`  value: 20`
`  another_value: test`
	`config:`
	`  some_config: !inherit`
	`  array: !inherit:abc`
	`  - !remove`
	`  abc: 1`
		`config:`
		`  some_config:`
		`  array:`
		`  - abc: 2`
		`  value: 10`

!append

Область применения: YAML sequence
Действие: скопировать элементы родительского массива и дописать новые в конец
Пример:

Изначальная конфигурация	Переопределение	Результирующая конфигурация

`config:`
`  some_config:`
`  array:`
`  - abc: 2`
`  value: 10`
`  - abc: 1`
`  value: 20`
`  another_value: test`
	`config:`
	`  some_config: !inherit`
	`  array: !append`
	`  - abc: 1`
	`  value: 30`
	`  - abc: 3`
	`  value: 40`
		`config:`
		`  some_config:`
		`  array:`
		`  - abc: 2`
		`  value: 10`
		`  - abc: 1`
		`  value: 20`
		`  another_value: test`
		`  - abc: 1`
		`  value: 30`
		`  - abc: 3`
		`  value: 40`

Генерация конечных конфигураций

Конфигурации могут содержать сложные наборы переопределений. С помощью YDB CLI существует возможность посмотреть конечные конфигурации для:

- конкретных узлов;
- наборов лейблов;
- все возможные комбинации для текущей конфигурации.

```
# Сгенерировать все возможные конечные конфигурации для cluster.yaml
ydb admin config resolve --all -f cluster.yaml
# Сгенерировать конфигурацию для cluster.yaml при лейблах tenant=/Root/test и canary=true
ydb admin config resolve -f cluster.yaml --label tenant=/Root/test --label canary=true
# Сгенерировать конфигурацию для cluster.yaml при лейблах аналогичных текущим на узле 1001
ydb admin config resolve -f cluster.yaml --node_id 1001
# Взять текущую конфигурацию кластера и сгенерировать для него финальную конфигурацию при лейблах аналогичных текущим на узле 1001
ydb admin config resolve --from-cluster --node_id 1001
```

Более подробно команда преобразования конфигурации описана в разделе Получить конфигурацию кластера.

Пример вывода `ydb admin config resolve --all -f cluster.yaml` для следующего файла конфигурации:

```
metadata:
  cluster: ""
  version: 8
config:
  actor_system_config:
    use_auto_config: true
    node_type: STORAGE
    cpu_count: 4
allowed_labels:
  dynamic:
    type: string
selector_config:
```

```
- description: Actorsystem for dynnodes # произвольная строка с описанием
  selector: # селектор для всех узлов с лейблом dynamic = true
    dynamic: true
  config:
    actor_system_config: !inherit # переиспользуем исходный actor_system_config, семантика !inherit описана в
разделе ниже
      node_type: COMPUTE
      cpu_count: 8
```

Вывод:

```
---
label_sets: # наборы лейблов, для которых сгенерирована конфигурация
- dynamic:
    type: NOT_SET # один из трех типов лейбла: NOT_SET | COMMON | EMPTY
config: # сгенерированная конфигурация
  invalid: 1
  actor_system_config:
    use_auto_config: true
    node_type: STORAGE
    cpu_count: 4
---
label_sets:
- dynamic:
    type: COMMON
    value: true # значение лейбла
config:
  invalid: 1
  actor_system_config:
    use_auto_config: true
    node_type: COMPUTE
    cpu_count: 8
```

Параметры конфигурации кластера

Конфигурация кластера задается в YAML-файле, который передается в параметре `--yaml-config` при запуске узлов кластера. В данной статье приведено описание основных разделов конфигурации и ссылки на подробную документацию по каждому разделу.

Каждый раздел конфигурации служит определенной цели в настройке работы кластера YDB, от распределения аппаратных ресурсов до настроек безопасности и функциональных флагов. Конфигурация организована в логические группы, соответствующие различным аспектам управления кластером и его работы.

Разделы конфигурации

Доступны следующие разделы конфигурации, расположенные в алфавитном порядке:

Раздел	Обязателен	Описание
actor_system_config	Да	Распределение CPU-ресурсов по пулам акторной системы
auth_config	Нет	Настройки аутентификации и авторизации
aws_client_config	Нет	Настройки по умолчанию клиента AWS
blob_storage_config	Нет	Конфигурация статической группы кластера для системных таблеток
bridge_config	Нет	Конфигурация режима bridge
client_certificate_authorization	Нет	Аутентификация с помощью клиентских сертификатов
cms_config	Нет	Конфигурация Cluster Management System (CMS)
domains_config	Нет	Конфигурация домена кластера, включая Blob Storage и State Storage
feature_flags	Нет	Функциональные флаги для включения или отключения определённых возможностей YDB
health_check_config	Нет	Пороговые значения и таймауты сервиса Health Check
hive_config	Нет	Конфигурация запуска таблеток
host_configs	Нет	Типовые конфигурации хостов для узлов кластера
hosts	Да	Конфигурация статических узлов кластера
immediate_controls_config	Нет	Конфигурация динамических настроек кластера
kafka_proxy_config	Нет	Конфигурация Kafka Proxy
log_config	Нет	Конфигурация и параметры логирования
memory_controller_config	Нет	Распределение памяти и лимиты для компонентов базы данных
monitoring_config	Нет	Параметры YDB Monitoring
node_broker_config	Нет	Конфигурация стабильных имен узлов
query_service_config	Нет	Конфигурация внешних источников для федеративных запросов
resource_broker_config	Нет	Брокер ресурсов для контроля потребления CPU и памяти
security_config	Нет	Настройки конфигурации безопасности
self_management_config	Нет	Настройки распределённой конфигурации V2 и автоматического управления компонентами кластера
system_tablet_backup_config	Нет	Конфигурация резервного копирования системных таблеток
table_service_config	Нет	Настройки конфигурации выполнения запросов
tli_config	Нет	Параметры диагностики инвалидации блокировок транзакций (TLI)
tls	Нет	Конфигурация TLS для безопасных соединений

Практические рекомендации

Этот раздел документации посвящён полному описанию доступных настроек, а практические рекомендации по тому, что и когда настраивать, можно найти в следующих местах:

- В рамках первоначального развёртывания кластера YDB:
- Ansible
- Kubernetes
- Вручную
- В рамках поиска и устранения неисправностей
- В рамках усиления безопасности

Примеры конфигураций кластеров

Модельные конфигурации кластера для развертывания можно найти в репозитории. Изучите их перед развертыванием кластера.

Расширение кластера

Вы можете расширить кластер YDB, добавив новые узлы в конфигурацию кластера. Ниже приведены необходимые действия по расширению кластера YDB, установленного вручную на виртуальные машины или физические серверы. Расширение кластера в среде Kubernetes осуществляется путём корректировки настроек контроллера YDB для Kubernetes.

Расширение кластера YDB не требует приостановки доступа пользователей к базам данных. При расширении кластера выполняется перезапуск его компонентов для применения изменений в конфигурации, что, в свою очередь, может привести к необходимости повтора выполняемых на кластере транзакций. Повторы транзакций выполняются автоматически за счёт использования приложениями возможностей SDK YDB по контролю ошибок и повтору операций.

Подготовка новых серверов

В случае размещения новых статических или динамических узлов кластера на новых серверах, не входивших ранее в состав расширяемого кластера YDB, на каждом новом сервере необходимо выполнить установку программного обеспечения YDB в соответствии с процедурами, описанными в инструкции по развёртыванию кластеров. В частности, необходимо:

1. создать учётную запись и группу в операционной системе для работы сервиса YDB;
2. установить программное обеспечение YDB;
3. подготовить и разместить на сервере соответствующий ему ключ и сертификат TLS;
4. скопировать на сервер актуальный конфигурационный файл кластера YDB.

Используемые на новых серверах сертификаты TLS должны соответствовать требованиям к заполнению полей и быть подписаны доверенным центром регистрации, используемым на уже существующих серверах расширяемого кластера YDB.

Добавление динамических узлов

Добавление динамических узлов позволяет увеличить доступные вычислительные ресурсы (процессорные ядра и оперативную память) для выполнения пользовательских запросов кластером YDB.

Для добавления динамического узла в кластер достаточно запустить процесс, обслуживающий этот узел, передав ему в параметрах командной строки путь к конфигурационной директории кластера, имя обслуживаемой базы данных и адреса любых трёх статических узлов кластера YDB, как показано в инструкции по развёртыванию кластеров.

После успешного добавления динамического узла в кластер информация о нём будет доступна на странице мониторинга кластера в YDB UI.

Для вывода динамического узла из кластера достаточно выполнить остановку процесса динамического узла.

Добавление статических узлов

Добавление статических узлов позволяет увеличить пропускную способность при выполнении операций ввода-вывода и увеличить доступную ёмкость для хранения данных в кластере YDB.

Для добавления статических узлов в кластер необходимо выполнить следующую последовательность действий:

1. Очистить диски, которые будут использоваться для хранения данных YDB, с использованием процедуры, описанной для этапа развёртывания кластера.
2. Получить токен аутентификации для выполнения административных команд с помощью YDB CLI, например:

   ```
   ydb -e grpcs://<node1.ydb.tech>:2135 -d /Root --ca-file ca.crt \
       --user root auth get-token --force > token-file
   ```

   В примере команды выше используются следующие параметры:

   - `node1.ydb.tech` — FQDN любого из серверов, на которых уже размещены статические узлы кластера;
   - `2135` — номер порта gRPCs сервиса статических узлов;
   - `ca.crt` — имя файла с сертификатом центра регистрации;
   - `root` — логин пользователя с административными правами;
   - `token-file` — имя файла, в который сохраняется токен аутентификации для последующего использования.

   При выполнении приведённой выше команды YDB CLI запросит пароль для аутентификации указанного пользователя.
3. Получить текущую конфигурацию кластера, выполнив команду следующего вида на любом из узлов кластера:

   ```
   ydb --token-file token-file --ca-file ca.crt -e grpcs://<node1.ydb.tech>:2135 \
       admin storage fetch > config.yaml
   ```

4. Скорректировать конфигурационный файл кластера, включив в конфигурацию описание добавляемых узлов (в секции `hosts`) и используемых на них дисков (в секции `host_configs`);
5. Разрешить кластеру YDB использовать диски на новых статических узлах для хранения данных, выполнив следующую команду на любом из узлов кластера:

   ```
   export LD_LIBRARY_PATH=/opt/ydb/lib
   ydb --token-file ydbd-token-file --ca-file ca.crt -e grpcs://<node1.ydb.tech>:2135 \
       admin cluster config replace -f config.yaml
   echo $?
   ```

   В примере команды выше используются следующие параметры:

   - `ydbd-token-file` — имя файла ранее полученного токена аутентификации;
   - `2135` — номер порта gRPCs сервиса статических узлов;
   - `ca.crt` — имя файла с сертификатом центра регистрации.

   Если при выполнении приведённой выше команды возвращается ошибка сверки версии конфигурации, это означает, что версия текущего конфига устарела, и необходимо получить новый из кластера, повторив шаг 3. Пример сообщения об ошибке сверки версии конфигурации:

```
ErrorDescription: "ConfigVersion mismatch ConfigVersionProvided# 0 ConfigVersionExpected# 1"
```

6. Создать на каждой новой машине пустую директорию `/opt/ydb/cfg` для работы кластера с конфигурацией. Если на одной машине запускается несколько узлов, использовать одну и ту же директорию. Выполнив специальную команду на каждой новой машине, инициализировать эту директорию с использованием произвольного статического узла кластера в качестве источника конфигурации.

   ```
   sudo mkdir -p /opt/ydb/cfg
   sudo chown -R ydb:ydb /opt/ydb/cfg
   ydb admin node config init --config-dir /opt/ydb/cfg --seed-node <node.ydb.tech:2135>
   ```

7. Запустить процессы, обслуживающие новые статические узлы кластера, на соответствующих серверах.
8. Убедиться в том, что новые статические узлы отображаются на странице мониторинга кластера в YDB UI.
9. Добавить дополнительные группы хранения в одну или несколько баз данных, выполнив команды следующего вида на любом из узлов кластера:

   ```
   export LD_LIBRARY_PATH=/opt/ydb/lib
   /opt/ydb/bin/ydbd -f ydbd-token-file --ca-file ca.crt -s grpcs://`hostname -f`:2135 \
       admin database /Root/testdb pools add ssd:1
   echo $?
   ```

   В примере команды выше используются следующие параметры:

   - `ydbd-token-file` - имя файла ранее полученного токена аутентификации;
   - `2135` - номер порта grpcs сервиса статических узлов;
   - `ca.crt` - имя файла с сертификатом центра регистрации;
   - `/Root/testdb` - полный путь к базе данных;
   - `ssd:1` - имя пула хранения и количество выделяемых групп хранения.

10. Убедиться, что добавленные группы хранения отображаются на странице мониторинга кластера в YDB UI.

Вывод статических узлов из кластера YDB производится в соответствии с документированной процедурой декомиссии.

В случае повреждения и невозможности ремонта сервера, на котором работает статический узел кластера, необходимо разместить недоступный статический узел на новом сервере, содержащем аналогичное или большее количество и объем дисков.

Удаление узла из кластера

> **Важно**
>
> Эта статья посвящена кластерам YDB, в которых используется конфигурация V2. Данный способ конфигурирования пока является экспериментальным и доступен только для версий YDB начиная с v25.1. Для использования в продакшене мы рекомендуем выбирать конфигурацию V1 — она является основной и официально поддерживаемой для всех кластеров YDB.

В этой статье описано удаление динамического или статического узла из кластера YDB, развёрнутого вручную на виртуальных машинах или физических серверах. Удаление узлов из кластера, развёрнутого в Kubernetes, в этой инструкции не рассматривается.

Удаление динамического узла

Для удаления динамического узла не требуется изменять конфигурацию кластера.

Чтобы удаление динамического узла не повлияло на выполнение запросов:

1. Выполните мягкий перенос таблеток с узла и дождитесь его завершения.
2. Остановите процесс YDB на узле. Сначала проверьте, что процесс можно безопасно остановить, а затем остановите его.

После остановки процесса проверьте на вкладке **Nodes** страницы мониторинга кластера, что удалённый узел больше не отображается активным. Динамический узел может исчезнуть из списка сразу или временно отображаться со статусом **Disconnected** до истечения аренды его идентификатора в NodeBroker. Если узел снова переходит в активное состояние, убедитесь, что процесс был остановлен именно на нужном хосте и не настроен на автоматический перезапуск, например службой `systemd` или супервизором.

Удаление статического узла

Статические узлы обслуживают систему хранения и перечислены в секции `hosts`. На дисках статического узла могут находиться VDisk динамических и статических групп, а на самом узле — реплики State Storage, Board и SchemeBoard. Поэтому сначала необходимо перенести эти ресурсы, а затем удалить узел из конфигурации.

Перед началом процедуры проверьте во встроенном UI, что затронутые группы хранения работоспособны, то есть все VDisk этих групп отображаются в состоянии `Ok` (выделены зелёным цветом), и ни один VDisk не находится в состоянии `Error` или `Degraded`.

На оставшихся узлах должно быть достаточно свободного места и слотов на PDisk для всех VDisk с удаляемого узла. Размещение VDisk по доменам отказа и областям отказа должно соответствовать используемой схеме кодирования, чтобы после удаления узла сохранялась отказоустойчивость групп. Расчёт необходимого запаса приведён в статье Оценка требуемого оборудования.

SelfHeal для динамических групп включён по умолчанию. Перед удалением узла убедитесь, что он также включён для остальных ресурсов, размещённых на этом узле:

- Если на узле есть VDisk статической группы, включите SelfHeal статической группы. Как альтернативу можно перенести VDisk статической группы с узла вручную, см. Перемещение статической группы.
- Если узел содержит реплики State Storage, Board или SchemeBoard, включите SelfHeal подсистем распространения метаданных. Как альтернативу можно перенести эти реплики с узла вручную, см. Конфигурирование подсистем распространения метаданных State Storage, Board, Scheme Board.

Чтобы удалить статический узел:

1. Если на узле работают таблетки, выполните их мягкий перенос.
2. Проверьте, что процесс можно безопасно остановить, затем остановите его.
3. Дождитесь, пока SelfHeal перенесёт VDisk с узла. С настройками по умолчанию перенос начинается приблизительно через час после остановки узла. Чтобы запустить перенос немедленно, сначала получите идентификаторы всех PDisk удаляемого узла с помощью YDB DSTool:

   ```
   ydb-dstool -e <bs_endpoint> pdisk list --columns NodeId:PDiskId FQDN Path
   ```

   `<bs_endpoint>` — точка подключения к любому доступному узлу хранения в формате `[PROTOCOL://]HOST[:PORT]`, например `http://node1.example.com:8765`.

   Выберите все строки, в которых `FQDN` совпадает с именем хоста удаляемого узла. Проверьте пути к дискам в столбце `Path` и сохраните все соответствующие значения `NodeId:PDiskId`.

   Затем переведите все найденные PDisk в статус `BROKEN` одной командой:

   ```
   ydb-dstool -e <bs_endpoint> pdisk set --status BROKEN --unavail-as-offline --pdisk-ids "<pdisk_id_1>"
   ... "<pdisk_id_N>"
   ```

   Используйте ту же точку подключения `<bs_endpoint>`. Вместо `"<pdisk_id_1>" ... "<pdisk_id_N>"` перечислите через пробел все сохранённые идентификаторы дисков в формате `[NodeId:PDiskId]`, например `"[3:1]" "[3:2]"` для двух дисков узла с `NodeId`, равным `3`. Флаг `--unavail-as-offline` позволяет считать PDisk, недоступные через службу мониторинга Whiteboard, неработающими.

   Команда выполняется на переднем плане. Дождитесь её успешного завершения, затем проверьте завершение переноса данных на следующем шаге. Подробнее см. в статье Перенос VDisk с повреждённого или недоступного тома блочного хранилища.
4. Во встроенном UI проверьте, что на удаляемом узле не осталось VDisk, а затронутые группы хранения работоспособны (все VDisk находятся в состоянии `Ok`). Если с узла переносились реплики State Storage, Board или SchemeBoard, проверьте, что перенос завершён.
5. Получите актуальную конфигурацию кластера с помощью команды ydb admin cluster config fetch:

   ```
   ydb [global options...] admin cluster config fetch > config.yaml
   ```

6. Если запись удаляемого узла не последняя в списке `hosts`, при удалении позиции всех следующих узлов сместятся. Чтобы сохранить их идентификаторы, для каждой следующей записи без явно заданного `node_id` укажите её текущий идентификатор до удаления: для такой записи он равен текущей позиции в списке, начиная с `1`. Если `node_id` уже задан явно, сохраните его существующее значение. Если удаляется последняя запись, этот шаг не требуется.

   > **Важно**
   >
   > Корректная нумерация узлов критически важна для работоспособности кластера. Ошибка при назначении `node_id` может привести к тому, что VDisk, реплики State Storage, Board или SchemeBoard окажутся привязаны не к тому хосту, что может стать причиной необратимой потери данных.

   Например, дана следующая конфигурация, из которой удаляется узел `node3`:

   ```
   hosts:
   - host: node1
   - host: node2
   - host: node3 # удаляется
   - host: node4
   - host: node5
     node_id: 50
   ```

   Перед удалением `node3` явно укажите `node_id: 4` для `node4`, чтобы его идентификатор не изменился на `3`. Для `node5` сохраните уже заданное значение `node_id: 50`. После удаления `node3` список будет выглядеть так:

   ```
   hosts:
   - host: node1
   - host: node2
   - host: node4
     node_id: 4
   - host: node5
     node_id: 50
   ```

7. Удалите запись узла из секции `hosts`.
8. Примените конфигурацию с помощью команды ydb admin cluster config replace:

   ```
   ydb [global options...] admin cluster config replace -f config.yaml
   ```

   Если команда завершилась с ошибкой

   Если на PDisk удаляемого узла остались VDisk, команда возвращает ошибку следующего вида:

   ```
   failed to remove PDisk# 1:1 as it has active VSlots
   ```

   В этом случае дождитесь, пока SelfHeal перенесёт оставшиеся VDisk. Продолжительность переноса зависит от объёма данных и производительности дисков. Следите за переносом на вкладке **Storage** удаляемого узла во встроенном UI. Когда на узле не останется VDisk, повторите команду `config replace` с тем же файлом.

   Если список VDisk не сокращается и репликация не идёт, перенесите оставшиеся VDisk вручную.

После успешного применения конфигурации сервер и его диски можно вывести из эксплуатации.

Конфигурирование подсистем распространения метаданных State Storage, Board, Scheme Board

Применяется если нужно изменить конфигурацию подсистем распространения метаданных состоящую из State Storage, Board, Scheme Board, на кластере YDB.

При использовании конфигурации V2 подсистемы распространения метаданных частично поддерживаются автоматически за счёт механизма SelfHeal — см. SelfHeal подсистем распространения метаданных (перенос и добавление реплик при изменениях топологии). Чтобы отключить это поведение, установите `state_storage_self_heal_config.enable` в `false`, как описано в том же разделе. Отключение не обязательно для нижеприведённых шагов: конфигурация после `ydb admin cluster config replace` применится до очередного срабатывания автоматики.

> **Важно**
>
> Ошибка в конфигурации подсистем распространения метаданных (в том числе в секции `domains_config`) или неправильная последовательность изменений могут привести к недоступности кластера YDB.

Для ручного изменения конфигурации State Storage в кластере YDB необходимо выполнить следующие шаги.

1. Получить текущую конфигурацию кластера с помощью команды ydb admin cluster config fetch:

```
ydb [global options...] admin cluster config fetch --v2-internal-state > config.yaml
```

В результате выполнения данной команды текущая конфигурация будет сохранена в файле `config.yaml`

2. Внести требуемые изменения в секции `domains_config` конфигурационного файла `config.yaml`:
   Правила изменения секции `domains_config` см. в разделе Правила конфигурирования State Storage.
3. Применить новую конфигурацию кластера с помощью ydb admin cluster config replace:

```
ydb [global options...] admin cluster config replace -f config.yaml
```

Правила конфигурирования State Storage

Перечисленные ниже правила применяются к `state_storage` и к раздельным полям `explicit_state_storage_config`, `explicit_state_storage_board_config`, `explicit_scheme_board_config` в секции `domains_config` файла `config.yaml` (см. Конфигурация State Storage). Ключи `explicit_*` соответствуют по отдельности State Storage, Board и Scheme Board.

Под кольцом в конфигурации понимается блок `ring` внутри элемента списка `ring_groups` (см. Конфигурация State Storage).

Изменение конфигурации производится в несколько шагов. Сначала добавляется новая группа колец, состоящая из правильно выбранных узлов (в соответствии с моделью отказа), а затем старая группа колец удаляется.

Чтобы избежать недоступности кластера, удаление и добавление групп колец выполняйте строго в последовательности шагов, описанной ниже.

1. Чтобы изменить конфигурацию подсистем распространения метаданных без недоступности кластера необходимо производить это путем добавления и удаления групп колец.
2. Добавлять и удалять можно только группы колец с параметром `WriteOnly: true`.
3. В новой конфигурации всегда должна присутствовать хотя бы одна группа колец из предыдущей конфигурации без параметра `WriteOnly`. Такая группа колец должна идти первой в списке.
4. Если в разных группах колец используются одни и те же узлы кластера, добавьте параметр `ring_group_actor_id_offset:1` к группе колец. Значение должно быть уникальным среди групп колец.

   Этот параметр сделает уникальными идентификаторы реплик в данной группе колец, они не будут совпадать с идентификаторами из других групп, и это позволит разместить на одном узле кластера несколько реплик одного типа.
5. Переход к новой конфигурации выполняется за 4 последовательных шага. На каждом шаге подготавливается новая конфигурация и применяется к кластеру.

   Только что созданные или готовые к удалению группы колец помечаются флагом `WriteOnly: true`. Это необходимо, чтобы запросы на чтение обрабатывались уже развернутой группой колец, пока новая конфигурация распространится на необходимое количество узлов, создадутся новые реплики или удалятся старые.

   Поэтому между шагами необходимо делать паузу как минимум в `1 минуту`.

   - Добавляем новую группу колец с параметром `WriteOnly: true` соответствующую целевой конфигурации.
   - Снимаем флаг `WriteOnly`.
   - Выставляем флаг `WriteOnly: true` на группу колец соответствующую старой конфигурации, новую группу колец переносим в начало списка групп колец.
   - Удаляем старую группу колец.

Пример

Рассмотрим на примере текущей конфигурации:

```
config:
  domains_config:
    explicit_scheme_board_config:
      ring:
        nto_select: 5
        node: [1,2,3,4,5,6,7,8]
```

и целевой конфигурации:

```
config:
  domains_config:
```

```
    explicit_scheme_board_config:
      ring:
        nto_select: 5
        node: [10,20,30,40,5,6,7,8]
```

Мы хотим перенести часть реплик на другие узлы кластера.

Шаг 1
На первом шаге подготовьте `ring_groups`, следуя правилам конфигурирования: первая группа колец совпадает с **текущей** конфигурацией из листингов выше, вторая — с **целевой** и помечена `WriteOnly: true`. Параметр `ring_group_actor_id_offset` указывайте так, как описано в тех же правилах, если наборы узлов у групп совпадают.

```
config:
  domains_config:
    explicit_scheme_board_config:
      ring_groups:
        - ring:
          nto_select: 5
          node: [1,2,3,4,5,6,7,8]
        - ring:
          nto_select: 5
          node: [10,20,30,40,5,6,7,8]
          write_only: true
          ring_group_actor_id_offset: 1
```

Шаг 2
Снимаем флаг `WriteOnly`.

```
config:
  domains_config:
    explicit_scheme_board_config:
      ring_groups:
        - ring:
          nto_select: 5
          node: [1,2,3,4,5,6,7,8]
        - ring:
          nto_select: 5
          node: [10,20,30,40,5,6,7,8]
          ring_group_actor_id_offset: 1
```

Шаг 3
Делаем новую группу колец первой в списке. На старую конфигурацию выставляем флаг `WriteOnly: true`

```
config:
  domains_config:
    explicit_scheme_board_config:
      ring_groups:
        - ring:
          nto_select: 5
          node: [10,20,30,40,5,6,7,8]
          ring_group_actor_id_offset: 1
        - ring:
          nto_select: 5
          node: [1,2,3,4,5,6,7,8]
          write_only: true
```

Шаг 4
Применяем на кластере целевую конфигурацию:

```
config:
  domains_config:
    explicit_scheme_board_config:
      ring_groups:
        - ring:
          nto_select: 5
          node: [10,20,30,40,5,6,7,8]
          ring_group_actor_id_offset: 1
```

Проверка результата

Проверить, что изменения применились, можно в разделе `CMS` в YDB UI кластера (доступен на порту 8765): перейдите на вкладку `Tablets` и убедитесь по репликам таблеток подсистем метаданных, что конфигурация подхватилась.

Перемещение State Storage

Статья находится в разработке

> **Важно**
>
> Эта статья посвящена кластерам YDB, в которых используется конфигурация V2. Данный способ конфигурирования пока является экспериментальным и доступен только для версий YDB начиная с v25.1. Для использования в продакшене мы рекомендуем выбирать конфигурацию V1 — она является основной и официально поддерживаемой для всех кластеров YDB.

Если нужно вывести из эксплуатации узел кластера YDB, на котором располагается часть State Storage, необходимо переместить её на другой узел.

> **Важно**
>
> Ошибка в конфигурации подсистем распространения метаданных (в том числе в секции `domains_config`) или неправильная последовательность изменений могут привести к недоступности кластера YDB.

При использовании Конфигурации V2 конфигурирование State Storage осуществляется автоматически. Для внесения изменений в конфигурацию необходимо выполнить следующие действия: остановить автоматическое управление конфигурацией State Storage, получить акутальную конфигурацию State Storage, изменить актуальную конфигурацию желаемым способом и указать ее в явном виде как целевую, после чего применить ее, убедиться в примененеии, выполнить вывод узла из эксплуатации, убрать явное указание конфигурации State Storage и включить автоматическое конфигурирование State Storage.

В качестве примера рассмотрим кластер YDB в котором узел 1 входит в состав State Storage, а узел 10 - не входит, и будем считать что цель изменения конфигурации State Storage заключается в выводе узла 1 из эксплуатации.

1. Остановить автоматическое управление конфигурацией State Storage
2. Получить акутальную конфигурацию State Storage
3. Изменить актуальную конфигурацию желаемым способом и указать ее в явном виде как целевую

   Допутсим, получена следующая конфигурациея State Storage:

   ```
   ...
     state_storage:
     - ring:
         node: [1, 2, 3, 4, 5, 6, 7, 8, 9]
         nto_select: 9
       ssid: 1
   ...
   ```

   На ухле с `node_id:1` сконфигурирован и запущен статический узел кластера, который обслуживает часть State Storage. Предположим, нам нужно вывести из эксплуатации этот узел.

   Для замены `node_id:1` мы используем другой хост с развернутым на нем статическим узлом с `node_id:10`.

   Чтобы переместить State Storage с узла `node_id:1` на `node_id:10`, в конфигурационном файле `config.yaml` измените список узлов `node`, заменив идентификатор удаляемого узла на идентификатор добавляемого:

   ```
   ...
     state_storage:
     - ring:
         node: [10, 2, 3, 4, 5, 6, 7, 8, 9]
         nto_select: 9
       ssid: 1
   ...
   ```

4. Применить конфигурацию
5. Убедиться в примененеии
   Учтите, что наполненприменение новых узлов State Storage после реконфигурации происходит с задержкой не менее 15 секунд.
6. Выполнить вывод узла из эксплуатации
7. Убрать явное указание конфигурации State Storage
8. Включить автоматическое конфигурирование State Storage

SelfHeal статической группы

> **Важно**
>
> Эта статья посвящена кластерам YDB, в которых используется конфигурация V2. Данный способ конфигурирования пока является экспериментальным и доступен только для версий YDB начиная с v25.1. Для использования в продакшене мы рекомендуем выбирать конфигурацию V1 — она является основной и официально поддерживаемой для всех кластеров YDB.

При использовании конфигурации V2 механизм SelfHeal может автоматически переносить VDisk статической группы с неисправных PDisk и восстанавливать отказоустойчивость группы.

> **Важно**
>
> На кластерах с конфигурацией V1 SelfHeal статической группы включить нельзя.

Общий механизм SelfHeal обнаруживает неисправный PDisk и инициирует перенос VDisk. Для динамических групп конфигурацию изменяет Blob Storage Controller, а конфигурацию статической группы изменяет распределённая конфигурация.

Чтобы разрешить распределённой конфигурации автоматически изменять статическую группу, включите параметр `self_management_config.automatic_static_group_management`. По умолчанию этот параметр выключен.

Включение и выключение SelfHeal статической группы

Для работы SelfHeal статической группы должны быть включены:

- распределённая конфигурация V2 — `self_management_config.enabled: true`;
- общий механизм SelfHeal, который включён по умолчанию.

Параметр `self_management_config.enabled` включает саму распределённую конфигурацию. Параметр `self_management_config.automatic_static_group_management` отдельно разрешает автоматический перенос VDisk статической группы.

Чтобы включить или выключить автоматическое управление статической группой:

1. Получите текущую конфигурацию кластера с помощью команды ydb admin cluster config fetch:

   ```
   ydb [global options...] admin cluster config fetch > config.yaml
   ```

2. В конфигурационном файле `config.yaml` установите значения параметров `self_management_config.enabled` и `self_management_config.automatic_static_group_management`:

   ```
   config:
     self_management_config:
       enabled: true
       automatic_static_group_management: true
   ```

   Значение `self_management_config.automatic_static_group_management: true` включает автоматическое управление статической группой, а `false` — выключает.

3. Примените новую конфигурацию с помощью команды ydb admin cluster config replace:

   ```
   ydb [global options...] admin cluster config replace -f config.yaml
   ```

Ограничение набора целевых узлов

По умолчанию SelfHeal может перенести VDisk статической группы на любой подходящий узел. Чтобы ограничить набор целевых узлов, укажите их идентификаторы в параметре `static_group_self_heal_allowed_nodes`:

```
config:
  self_management_config:
    enabled: true
    automatic_static_group_management: true
    static_group_self_heal_allowed_nodes:
    - 1
    - 2
    - 3
```

Пустой список означает отсутствие ограничений. Для каждого разрешённого узла выберите подходящий PDisk и проверьте, что его свободная ёмкость не меньше объёма, занятого VDisk на исходном PDisk, плюс эксплуатационный запас до порога предупреждения о заполнении. Единого числового значения запаса нет: ориентируйтесь на порог предупреждения, используемый в мониторинге вашего кластера. Размещение после переноса должно соответствовать модели отказа.

Перемещение статической группы

Статья находится в разработке

> **Важно**
>
> Эта статья посвящена кластерам YDB, в которых используется конфигурация V2. Данный способ конфигурирования пока является экспериментальным и доступен только для версий YDB начиная с v25.1. Для использования в продакшене мы рекомендуем выбирать конфигурацию V1 — она является основной и официально поддерживаемой для всех кластеров YDB.

При использовании Конфигурации V2 управление статической группой выполняется автоматически: механизм SelfHeal реконфигурирует группу при отказе одного из её узлов.

Если требуется управлять конфигурацией статической группы вручную, отключите автоматическое управление, получите текущую конфигурацию, внесите изменения и примените её как целевую. Затем удалите целевую конфигурацию статической группы из файла конфигурации и снова включите автоматическое управление.

> **Важно**
>
> Ошибка в конфигурации подсистем распространения метаданных (в том числе в секции `domains_config`) или неправильная последовательность изменений могут привести к недоступности кластера YDB.

В качестве примера рассмотрим кластер YDB, в котором на хосте с `node_id:1` сконфигурирован и запущен статический узел. Этот узел обслуживает часть статической группы.

Фрагмент конфигурации статической группы:

```
...
groups:
  ...
  rings:
    ...
    fail_domains:
    - vdisk_locations:
      - node_id: 1
        path: /dev/vda
        pdisk_category: SSD
    ...
...
```

Для замены `node_id:1` используйте другой хост с развернутым на нем статическим узлом с `node_id:10`.

Чтобы переместить часть статической группы с хоста `node_id:1` на `node_id:10`:

1. Выключите автоматическое управление статической группой.
2. Получите текущую конфигурацию статической группы.
3. Внесите изменения и примените измененную конфигурацию в качестве целевой конфигурации статической группы.

   В конфигурационном файле `config.yaml` измените значение `node_id`, заменив идентификатор удаляемого хоста на идентификатор добавляемого:

   ```
   ...
   groups:
     ...
     rings:
       ...
       fail_domains:
       - vdisk_locations:
         - node_id: 10
           path: /dev/vda
           pdisk_category: SSD
       ...
   ...
   ```

   Измените путь `path` и категорию `pdisk_category` диска, если на хосте с `node_id: 10` они отличаются.
4. Перейдите на страницу мониторинга YDB UI и убедитесь, что VDisk статической группы появился на целевом физическом диске и реплицируется. Подробнее см. Мониторинг статической группы.
5. Уберите целевую конфигурацию статической группы из файла конфигурации и включите автоматическое управление конфигурацией статической группы.

Замена FQDN статического узла

> **Важно**
>
> Эта статья посвящена кластерам YDB, в которых используется конфигурация V2. Данный способ конфигурирования пока является экспериментальным и доступен только для версий YDB начиная с v25.1. Для использования в продакшене мы рекомендуем выбирать конфигурацию V1 — она является основной и официально поддерживаемой для всех кластеров YDB.

Полное доменное имя (Fully Qualified Domain Name, FQDN) статического узла задаётся параметром `host` в секции `hosts`. Чтобы изменить FQDN без замены сервера и дисков, выполните следующие действия.

Создайте DNS-запись для нового FQDN, указывающую на тот же сервер. Новый FQDN должен разрешаться со всех узлов кластера. Не удаляйте старую DNS-запись до окончания процедуры. Если кластер использует TLS, подготовьте сертификат узла, содержащий новый FQDN.

1. Получите текущую конфигурацию кластера с помощью команды ydb admin cluster config fetch:

   ```
   ydb [global options...] admin cluster config fetch > config.yaml
   ```

2. В файле `config.yaml` найдите существующий элемент списка `hosts` для заменяемого узла и измените только его значение `host`. Не удаляйте запись узла и не создавайте новую в другом месте — это изменит позицию узла в конфигурации и нарушит обновление.

   Например, для узла с идентификатором `1` замените `node-1.example.com` на `node-1-new.example.com`:

   ```
   config:
     hosts:
     - host: node-1-new.example.com
       node_id: 1
   ```

3. Примените новую конфигурацию с помощью команды ydb admin cluster config replace:

   ```
   ydb [global options...] admin cluster config replace -f config.yaml
   ```

4. Дождитесь, пока в локальной копии файла `config.yaml` на заменяемом узле появится новый FQDN.
5. Подготовьте заменяемый узел к перезапуску:
   - Если кластер использует TLS, установите подготовленный сертификат.
   - Если узел запускается с параметром `--node static`, убедитесь, что новое значение `host` совпадает с выводом команды `hostname` или `hostname -f`. Если значения не совпадают, перед перезапуском настройте операционную систему так, чтобы команда `hostname -f` возвращала новое значение `host`.
6. Получите разрешение на остановку процесса и перезапустите процесс статического узла. Например, при использовании `systemd`:

   ```
   sudo systemctl restart ydbd-storage
   ```

   Имя сервиса может отличаться в зависимости от способа развёртывания.
7. На вкладке **Nodes** страницы мониторинга кластера убедитесь, что узел подключился с прежним идентификатором и новым FQDN. После этого можно удалить старую DNS-запись.

Аутентификация и авторизация узлов баз данных

Аутентификация узлов в кластере YDB обеспечивает проверку подлинности узлов баз данных при выполнении служебных вызовов к другим узлам по протоколу gRPC. Авторизация узлов обеспечивает проверку и предоставление необходимых полномочий при обработке служебных вызовов, включая операции регистрации запускаемых узлов в кластере и доступа к конфигурации. Использование аутентификации и авторизации узлов рекомендуется для всех кластеров YDB, поскольку позволяет избежать ситуаций несанкционированного доступа к данным через включение в кластер контролируемых злоумышленниками узлов.

Аутентификация и авторизация узлов баз данных осуществляется в следующем порядке:

1. Запускаемый узел базы данных открывает gRPC-подключение к одному из узлов хранения кластера, указанных в опции командной строки `--node-broker`. При подключении используется протокол TLS, и в настройках подключения в качестве клиентского сертификата используется сертификат запускаемого узла.
2. Узел хранения и узел базы данных осуществляют взаимные проверки подлинности с использованием протокола TLS: производится проверка цепочки доверия сертификата и проверка соответствия имени хоста значению поля «Subject Name» сертификата.
3. Узел хранения проверяет заполнение поля «Subject» сертификата на соответствие требованиям, устанавливаемым настройками в статической конфигурации.
4. При успешном прохождении перечисленных выше проверок подключение со стороны узла базы данных считается аутентифицированным, и ему присваивается идентификатор субъекта доступа — SID, определяемый настройками.
5. Узел базы данных использует установленное gRPC-подключение для регистрации в составе кластера с использованием соответствующего служебного вызова. При регистрации узел базы данных передаёт свой сетевой адрес, предназначенный для взаимодействия с другими узлами кластера.
6. Узел хранения проверяет наличие SID, присвоенного gRPC-подключению, в списке допустимых. При успешном прохождении этой проверки узел хранения регистрирует узел базы данных в кластере, сопоставляя полученный сетевой адрес с идентификатором узла.
7. Узел базы данных входит в кластер, подключаясь через свой сетевой адрес и указывая идентификатор узла, полученный при регистрации. Попытки входа в кластер узлов с неизвестными сетевыми адресами или идентификаторами блокируются другими узлами.

Далее описаны настройки, необходимые для включения функции аутентификации и авторизации узлов.

Пререквизиты для настройки

1. В развёрнутом кластере YDB должно быть настроено шифрование трафика gRPC с использованием протокола TLS.
2. При подготовке сертификатов узлов для кластера, в котором планируется использовать функции аутентификации и авторизации узлов, необходимо обеспечить единые правила заполнения поля «Subject» сертификатов, позволяющие идентифицировать сертификаты, выпущенные для узлов кластера. Более подробная информация приведена в документации по настройке правил проверки сертификатов.

   > **Примечание**
   >
   > Предлагаемый пример скрипта для генерации самоподписанных сертификатов узлов YDB заполняет поле «Subject» значением `O=YDB` для всех сертификатов узлов. Приведённые далее примеры настроек подготовлены для сертификатов с именно таким заполнением поля «Subject».

3. В параметры командной строки для запуска узлов баз данных необходимо добавить опции, задающие пути к файлам сертификатов доверенных центров сертификации, сертификата узла и ключа узла. Список дополнительных опций приведён в таблице ниже.

Опция командной строки	Описание
`--grpc-ca`	Путь к файлу сертификатов доверенных центров сертификации `ca.crt`.
`--grpc-cert`	Путь к файлу сертификата узла `node.crt`.
`--grpc-key`	Путь к файлу секретного ключа узла `node.key`.

   Пример команды для запуска узла базы данных с опциями, указывающими пути к ключам и сертификатам TLS для протокола gRPC:

   ```
   /opt/ydb/bin/ydbd server --config-dir /opt/ydb/cfg --tenant /Root/testdb \
       --grpcs-port 2136 --grpc-ca /opt/ydb/certs/ca.crt \
       --grpc-cert /opt/ydb/certs/node.crt --grpc-key /opt/ydb/certs/node.key \
       --ic-port 19002 --ca /opt/ydb/certs/ca.crt \
       --mon-port 8766 --mon-cert /opt/ydb/certs/web.pem \
       --node-broker grpcs://<ydb1>:2135 \
       --node-broker grpcs://<ydb2>:2135 \
       --node-broker grpcs://<ydb3>:2135
   ```

Включение режима аутентификации и авторизации узлов

> **Примечание**
>
> Начиная с версии 25.2, корректно настроенная аутентификация узлов баз данных является обязательной.
>
> Регистрация узлов в кластере невозможна без выполнения аутентификации.

Для включения обязательной авторизации узлов баз данных в файл конфигурации кластера необходимо добавить следующие блоки настроек:

1. На корневом уровне конфигурации добавьте блок `client_certificate_authorization`, в котором укажите требования к заполнению поля "Subject" доверенных сертификатов подключаемых узлов, например:

```
client_certificate_authorization:
  request_client_certificate: true
  client_certificate_definitions:
    - member_groups: ["registerNode@cert"]
      subject_terms:
      - short_name: "O"
        values: ["YDB"]
```

При необходимости добавьте другие проверки сертификатов в соответствии с документацией.

В случае успешной проверки сертификата и соответствия компонентов поля "Subject" сертификата требованиям, установленным в блоке `subject_terms`, подключению будут присвоены субъекты доступа, перечисленные в параметре `member_groups`. Чтобы отделить такие субъекты доступа от других групп и учётных записей, к их наименованию добавляется суффикс `@cert`.

2. В блок настроек аутентификации кластера `security_config` добавьте элемент `register_dynamic_node_allowed_sids`, указав список субъектов доступа, которым разрешена регистрация узлов баз данных. Пример:

```
security_config:
  enforce_user_token_requirement: true
  ...
  register_dynamic_node_allowed_sids:
  - "registerNode@cert"
```

Более подробная информация по настройке параметров аутентификации кластера приведена в соответствующем разделе документации.

3. В блоке настроек `auth_config` присвойте параметру `node_registration_token` пустую строку. Пример:

```
auth_config:
  ...
  node_registration_token: ""
```

4. Обновите файлы статической конфигурации на всех узлах кластера вручную либо с помощью плейбука Ansible.
5. Выполните поэтапный перезапуск узлов хранения кластера с помощью ydbops либо с использованием плейбука Ansible.
6. Выполните поэтапный перезапуск узлов баз данных кластера с помощью ydbops либо с использованием плейбука Ansible.

Миграция конфигурации кластера

В YDB поддерживается два механизма управления конфигурацией: V1 и V2 (экспериментальный, доступен с версии 25.1). Ключевые различия между ними описаны в статье Сравнение конфигураций V1 и V2.

Совет

Новые кластеры YDB рекомендуется разворачивать сразу с использованием конфигурации V2. Если кластер был развёрнут на конфигурации V1, то она продолжит использоваться даже после обновления на версию YDB 25.1 или выше. После такого обновления рекомендуется запланировать и провести миграцию на V2, так как в будущих версиях YDB поддержка V1 будет прекращена. Узнать, какая версия конфигурации используется на кластере, можно по инструкции.

В зависимости от текущего состояния вашего кластера, вы можете выполнить миграцию:

- **На конфигурацию V2:** Если ваш кластер работает под управлением конфигурации V1, вы можете перейти на экспериментальный механизм конфигурации V2.
- **На конфигурацию V1:** Если после перехода на конфигурацию V2 возникли непредвиденные проблемы, вы можете выполнить обратную миграцию на ручное управление конфигурацией (V1).

Проблемы при миграции?

Если при использовании инструкций по миграции (особенно при откате на V1) возникли непредвиденные проблемы, рекомендуется незамедлительно сообщать о них в виде GitHub issue, предоставляя максимум контекста и диагностики для воспроизведения.

Прежде чем выполнять миграцию, убедитесь, какая версия конфигурации используется в вашем кластере в данный момент, с помощью инструкции по проверке.

Миграция на конфигурацию V2

> **Важно**
>
> Эта статья посвящена кластерам YDB, в которых используется конфигурация V2. Данный способ конфигурирования пока является экспериментальным и доступен только для версий YDB начиная с v25.1. Для использования в продакшене мы рекомендуем выбирать конфигурацию V1 — она является основной и официально поддерживаемой для всех кластеров YDB.

Данный документ содержит инструкцию по миграции с конфигурации V1 на конфигурацию V2.

В конфигурации V1 существует два различных механизма применения конфигурационных файлов:

- статическая конфигурация управляет узлами хранения кластера YDB и требует ручного размещения файлов на каждом узле кластера;
- динамическая конфигурация управляет узлами базы данных кластера YDB и загружается в кластер централизованно с помощью команд YDB CLI.

В конфигурации V2 этот процесс унифицирован: единый конфигурационный файл загружается в систему через команды YDB CLI, автоматически доставляясь на все узлы кластера.

Компоненты State Storage и статической группы кластера YDB являются ключевыми для корректной работы кластера. При работе с конфигурацией V1 данные компоненты настраиваются вручную через задание секций `domains_config` и `blob_storage_config` в конфигурационном файле.
В конфигурации V2 возможна автоматическая конфигурация этих компонентов без указания соответствующих секций в конфигурационном файле.

Исходное состояние

Миграция на конфигурацию V2 может быть осуществлена в случае выполнения следующих условий:

1. Кластер YDB обновлён до версии 25.1 и выше.
2. Кластер YDB сконфигурирован с файлом конфигурации V1 `config.yaml`, расположенным в файловой системе узлов и подключённым через аргумент `ydbd --yaml-config`.
3. В конфигурационном файле кластера заданы разделы `domains_config` и `blob_storage_config` для настройки State Storage и статической группы соответственно.

Проверка текущей версии конфигурации

Перед началом миграции убедитесь, что ваш кластер работает на конфигурации V1. Узнать текущую версию конфигурации на узлах можно несколькими способами, описанными в статье Проверка версии конфигурации.

Продолжать выполнение данной инструкции следует только в том случае, если узлы работают на версии конфигурации V1. Если на всех узлах уже включена версия V2, миграция не требуется.

Инструкция по миграции на конфигурацию V2

Для того чтобы перевести кластер YDB на конфигурацию V2, необходимо проделать следующие шаги:

1. Проверить наличие файла динамической конфигурации в кластере. Для этого необходимо выполнить команду ydb admin cluster config fetch:

   ```
   ydb -e grpc://<node.ydb.tech>:2135 admin cluster config fetch > config.yaml
   ```

   В случае отсутствия такой конфигурации в кластере команда выдаст сообщение:

   ```
   No config returned.
   ```

   Если файл найден, следует использовать его и пропустить следующий шаг данной инструкции. Перед дальнейшими шагами обязательно проверьте, что найденный файл содержит настройки, которые в конфигурации V1 находились в статическом конфигурационном файле на узлах хранения.
2. В случае отсутствия файла динамической конфигурации в кластере выполнить команду генерации файла динамической конфигурации ydb admin cluster config generate. Файл будет сгенерирован на основе файла статической конфигурации, расположенного на узлах кластера.

   ```
   ydb -e grpc://<node.ydb.tech>:2135 admin cluster config generate > config.yaml
   ```

3. Проверить, что полученный на шаге 1 или 2 файл `config.yaml` содержит настройки узлов хранения из актуальной статической конфигурации V1. Если файл был получен из существующей динамической конфигурации, эти настройки могут отсутствовать, так как в V1 динамическая конфигурация могла не содержать параметры узлов хранения.

   Цель этого шага — получить единый объединенный конфигурационный файл, который содержит как параметры из динамической конфигурации, так и настройки из статической конфигурации.

   Перенесите из статического конфигурационного файла в секцию `config` файла `config.yaml` все секции и параметры, которые описывают узлы хранения, диски, State Storage, статическую группу и профили каналов. В том числе проверьте наличие следующих секций и параметров, если они заданы в статической конфигурации V1:

   - `hosts`;
   - `host_configs`;
   - `domains_config`;
   - `blob_storage_config`;
   - `channel_profile_config`;
   - другие связанные параметры storage-конфигурации, например `static_erasure`.
4. Включить фича-флаг `switch_to_config_v2` в проверенном и дополненном файле `config.yaml`:

```
feature_flags:
    ...
    switch_to_config_v2: true
```

Подробнее

Включение данного флага означает, что за хранение конфигурации и операции над ней теперь отвечает таблетка DS Controller, а не таблетка Console. Это переключает основной механизм управления конфигурацией кластера.

5. Разместить файл `config.yaml` на всех узлах кластера, заменив им предыдущий файл конфигурации.
6. Создать директорию для работы узла YDB с конфигурацией на каждом из узлов. В случае запуска нескольких узлов кластера на одном хосте создайте отдельные директории под каждый узел. Инициализируйте директорию, выполнив команду ydb admin node config init на каждом из узлов. В параметре `--from-config` укажите путь к файлу `config.yaml`, размещённому на узлах ранее.

```
sudo mkdir -p /opt/ydb/config-dir
sudo chown -R ydb:ydb /opt/ydb/config-dir
ydb admin node config init --config-dir /opt/ydb/config-dir --from-config /opt/ydb/cfg/config.yaml
```

Подробнее

В дальнейшем система самостоятельно будет сохранять актуальную конфигурацию в указанные директории.

7. Перезапустить все узлы кластера с помощью процедуры rolling-restart, добавив опцию `ydbd --config-dir` при запуске узла с указанием пути до директории, а также убрав опцию `ydbd --yaml-config`.

Вручную

При ручном запуске добавьте опцию `--config-dir` к команде `ydbd server`, не указывая опцию `--yaml-config`:

```
ydbd server --config-dir /opt/ydb/config-dir
```

С использованием systemd

При использовании systemd добавьте опцию `--config-dir` к команде `ydbd server` в конфигурационный файл systemd, а также удалите опцию `--yaml-config`:

```
ExecStart=/opt/ydb/bin/ydbd server --config-dir /opt/ydb/config-dir
```

После обновления файла systemd выполните следующую команду, чтобы применить изменения:

```
sudo systemctl daemon-reload
```

8. Загрузить полученный ранее конфигурационный файл `config.yaml` в систему с помощью команды ydb admin cluster config replace:

```
ydb -e grpc://<node.ydb.tech>:2135 admin cluster config replace -f config.yaml
```

Команда запросит подтверждение на выполнение операции `This command may damage your cluster, do you want to continue? [y/N]`, в ответ на этот запрос необходимо согласиться и ввести `y`.

Подробнее

После выполнения команды конфигурационный файл загрузится во внутреннее хранилище таблетки DS Controller и сохранится в директориях, указанных в опции `--config-dir` на каждом узле. С этого момента любое изменение конфигурации на существующих узлах выполняется с помощью специальных команд YDB CLI. Также при запуске узла актуальная конфигурация будет автоматически загружаться из конфигурационной директории.

9. Получить текущую конфигурацию кластера с помощью ydb admin cluster config fetch:

```
ydb -e grpc://<node.ydb.tech>:2135 admin cluster config fetch > config.yaml
```

Файл `config.yaml` должен совпадать с конфигурационными файлами, разложенными по узлам кластера, за исключением поля `metadata.version`, которое должно быть больше на единицу по сравнению с версией на узлах кластера.

10. Добавить в `config.yaml` в разделе `config` следующий блок:

```
self_management_config:
  enabled: true
```

Подробнее

Данная секция отвечает за включение механизма распределённой конфигурации в кластере. Хранение конфигурации и любые операции над ней будут осуществляться через данный механизм.

11. Загрузить обновлённый конфигурационный файл в кластер с помощью ydb admin cluster config replace:

```
ydb -e grpc://<node.ydb.tech>:2135 admin cluster config replace -f config.yaml
```

12. Перезапустить все узлы хранения кластера с помощью процедуры rolling restart.
13. При наличии секции `config.domains_config.security_config` в файле `config.yaml` вынести её на уровень выше — в секцию `config`.
14. Удалить из файла `config.yaml` секции `config.blob_storage_config` и `config.domains_config`.
15. Загрузить обновлённый конфигурационный файл в кластер:

    ```
    ydb -e grpc://<node.ydb.tech>:2135 admin cluster config replace -f config.yaml
    ```

    Подробнее

    > После загрузки конфигурации кластер YDB будет переведён в режим автоматического управления конфигурацией State Storage и статической группой с помощью механизма распределённой конфигурации.

Убедиться в успешном завершении миграции можно, проверив версию конфигурации на узлах кластера одним из способов, описанных в статье Проверка версии конфигурации. На всех узлах кластера версия `Configuration version` должна быть равна `v2`.

Результат

В результате проделанных действий кластер будет переведён на режим конфигурации V2. Управление единой конфигурацией осуществляется с помощью специальных команд YDB CLI, статическая группа и State Storage управляются системой автоматически.

Миграция на конфигурацию V1

Данный документ содержит инструкцию по миграции с конфигурации V2 на конфигурацию V1.

> **Примечание**
>
> Данная инструкция предназначена для аварийных ситуаций, когда после перехода на конфигурацию V2 возникли непредвиденные проблемы и требуется откат на конфигурацию V1, например, для последующего отката на версию YDB ниже v25-1. В штатном режиме работы эта процедура не требуется.

Исходное состояние

Миграция на конфигурацию V1 возможна только в том случае, если в кластере используется конфигурация V2. Это может быть достигнуто:

- в результате миграции на конфигурацию V2;
- при первоначальном развёртывании кластера.

Узнать текущую версию конфигурации на узлах можно несколькими способами, описанными в статье Проверка версии конфигурации. Перед началом миграции убедитесь, что кластер работает на конфигурации V2.

Инструкция по миграции на конфигурацию V1

Для того чтобы перевести кластер YDB на конфигурацию V1, необходимо проделать следующие шаги:

1. Получить текущую конфигурацию кластера с помощью команды ydb admin cluster config fetch:

   ```
   ydb -e grpc://<node.ydb.tech>:2135 admin cluster config fetch --v2-internal-state > config.yaml
   ```

   Подробнее

   Аргумент `--v2-internal-state` указывает, что будет получена полная конфигурация кластера, включая параметры настройки State Storage и статической группы.

2. Изменить конфигурационный файл `config.yaml`, поменяв значение параметра `self_management_config.enabled` с `true` на `false`:

   ```
   self_management_config:
     enabled: false
   ```

   Подробнее

   Данная секция отвечает за управление механизмом распределённой конфигурации. Установка значения `enabled: false` отключает этот механизм. Далее управление конфигурацией State Storage и статической группы будет осуществляться вручную через секции `domains_config` и `blob_storage_config` соответственно в конфигурационном файле (эти секции были получены на предыдущем шаге при использовании флага `--full`).

3. Загрузить обновлённый конфигурационный файл в кластер с помощью ydb admin cluster config replace:

   ```
   ydb -e grpc://<node.ydb.tech>:2135 admin cluster config replace -f config.yaml
   ```

4. Перезапустить все узлы кластера с помощью процедуры rolling-restart.

   Подробнее

   После перезапуска узлов кластер будет переведён в режим ручного управления State Storage и статической группой, но всё ещё будет использовать единый конфигурационный файл, доставляемый через таблетку BSController. Конфигурация узлов при запуске всё ещё будет читаться из директории, указанной в опции `ydbd --config-dir`, и там же сохраняться.

5. Получить текущую конфигурацию кластера с помощью `ydb admin cluster config fetch`:

   ```
   ydb -e grpc://<node.ydb.tech>:2135 admin cluster config fetch > config.yaml
   ```

   Подробнее

   В полученной конфигурации будут отсутствовать секции `domains_config` и `blob_storage_config`, так как они управляются вручную и не должны быть частью динамической конфигурации.

6. Разместить полученный файл `config.yaml` (это будет ваша статическая конфигурация V1) в файловую систему каждого узла кластера.
7. Перезапустить все узлы кластера с помощью процедуры rolling-restart, указав путь к статическому конфигурационному файлу через опцию `ydbd --yaml-config` и убрав опцию `ydbd --config-dir`:

   Вручную

   При ручном запуске добавьте опцию `--yaml-config` к команде `ydbd server`, не указывая опцию `--config-dir`:

   ```
   ydbd server --yaml-config /opt/ydb/cfg/config.yaml
   ```

С использованием systemd

При использовании systemd добавьте опцию `--yaml-config` к команде `ydbd server` в конфигурационный файл systemd, а также удалите опцию `--config-dir`:

```
ExecStart=/opt/ydb/bin/ydbd server --yaml-config /opt/ydb/config/config.yaml
```

После обновления файла systemd выполните следующую команду, чтобы применить изменения:

```
sudo systemctl daemon-reload
```

Убедиться в успешном завершении миграции можно, проверив версию конфигурации на узлах кластера одним из способов, описанных в статье Проверка версии конфигурации. На всех узлах кластера должна использоваться конфигурация `v1`.

Результат

В результате проделанных действий кластер будет переведён в режим конфигурации V1. Конфигурация состоит из двух частей: статической и динамической, управление статической группой и State Storage осуществляется вручную.

Проверка версии конфигурации

Существует два основных способа проверить, какую версию механизма конфигурации (V1 или V2) используют узлы вашего кластера YDB:

1. Через YDB UI
2. Через метрики кластера

Через YDB UI

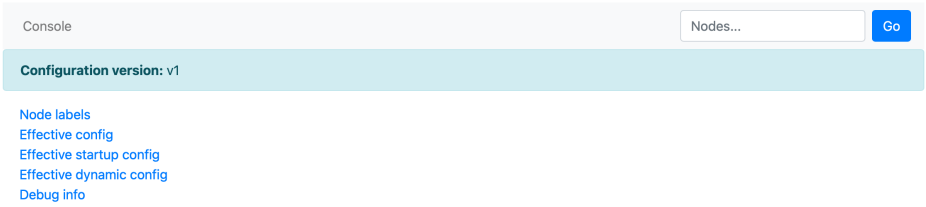
Данный способ может быть применен, если сбор метрик с кластера YDB в систему мониторинга не настроен. Вы можете проверить версию конфигурации для конкретного узла или переключаться между узлами через YDB UI:

1. Откройте в браузере страницу актора `configs_dispatcher` для любого узла кластера. URL имеет вид:

   ```
   http://<endpoint>:8765/actors/configs_dispatcher
   ```

   где `<endpoint>` - адрес произвольного узла кластера YDB.
2. В верхней части открывшейся страницы найдите поле `Configuration version`. В нем будет указана версия конфигурации (`v1` или `v2`), используемая данным узлом.

   Так выглядит страница узла, использующего конфигурацию V1:

3. Чтобы проверить другие узлы, используйте поле поиска `Nodes...` в правом верхнем углу страницы для переключения между узлами.

Через метрики кластера

Данный способ удобен при наличии большого числа узлов в кластере YDB. Если у вас настроен сбор метрик с кластера YDB в систему мониторинга, проделайте следующие действия:

1. Найдите дашборд, отображающий метрики кластера.
2. Перейдите к группе сенсоров `config` и подсистеме `configs_dispatcher`.
3. Обратите внимание на сенсоры `ConfigurationV1` и `ConfigurationV2`. Значения этих сенсоров показывают количество узлов кластера, работающих с конфигурацией V1 и V2 соответственно.

Например, если `ConfigurationV1 > 0`, значит в кластере есть узлы, которые используют конфигурацию v1. Если `ConfigurationV2 = 0` и `ConfigurationV2` равен общему числу узлов, значит, все узлы используют конфигурацию V2.

Сравнение конфигураций кластера YDB: V1 и V2

В YDB существует два основных подхода к управлению конфигурацией кластера: V1 и V2. Начиная с версии YDB 25.1, поддерживается конфигурация V2, которая унифицирует управление кластерами YDB, позволяет работать с конфигурацией полностью через YDB CLI, а также автоматизирует наиболее сложные аспекты конфигурации (управление статической группой и State Storage). Конфигурация V2 пока является экспериментальной и не рекомендуется для использования в продуктиве.

В этой статье описываются ключевые различия между этими двумя подходами.

Характеристика	Конфигурация V1	Конфигурация V2
Структура конфигурации	Раздельная: статическая и динамическая.	**Единая** конфигурация.
Управление файлами	Статическая: ручное размещение файла на каждом узле. Динамическая: централизованная загрузка через CLI.	Единая: централизованная загрузка через CLI, автоматическая доставка на все узлы.
Механизм доставки и применения	Статическая: читается и применяется из локального файла при запуске. Динамическая: через таблетку `Console`.	Полностью автоматически через механизм распределённой конфигурации. Технические подробности.
Управление State Storage и статической группой	**Ручное**: через обязательные секции `domains_config` и `blob_storage_config` в статической конфигурации.	**Автоматическое**: управляется системой распределённой конфигурации.
Рекомендуемое использование	Продакшн использование для всех версий YDB.	Экспериментальное использование на кластерах версии 25.1 и выше.

Конфигурация V1

Конфигурация V1 кластера YDB состоит из двух частей:

- **Статическая конфигурация:** управляет ключевыми параметрами узлов, включая конфигурацию State Storage и статической группы (секции `domains_config` и `blob_storage_config` соответственно). Требует ручного размещения одного и того же файла конфигурации на каждом узле кластера. Путь до конфигурации указывается при запуске узла через опцию `--yaml-config`.
- **Динамическая конфигурация:** управляет остальными параметрами кластера. Загружается централизованно с помощью команды `ydb admin config replace` и распространяется на узлы базы данных.

Использование конфигурации V1 является рекомендуемым для всех кластеров YDB.

Конфигурация V2

Начиная с версии YDB 25.1, поддерживается конфигурация V2. Ключевые особенности:

- **Единый конфигурационный файл:** вся конфигурация кластера хранится и управляется как единое целое.
- **Централизованное управление:** конфигурация загружается на кластер с помощью команды `ydb admin cluster config replace` и автоматически доставляется до всех узлов самим кластером YDB через механизм распределённой конфигурации.
- **Ранняя валидация:** проверка корректности происходит ещё до доставки конфигурационного файла на узлы кластера, а не при рестарте серверных процессов.
- **Автоматическое управление State Storage и статической группой:** V2 поддерживает автоматическую конфигурацию, что позволяет не указывать эти секции в конфигурационном файле вручную.
- **Хранение на узлах:** актуальная конфигурация автоматически сохраняется каждым узлом в специальной директории (указывается опцией `--config-dir` при запуске `ydbd`) и используется при последующих перезапусках.

Способы развёртывания YDB

В этом разделе документации YDB описаны способы развёртывания кластеров YDB с помощью различных инструментов, а также инструкции по дальнейшей работе с ними в зависимости от выбранного способа.

- **Ansible**: для развёртываний на физическом оборудовании и виртуальных машинах.
- **Kubernetes**: для развёртываний в контейнерах.
- **Вручную**: для ручного развёртывания.

Вне зависимости от выбранного способа управления инфраструктурой, перед началом работы рекомендуется ознакомиться с системными требованиями YDB.

Работа с YDB с помощью Ansible

Этот раздел документации YDB содержит набор статей по администрированию кластеров YDB при помощи Ansible. Это рекомендуемый подход к запуску produciton кластеров YDB непосредственно на виртуальных машинах и физических серверах. Для контейнерных окружений вместо Ansible рекомендуется использовать Kubernetes.

Ключевые статьи для начала работы с этим разделом:

- Развёртывание YDB кластера с помощью Ansible
- Развёртывание инфраструктуры для кластера YDB с помощью Terraform
- Перезапуск кластеров YDB, развёрнутых с помощью Ansible
- Наблюдаемость:
  - Логирование в кластерах, развёрнутых с помощью Ansible

Развёртывание YDB кластера с помощью Ansible

В этом руководстве описан процесс развёртывания кластера YDB на группе серверов с использованием Ansible. Такой способ рекомендуется применять в средах с физическими серверами или виртуальными машинами.

Основные материалы:

- Подготовка к развертыванию
- Развёртывание кластера с использованием конфигурации V1
- Развёртывание кластера с использованием конфигурации V2

Подготовка к развертыванию

Предварительные требования

Выберите топологию для установки

Перед установкой выберите подходящую топологию кластера YDB — от неё зависит количество необходимых серверов и дисков:

Для начала работы с YDB, рекомендуется выбрать топологию `mirror-3dc-3-nodes`. Для этого потребуется всего 3 сервера и 9 дисков для пользовательских данных — это самый простой и быстрый способ создать пилотный кластер.

Для масштабных развёртываний выберите одну из следующих опций:

- `mirror-3-dc` — для кластеров, размещённых в нескольких дата-центрах. Потребуется не менее 9 серверов и 9 дисков для пользовательских данных.
- `block-4-2` — для размещения кластера в одном дата-центре. Потребуется не менее 8 серверов и 8 дисков для пользовательских данных.

В каждом сервере должен быть хотя бы один отдельный диск для пользовательских данных. Рекомендуется также добавить отдельный небольшой диск под операционную систему. Подробнее о различных вариантах топологии и избыточности читайте в этой статье.

В дальнейшем можно при необходимости расширять кластер без остановки его работы и прерывания доступа пользователей к данным.

Настройка серверов

Примечание

Рекомендуемые требования к серверам:

- 16 CPU (рассчитывается исходя из утилизации 8 CPU узлом хранения и 8 CPU динамическим узлом).
- 16 GB RAM (рекомендуемый минимальный объем RAM).
- Дополнительные SSD-диски для данных, не менее 120 GB каждый.
- Доступ по SSH.
- Сетевая связность между машинами в кластере.
- ОС: Ubuntu 18+, Debian 9+.
- Доступ в интернет для обновления репозиториев и скачивания необходимых пакетов.

Подробнее см. Системные требования и рекомендации для YDB.

При использовании виртуальных машин у публичного облачного провайдера рекомендуется следовать Развёртывание инфраструктуры для кластера YDB с помощью Terraform.

Настройка программного обеспечения

Для работы с проектом на локальной (промежуточной или установочной) машине потребуется:

- Python 3 версии 3.10+
- Поддерживаются версии Ansible core начиная с 2.11 и до 2.18
- Рабочая директория на сервере с SSH-доступом ко всем серверам кластера

Совет

Рекомендуется хранить рабочую директорию и все файлы, созданные в ходе этого руководства, в репозитории системы управления версиями, например Git. Если несколько DevOps-инженеров будут работать с развёртываемым кластером, они должны взаимодействовать в общем репозитории.

Если Ansible уже установлен, можно перейти к разделу «Настройка проекта Ansible». Если Ansible ещё не установлен, установите его одним из следующих способов:

Установка Ansible глобально

На примере Ubuntu 22.04 LTS:

- Обновите список пакетов apt командой `sudo apt-get update`.
- Обновите пакеты командой `sudo apt-get upgrade`.
- Установите пакет `software-properties-common` для управления источниками программного обеспечения вашего дистрибутива — `sudo apt install software-properties-common`.
- Добавьте новый PPA в apt — `sudo add-apt-repository --yes --update ppa:ansible/ansible`.
- Установите Ansible — `sudo apt-get install ansible-core` (убедитесь, что устанавливаемая версия не выше 2.18 и не меньше 2.11; установка пакета просто `ansible` приведёт к устаревшей и неподходящей версии).
- Проверьте версию Ansible core — `ansible --version`.

Установка Ansible в виртуальное окружение Python

На примере Ubuntu 22.04 LTS:

- Обновите список доступных deb пакетов — `sudo apt-get update`.
- Установите пакет `python3-venv` для управления Python виртуальными окружениями — `sudo apt-get install venv`.
- Создайте директорию, где будет создано виртуальное окружение. Например, `mkdir venv-ansible`.
- Создайте виртуальное окружение Python — `python3 -m venv venv-ansible`, где `venv-ansible` - путь к директории созданной на предыдущем шаге.

- Активируйте виртуальное окружение — `source venv-ansible/bin/activate`. Все дальнейшие действия с Ansible выполняются внутри виртуального окружения. Выйти из него можно командой `deactivate`.
- Установите рекомендуемую версию Ansible с помощью команды `pip3 install "ansible-core>=2.11,<2.19"` (убедитесь, что устанавливаемая версия не выше 2.18 и не меньше 2.11). Проверьте установленную версию Ansible — `ansible --version`.

Для получения дополнительной информации и других вариантов установки ознакомьтесь с руководством по установке Ansible.

Настройка проекта Ansible

Установка YDB Ansible плейбуков

Через requirements.yaml

```
cat <<EOF > requirements.yaml
roles: []
collections:
  - name: git+https://github.com/ydb-platform/ydb-ansible
    type: git
    version: latest
EOF
ansible-galaxy install -r requirements.yaml
```

Однократно

```
ansible-galaxy collection install git+https://github.com/ydb-platform/ydb-ansible.git,latest
```

После выполнения подготовительных мероприятий можно переходить к развёртыванию системы. Выберите инструкцию в соответствии с вашей конфигурацией:

- Развёртывание кластера с использованием конфигурации V1
- Развёртывание кластера с использованием конфигурации V2

Развёртывание кластера с использованием конфигурации V1

Подготовьте окружение

Перед развёртыванием системы обязательно выполните подготовительные действия. Ознакомьтесь с документом Подготовка к развертыванию.

Создайте директорию для работы

```
mkdir deployment
cd deployment
mkdir inventory
mkdir files
```

Создайте конфигурационный файл Ansible

Создайте `ansible.cfg` с конфигурацией Ansible, подходящей для целевой среды развёртывания. Подробности в справочнике по конфигурации Ansible. Дальнейшее руководство предполагает, что поддиректория `./inventory` рабочей директории настроена для использования файлов инвентаризации.

Пример стартового ansible.cfg

> **Примечание**
>
> Использование параметра `StrictHostKeyChecking=no` в `ssh_args` повышает удобство автоматизации, но снижает уровень безопасности SSH-соединения (отключает проверку подлинности хоста). Для production-окружений рекомендуется не указывать этот аргумент и настроить доверенные ключи вручную. Используйте этот параметр только для тестовых и временных установок.

```
[defaults]
conditional_bare_variables = False
force_handlers = True
gathering = explicit
interpreter_python = /usr/bin/python3
inventory = ./inventory
pipelining = True
private_role_vars = True
timeout = 5
verbosity = 1
log_path = ./ydb.log
vault_password_file = ./ansible_vault_password_file

[ssh_connection]
retries = 5
timeout = 60
ssh_args = -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no -o ControlMaster=auto -o ControlPe
rsist=60s -o ControlPath=/tmp/ssh-%h-%p-%r -o ServerAliveCountMax=3 -o ServerAliveInterval=10
```

Создайте основной файл инвентаризации

Создайте файл `inventory/50-inventory.yaml` и заполните его в зависимости от выбранной топологии (подробнее: выбор топологии). Примеры для каждой поддерживаемой топологии приведены ниже во вкладках — выберите подходящий.

mirror-3-dc-3nodes

```
all:
  children:
    ydb:
      #Серверы
      hosts:
        static-node-1.ydb-cluster.com:
        static-node-2.ydb-cluster.com:
        static-node-3.ydb-cluster.com:

      vars:
        # Ansible
        ansible_user: имя_пользователя
        ansible_ssh_private_key_file: "/путь/к/вашему/id_rsa"

        # Система
        system_timezone: UTC
        system_ntp_servers: [time.cloudflare.com, time.google.com, ntp.ripe.net, pool.ntp.org]

        # Узлы
        ydb_config: "{{ ansible_config_file | dirname }}/files/config.yaml"
        ydb_version: "версия_системы"

        # Хранилище
        ydb_cores_static: 8
        ydb_disks:
          - name: /dev/vdb
            label: ydb_disk_1
          - name: /dev/vdc
            label: ydb_disk_2
          - name: /dev/vdd
```

```
            label: ydb_disk_3
        ydb_allow_format_drives: true
        ydb_skip_data_loss_confirmation_prompt: false
        ydb_pool_kind: ssd
        ydb_database_groups: 8
        ydb_cores_dynamic: 8
        ydb_dynnodes:
          - { instance: 'a', offset: 1 }
          - { instance: 'b', offset: 2 }
        ydb_brokers:
          - static-node-1.ydb-cluster.com
          - static-node-2.ydb-cluster.com
          - static-node-3.ydb-cluster.com

        # База данных
        ydb_user: root
        ydb_domain: Root
        ydb_dbname: database

        #Настройки авторизации
        ydb_enforce_user_token_requirement: true
        ydb_request_client_certificate: true
```

mirror-3-dc-9-nodes

```
all:
  children:
    ydb:
      #Серверы
      hosts:
        static-node-1.ydb-cluster.com:
        static-node-2.ydb-cluster.com:
        static-node-3.ydb-cluster.com:
        static-node-4.ydb-cluster.com:
        static-node-5.ydb-cluster.com:
        static-node-6.ydb-cluster.com:
        static-node-7.ydb-cluster.com:
        static-node-8.ydb-cluster.com:
        static-node-9.ydb-cluster.com:

      vars:
        # Ansible
        ansible_user: имя_пользователя
        ansible_ssh_private_key_file: "/путь/к/вашему/id_rsa"

        # Система
        system_timezone: UTC
        system_ntp_servers: [time.cloudflare.com, time.google.com, ntp.ripe.net, pool.ntp.org]

        # Узлы
        ydb_config: "{{ ansible_config_file | dirname }}/files/config.yaml"
        ydb_version: "версия_системы"

        # Хранилище
        ydb_cores_static: 8
        ydb_disks:
          - name: /dev/vdb
            label: ydb_disk_1
        ydb_allow_format_drives: true
        ydb_skip_data_loss_confirmation_prompt: false
        ydb_pool_kind: ssd
        ydb_database_groups: 8
        ydb_cores_dynamic: 8
        ydb_dynnodes:
          - { instance: 'a', offset: 1 }
          - { instance: 'b', offset: 2 }
        ydb_brokers:
          - static-node-1.ydb-cluster.com
          - static-node-2.ydb-cluster.com
          - static-node-3.ydb-cluster.com

        # База данных
        ydb_user: root
        ydb_domain: Root
        ydb_dbname: database

        #Настройки авторизации
        ydb_enforce_user_token_requirement: true
        ydb_request_client_certificate: true
```

block-4-2

```
all:
  children:
    ydb:
      #Серверы
      hosts:
        static-node-1.ydb-cluster.com:
        static-node-2.ydb-cluster.com:
        static-node-3.ydb-cluster.com:
```

```
        static-node-4.ydb-cluster.com:
        static-node-5.ydb-cluster.com:
        static-node-6.ydb-cluster.com:
        static-node-7.ydb-cluster.com:
        static-node-8.ydb-cluster.com:

    vars:
      # Ansible
      ansible_user: имя_пользователя
      ansible_ssh_private_key_file: "/путь/к/вашему/id_rsa"

      # Система
      system_timezone: UTC
      system_ntp_servers: [time.cloudflare.com, time.google.com, ntp.ripe.net, pool.ntp.org]

      # Узлы
      ydb_config: "{{ ansible_config_file | dirname }}/files/config.yaml"
      ydb_version: "версия_системы"

      # Хранилище
      ydb_cores_static: 8
      ydb_disks:
        - name: /dev/vdb
          label: ydb_disk_1
      ydb_allow_format_drives: true
      ydb_skip_data_loss_confirmation_prompt: false
      ydb_pool_kind: ssd
      ydb_database_groups: 7
      ydb_cores_dynamic: 8
      ydb_dynnodes:
        - { instance: 'a', offset: 1 }
        - { instance: 'b', offset: 2 }
      ydb_brokers:
        - static-node-1.ydb-cluster.com
        - static-node-2.ydb-cluster.com
        - static-node-3.ydb-cluster.com

      # База данных
      ydb_user: root
      ydb_domain: Root
      ydb_dbname: database

      #Настройки авторизации
      ydb_enforce_user_token_requirement: true
      ydb_request_client_certificate: true
```

Обязательные настройки, которые нужно адаптировать под окружение в выбранном шаблоне:

1. **Имена серверов.** Замените `static-node-*.ydb-cluster.com` в `all.children.ydb.hosts` и `all.children.vars.ydb_brokers` на реальные FQDN.
2. **Настройка SSH доступа.** Укажите пользователя `ansible_user` и путь к приватному ключу `ansible_ssh_private_key_file`, которые Ansible будет использовать для подключения к вашим серверам.
3. **Пути к блочным устройствам в файловой системе.** В секции `all.children.ydb.vars.ydb_disks` шаблон предполагает, что `/dev/vda` предназначен для операционной системы, а следующие диски, такие как `/dev/vdb`, — для слоя хранения YDB. Метки дисков создаются плейбуками автоматически, и их имена могут быть произвольными.
4. **Версия системы.** В параметре `ydb_version` укажите номер версии YDB, которую нужно установить. Список доступных версий вы найдёте на странице загрузок.

Рекомендуемые настройки для адаптации:

- `ydb_domain`. Это будет первый компонент пути для всех объектов схемы в кластере. Например, можно поместить туда название своей компании, регион кластера и т.д.
- `ydb_dbname`. Это будет второй компонент пути для всех объектов схемы в базе данных. Например, можно поместить туда название сценария использования или проекта.

Дополнительные настройки

Существует несколько вариантов указания того, какие именно исполняемые файлы YDB вы хотите использовать для кластера:

- `ydb_version`: автоматически загрузить один из официальных релизов YDB по номеру версии. Например, `23.4.11`.
- `ydb_archive`: локальный путь файловой системы к архиву дистрибутива YDB, загруженному или иным образом подготовленному заранее.

Для использования федеративных запросов может потребоваться установка коннектора. Плейбук может развернуть fq-connector-go на хостах с динамическими узлами. Используйте следующие настройки:

- `ydb_install_fq_connector` — установите в `true` для установки коннектора.
- Выберите один из доступных вариантов развёртывания исполняемых файлов fq-connector-go:
  - `ydb_fq_connector_version`: автоматически загрузить один из официальных релизов fq-connector-go по номеру версии. Например, `v0.7.1`.
  - `ydb_fq_connector_git_version`: автоматически скомпилировать исполняемый файл fq-connector-go из исходного кода, загруженного из официального репозитория GitHub. Значение настройки — это имя ветки, тега или коммита. Например, `main`.
  - `ydb_fq_connector_archive`: локальный путь файловой системы к архиву дистрибутива fq-connector-go, загруженному или иным образом подготовленному заранее.
  - `ydb_fq_connector_binary`: локальные пути файловой системы к исполняемому файлу fq-connector-go, загруженному или иным образом подготовленному заранее.

- `ydb_tls_dir` — укажите локальный путь к папке с TLS-сертификатами, подготовленными заранее. Она должна содержать файл `ca.crt` и подкаталоги с именами, соответствующими именам хостов узлов, содержащие сертификаты для данного узла. Если не указано, самоподписанные TLS-сертификаты будут сгенерированы автоматически для всего кластера YDB.
- `ydb_brokers` — перечислите FQDN узлов брокеров. Например:

```
ydb_brokers:
    - static-node-1.ydb-cluster.com
    - static-node-2.ydb-cluster.com
    - static-node-3.ydb-cluster.com
```

Оптимальное значение настройки `ydb_database_groups` в разделе `vars` зависит от доступных дисков. Предполагая только одну базу данных в кластере, используйте следующую логику:

- Для промышленных развёртываний используйте диски ёмкостью более 800 ГБ с высокой производительностью IOPS, затем выберите значение для этой настройки на основе топологии кластера:
  - Для `block-4-2` установите `ydb_database_groups` на 95% от общего количества дисков, округляя вниз.
  - Для `mirror-3-dc` установите `ydb_database_groups` на 84% от общего количества дисков, округляя вниз.
- Для тестирования YDB на небольших дисках установите `ydb_database_groups` в 1 независимо от топологии кластера.

Значения переменных `system_timezone` и `system_ntp_servers` зависят от свойств инфраструктуры, на которой развёртывается кластер YDB. По умолчанию `system_ntp_servers` включает набор NTP-серверов без учёта географического расположения инфраструктуры, на которой будет развёрнут кластер YDB. Мы настоятельно рекомендуем использовать локальный NTP-сервер для on-premise инфраструктуры и следующие NTP-серверы для облачных провайдеров:

AWS

- `system_timezone`: USA/<region_name>
- `system_ntp_servers`: [169.254.169.123, time.aws.com] Подробнее о настройках NTP-серверов AWS.

Azure

- О том, как настраивается синхронизация времени на виртуальных машинах Azure, можно прочитать в этой статье.

Alibaba

- Специфика подключения к NTP-серверам в Alibaba описана в этой статье.

Yandex Cloud

- `system_timezone`: Europe/Moscow
- `system_ntp_servers`: [0.ru.pool.ntp.org, 1.ru.pool.ntp.org, ntp0.NL.net, ntp2.vniiftri.ru, ntp.ix.ru, ntps1-1.cs.tu-berlin.de] Подробнее о настройках NTP-серверов Yandex Cloud.

Измените пароль пользователя root

Создайте файл `ansible_vault_password_file` с содержимым:

```
password
```

Этот файл содержит пароль, который Ansible будет использовать для автоматического шифрования и расшифровки конфиденциальных данных, например, файлов с паролями пользователей. Благодаря этому пароли не хранятся в открытом виде в репозитории. Подробнее о работе механизма Ansible Vault можно прочитать в официальной документации.

Далее необходимо установить пароль для начального пользователя, указанного в настройке `ydb_user` (по умолчанию `root`). Этот пользователь изначально будет иметь полные права доступа в кластере, но при необходимости это можно изменить позже. Создайте `inventory/99-inventory-vault.yaml` со следующим содержимым (замените `<password>` на фактический пароль):

```
all:
  children:
    ydb:
      vars:
        ydb_password: <password>
```

Зашифруйте этот файл с помощью команды `ansible-vault encrypt inventory/99-inventory-vault.yaml`.

Подготовьте конфигурационный файл YDB

Примеры ниже содержат секцию `feature_flags`. Перед изменением её значений ознакомьтесь с назначением фича-флагов и правилами их переключения.

Создайте файл `files/config.yaml` и заполните его в зависимости от выбранной топологии (подробнее: выбор топологии). Примеры для каждой поддерживаемой топологии приведены ниже во вкладках — выберите подходящий.

mirror-3-dc-3nodes

```
storage_config_generation: 0
static_erasure: mirror-3-dc
host_configs:
- drive:
  - path: /dev/disk/by-partlabel/ydb_disk_1
    type: SSD
```

```
    - path: /dev/disk/by-partlabel/ydb_disk_2
      type: SSD
    - path: /dev/disk/by-partlabel/ydb_disk_3
      type: SSD
  host_config_id: 1
hosts:
- host: static-node-1.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 1
    data_center: 'zone-a'
    rack: '1'
- host: static-node-2.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 2
    data_center: 'zone-b'
    rack: '2'
- host: static-node-3.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 3
    data_center: 'zone-d'
    rack: '3'
domains_config:
  domain:
  - name: Root
    storage_pool_types:
    - kind: ssd
      pool_config:
        box_id: 1
        erasure_species: mirror-3-dc
        kind: ssd
        geometry:
          realm_level_begin: 10
          realm_level_end: 20
          domain_level_begin: 10
          domain_level_end: 256
        pdisk_filter:
        - property:
          - type: SSD
        vdisk_kind: Default
  state_storage:
  - ring:
      node: [1, 2, 3]
      nto_select: 3
    ssid: 1
  security_config:
    enforce_user_token_requirement: true
    monitoring_allowed_sids:
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
    administration_allowed_sids:
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
    viewer_allowed_sids:
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
    register_dynamic_node_allowed_sids:
    - databaseNodes@cert
    - root@builtin
blob_storage_config:
  service_set:
    groups:
    - erasure_species: mirror-3-dc
      rings:
      - fail_domains:
        - vdisk_locations:
          - node_id: static-node-1.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_1
        - vdisk_locations:
          - node_id: static-node-1.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_2
        - vdisk_locations:
          - node_id: static-node-1.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_3
      - fail_domains:
        - vdisk_locations:
          - node_id: static-node-2.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_1
        - vdisk_locations:
          - node_id: static-node-2.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_2
```

```
        - vdisk_locations:
          - node_id: static-node-2.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_3
      - fail_domains:
        - vdisk_locations:
          - node_id: static-node-3.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_1
        - vdisk_locations:
          - node_id: static-node-3.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_2
        - vdisk_locations:
          - node_id: static-node-3.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_3
channel_profile_config:
  profile:
  - channel:
    - erasure_species: mirror-3-dc
      pdisk_category: 1   # 0=ROT, 1=SSD, 2=NVME
      storage_pool_kind: ssd
    - erasure_species: mirror-3-dc
      pdisk_category: 1
      storage_pool_kind: ssd
    - erasure_species: mirror-3-dc
      pdisk_category: 1
      storage_pool_kind: ssd
    profile_id: 0
interconnect_config:
    start_tcp: true
    encryption_mode: OPTIONAL
    path_to_certificate_file: "/opt/ydb/certs/node.crt"
    path_to_private_key_file: "/opt/ydb/certs/node.key"
    path_to_ca_file: "/opt/ydb/certs/ca.crt"
grpc_config:
    cert: "/opt/ydb/certs/node.crt"
    key: "/opt/ydb/certs/node.key"
    ca: "/opt/ydb/certs/ca.crt"
    services_enabled:
    - legacy
    - discovery
auth_config:
  path_to_root_ca: /opt/ydb/certs/ca.crt
client_certificate_authorization:
  request_client_certificate: true
  client_certificate_definitions:
      - member_groups: ["databaseNodes@cert"]
        subject_terms:
        - short_name: "O"
          values: ["YDB"]
query_service_config:
  generic:
    connector:
      endpoint:
        host: localhost
        port: 19102
      use_ssl: false
    default_settings:
      - name: DateTimeFormat
        value: string
      - name: UsePredicatePushdown
        value: "true"
feature_flags:
  enable_external_data_sources: true
  enable_script_execution_operations: true
```

mirror-3-dc-9-nodes

```
storage_config_generation: 0
static_erasure: mirror-3-dc
host_configs:
- drive:
  - path: /dev/disk/by-partlabel/ydb_disk_1
    type: SSD
  host_config_id: 1
hosts:
- host: static-node-1.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 1
    data_center: 'zone-a'
    rack: '1'
- host: static-node-2.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 2
    data_center: 'zone-a'
    rack: '2'
- host: static-node-3.ydb-cluster.com
```

```
  host_config_id: 1
  walle_location:
    body: 3
    data_center: 'zone-a'
    rack: '3'
- host: static-node-4.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 4
    data_center: 'zone-b'
    rack: '4'
- host: static-node-5.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 5
    data_center: 'zone-b'
    rack: '5'
- host: static-node-6.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 6
    data_center: 'zone-b'
    rack: '6'
- host: static-node-7.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 7
    data_center: 'zone-d'
    rack: '7'
- host: static-node-8.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 8
    data_center: 'zone-d'
    rack: '8'
- host: static-node-9.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 9
    data_center: 'zone-d'
    rack: '9'
domains_config:
  domain:
  - name: Root
    storage_pool_types:
    - kind: ssd
      pool_config:
        box_id: 1
        erasure_species: mirror-3-dc
        kind: ssd
        pdisk_filter:
        - property:
          - type: SSD
        vdisk_kind: Default
  state_storage:
  - ring:
      node: [1, 2, 3, 4, 5, 6, 7, 8, 9]
      nto_select: 9
    ssid: 1
  security_config:
    enforce_user_token_requirement: true
    monitoring_allowed_sids:
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
    administration_allowed_sids:
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
    viewer_allowed_sids:
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
    register_dynamic_node_allowed_sids:
    - databaseNodes@cert
    - root@builtin
blob_storage_config:
  service_set:
    groups:
    - erasure_species: mirror-3-dc
      rings:
      - fail_domains:
        - vdisk_locations:
          - node_id: static-node-1.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_1
        - vdisk_locations:
          - node_id: static-node-2.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_1
        - vdisk_locations:
```

```
          - node_id: static-node-3.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_1
      - fail_domains:
        - vdisk_locations:
          - node_id: static-node-4.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_1
        - vdisk_locations:
          - node_id: static-node-5.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_1
        - vdisk_locations:
          - node_id: static-node-6.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_1
      - fail_domains:
        - vdisk_locations:
          - node_id: static-node-7.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_1
        - vdisk_locations:
          - node_id: static-node-8.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_1
        - vdisk_locations:
          - node_id: static-node-9.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_1
channel_profile_config:
  profile:
  - channel:
    - erasure_species: mirror-3-dc
      pdisk_category: 1   # 0=ROT, 1=SSD, 2=NVME
      storage_pool_kind: ssd
    - erasure_species: mirror-3-dc
      pdisk_category: 1
      storage_pool_kind: ssd
    - erasure_species: mirror-3-dc
      pdisk_category: 1
      storage_pool_kind: ssd
    profile_id: 0
interconnect_config:
    start_tcp: true
    encryption_mode: OPTIONAL
    path_to_certificate_file: "/opt/ydb/certs/node.crt"
    path_to_private_key_file: "/opt/ydb/certs/node.key"
    path_to_ca_file: "/opt/ydb/certs/ca.crt"
grpc_config:
    cert: "/opt/ydb/certs/node.crt"
    key: "/opt/ydb/certs/node.key"
    ca: "/opt/ydb/certs/ca.crt"
    services_enabled:
    - legacy
auth_config:
  path_to_root_ca: /opt/ydb/certs/ca.crt
client_certificate_authorization:
  request_client_certificate: true
  client_certificate_definitions:
      - member_groups: ["databaseNodes@cert"]
        subject_terms:
        - short_name: "O"
          values: ["YDB"]
query_service_config:
  generic:
    connector:
      endpoint:
        host: localhost
        port: 19102
      use_ssl: false
    default_settings:
      - name: DateTimeFormat
        value: string
      - name: UsePredicatePushdown
        value: "true"
feature_flags:
  enable_external_data_sources: true
  enable_script_execution_operations: true
```

block-4-2

```
storage_config_generation: 0
static_erasure: block-4-2
host_configs:
- drive:
  - path: /dev/disk/by-partlabel/ydb_disk_1
    type: SSD
  host_config_id: 1
hosts:
- host: static-node-1.ydb-cluster.com
  host_config_id: 1
```

```
  walle_location:
    body: 1
    data_center: 'zone-a'
    rack: '1'
- host: static-node-2.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 2
    data_center: 'zone-a'
    rack: '2'
- host: static-node-3.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 3
    data_center: 'zone-a'
    rack: '3'
- host: static-node-4.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 4
    data_center: 'zone-a'
    rack: '4'
- host: static-node-5.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 5
    data_center: 'zone-a'
    rack: '5'
- host: static-node-6.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 6
    data_center: 'zone-a'
    rack: '6'
- host: static-node-7.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 7
    data_center: 'zone-a'
    rack: '7'
- host: static-node-8.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 8
    data_center: 'zone-a'
    rack: '8'
domains_config:
  domain:
  - name: Root
    storage_pool_types:
    - kind: ssd
      pool_config:
        box_id: 1
        erasure_species: block-4-2
        kind: ssd
        pdisk_filter:
        - property:
          - type: SSD
        vdisk_kind: Default
  state_storage:
  - ring:
      node:
        - 1
        - 2
        - 3
        - 4
        - 5
        - 6
        - 7
        - 8
      nto_select: 8
    ssid: 1
  security_config:
    enforce_user_token_requirement: true
    monitoring_allowed_sids:
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
    administration_allowed_sids:
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
    viewer_allowed_sids:
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
    register_dynamic_node_allowed_sids:
    - databaseNodes@cert
    - root@builtin
blob_storage_config:
  service_set:
```

```
    groups:
    - erasure_species: block-4-2
      rings:
      - fail_domains:
        - vdisk_locations:
          - node_id: static-node-1.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_1
        - vdisk_locations:
          - node_id: static-node-2.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_1
        - vdisk_locations:
          - node_id: static-node-3.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_1
        - vdisk_locations:
          - node_id: static-node-4.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_1
        - vdisk_locations:
          - node_id: static-node-5.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_1
        - vdisk_locations:
          - node_id: static-node-6.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_1
        - vdisk_locations:
          - node_id: static-node-7.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_1
        - vdisk_locations:
          - node_id: static-node-8.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_1
channel_profile_config:
  profile:
  - channel:
    - erasure_species: block-4-2
      pdisk_category: 1   # 0=ROT, 1=SSD, 2=NVME
      storage_pool_kind: ssd
    - erasure_species: block-4-2
      pdisk_category: 1
      storage_pool_kind: ssd
    - erasure_species: block-4-2
      pdisk_category: 1
      storage_pool_kind: ssd
    profile_id: 0
interconnect_config:
    start_tcp: true
    encryption_mode: OPTIONAL
    path_to_certificate_file: "/opt/ydb/certs/node.crt"
    path_to_private_key_file: "/opt/ydb/certs/node.key"
    path_to_ca_file: "/opt/ydb/certs/ca.crt"
grpc_config:
    cert: "/opt/ydb/certs/node.crt"
    key: "/opt/ydb/certs/node.key"
    ca: "/opt/ydb/certs/ca.crt"
    services_enabled:
    - legacy
    - discovery
auth_config:
  path_to_root_ca: /opt/ydb/certs/ca.crt
client_certificate_authorization:
  request_client_certificate: true
  client_certificate_definitions:
      - member_groups: ["databaseNodes@cert"]
        subject_terms:
        - short_name: "O"
          values: ["YDB"]
query_service_config:
  generic:
    connector:
      endpoint:
        host: localhost
        port: 19102
      use_ssl: false
    default_settings:
      - name: DateTimeFormat
        value: string
      - name: UsePredicatePushdown
        value: "true"
feature_flags:
  enable_external_data_sources: true
  enable_script_execution_operations: true
```

Для ускорения и упрощения первичного развёртывания YDB конфигурационный файл уже содержит большинство настроек для установки кластера. Достаточно заменить стандартные хосты FQDN на актуальные в разделах `hosts` и `blob_storage_config`.

- Раздел `hosts`:

```
...
hosts:
  - host: static-node-1.ydb-cluster.com #FQDN BM
    host_config_id: 1
    walle_location:
      body: 1
      data_center: 'zone-a'
      rack: '1'
...
```

- Раздел `blob_storage_config`:

```
...
- fail_domains:
  - vdisk_locations:
    - node_id: static-node-1.ydb-cluster.com #FQDN BM
      pdisk_category: SSD
      path: /dev/disk/by-partlabel/ydb_disk_1
...
```

Остальные секции и настройки конфигурационного файла остаются без изменений.

Разверните кластер YDB

После завершения всех описанных выше подготовительных действий фактическое первоначальное развёртывание кластера сводится к выполнению следующей команды из рабочей директории:

```
ansible-playbook ydb_platform.ydb.initial_setup
```

Вскоре после начала будет необходимо подтвердить полную очистку настроенных дисков. Затем завершение развёртывания может занять десятки минут в зависимости от окружения и настроек. Этот плейбук выполняет примерно те же шаги, которые описаны в инструкциях для ручного развёртывания кластера YDB.

Проверьте состояние кластера

На последнем этапе плейбук выполнит несколько тестовых запросов с использованием настоящих временных таблиц для проверки корректной работы. При успешном выполнении для каждого сервера будут отображены статус ok, failed=0 и результаты тестовых запросов (3 и 6) при достаточно подробном выводе плейбука.

Пример вывода

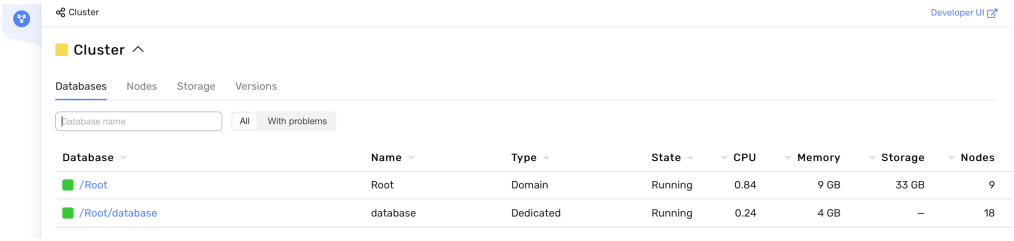
```
...

TASK [ydb_platform.ydb.ydbd_dynamic : run test queries] ************************************************
*********************************************************************************************
ok: [static-node-1.ydb-cluster.com] => (item={'instance': 'a'}) => {"ansible_loop_var": "item", "change
d": false, "item": {"instance": "a"}, "msg": "all test queries were successful, details: {\"count\":3,\"s
um\":6}\n"}
ok: [static-node-1.ydb-cluster.com] => (item={'instance': 'b'}) => {"ansible_loop_var": "item", "change
d": false, "item": {"instance": "b"}, "msg": "all test queries were successful, details: {\"count\":3,\"s
um\":6}\n"}
ok: [static-node-2.ydb-cluster.com] => (item={'instance': 'a'}) => {"ansible_loop_var": "item", "change
d": false, "item": {"instance": "a"}, "msg": "all test queries were successful, details: {\"count\":3,\"s
um\":6}\n"}
ok: [static-node-2.ydb-cluster.com] => (item={'instance': 'b'}) => {"ansible_loop_var": "item", "change
d": false, "item": {"instance": "b"}, "msg": "all test queries were successful, details: {\"count\":3,\"s
um\":6}\n"}
ok: [static-node-3.ydb-cluster.com] => (item={'instance': 'a'}) => {"ansible_loop_var": "item", "change
d": false, "item": {"instance": "a"}, "msg": "all test queries were successful, details: {\"count\":3,\"s
um\":6}\n"}
ok: [static-node-3.ydb-cluster.com] => (item={'instance': 'b'}) => {"ansible_loop_var": "item", "change
d": false, "item": {"instance": "b"}, "msg": "all test queries were successful, details: {\"count\":3,\"s
um\":6}\n"}
PLAY RECAP *********************************************************************************************
*********************************************************************************************
static-node-1.ydb-cluster.com : ok=167  changed=80   unreachable=0    failed=0    skipped=167  rescued=0
ignored=0
static-node-2.ydb-cluster.com : ok=136  changed=69   unreachable=0    failed=0    skipped=113  rescued=0
ignored=0
static-node-3.ydb-cluster.com : ok=136  changed=69   unreachable=0    failed=0    skipped=113  rescued=0
ignored=0
```

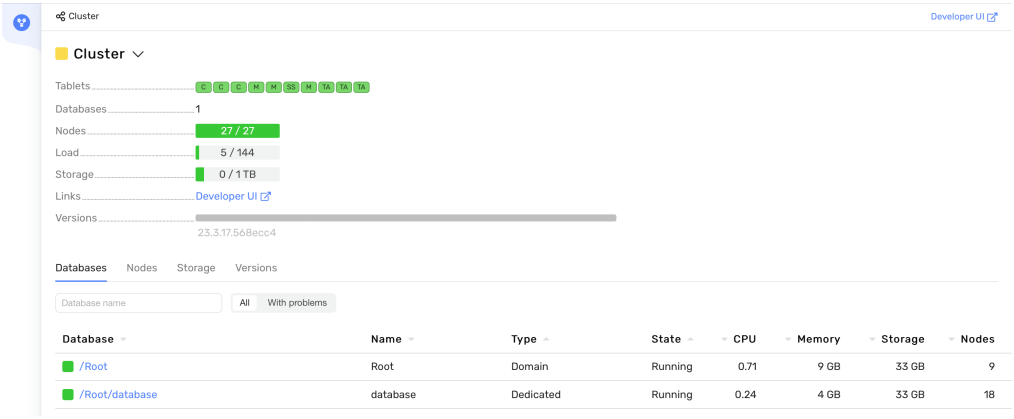
В результате выполнения плейбука `ydb_platform.ydb.initial_setup` будет создан кластер YDB. Он будет содержать домен с именем из настройки `ydb_domain` (по умолчанию `Root`), базу данных с именем из настройки `ydb_dbname` (по умолчанию `database`) и начального пользователя с именем из настройки `ydb_user` (по умолчанию `root`).

Дополнительные шаги

Самый простой способ исследовать только что развёрнутый кластер — использовать YDB UI, работающий на порту 8765 каждого сервера. Если нет прямого доступа к порту из браузера, можно настроить SSH-туннелирование. Для этого выполните команду `ssh -L 8765:localhost:8765 -i <private-key> <user>@<any-ydb-server-hostname>` на локальной машине (при необходимости добавьте дополнительные опции). После успешного установления соединения можно перейти по URL localhost:8765 через браузер. Браузер может попросить принять исключение безопасности. Пример того, как это может выглядеть:



После успешного создания кластера YDB проверьте его состояние, используя следующую страницу YDB UI: http://localhost:8765/monitoring/cluster/tenants. Это может выглядеть так:

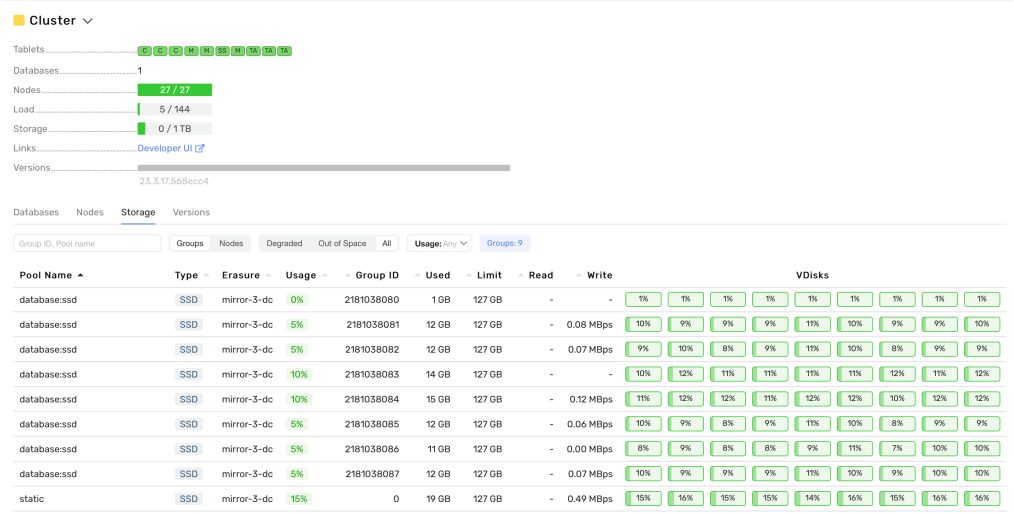


В этом разделе отображаются следующие параметры кластера YDB, отражающие его состояние:

- `Tablets` — список запущенных таблеток. Все индикаторы состояния таблеток должны быть зелёными.
- `Nodes` — количество и состояние узлов хранения и узлов базы данных, запущенных в кластере. Индикатор состояния узлов должен быть зелёным, а количество созданных и запущенных узлов должно быть равным. Например, 18/18 для кластера из девяти узлов с одним узлом базы данных на сервер.

Индикаторы `Load` (количество используемой RAM) и `Storage` (количество используемого дискового пространства) также должны быть зелёными.

Проверить состояние группы хранения можно в разделе `storage` — http://localhost:8765/monitoring/cluster/storage:



Индикаторы `VDisks` должны быть зелёными, а статус `state` (находится в подсказке при наведении на индикатор Vdisk) должен быть `Ok`. Больше об индикаторах состояния кластера и мониторинге можно прочитать в статье YDB Monitoring.

Тестирование кластера

Протестировать кластер можно с помощью встроенных нагрузочных тестов в YDB CLI. Для этого установите YDB CLI и создайте профиль с параметрами подключения, заменив заполнители:

```
ydb \
  config profile create <profile-name> \
  -d /<ydb-domain>/<ydb-database> \
  -e grpcs://<any-ydb-cluster-hostname>:2135 \
  --ca-file $(pwd)/files/TLS/certs/ca.crt \
  --user root \
  --password-file <path-to-a-file-with-password>
```

Параметры команды и их значения:

- `config profile create` — эта команда используется для создания профиля подключения. Укажите имя профиля. Более подробную информацию о том, как создавать и изменять профили, можно найти в статье Создание и изменение профиля.

- `-d` — путь базы данных в виде `/<домен>/<имя_базы>`. Вместо `<ydb-domain>` укажите значение параметра `ydb_domain`, вместо `<ydb-database>` — `ydb_dbname` из `inventory/50-inventory.yaml` (секция «База данных»). При настройках по умолчанию путь будет `/Root/database`.
- `-e` — конечная точка, строка в формате `protocol://host:port`. Допускается указать FQDN любого узла кластера и опустить порт. По умолчанию используется порт 2135.
- `--ca-file` — путь к корневому сертификату для подключений к базе данных по `grpcs`.
- `--user` — пользователь для подключения к базе данных.
- `--password-file` — путь к файлу с паролем. Опустите это, чтобы ввести пароль вручную.

Проверить, создался ли профиль, можно с помощью команды `ydb config profile list`, которая отобразит список профилей. После создания профиля его нужно активировать командой `ydb config profile activate <profile-name>`. Чтобы убедиться, что профиль активирован, можно повторно выполнить команду `ydb config profile list` — активный профиль будет иметь отметку `(active)`.

Для выполнения YQL запроса можно использовать команду `ydb sql -s 'SELECT 1;'`, которая вернёт результат запроса `SELECT 1` в табличной форме в терминал. После проверки соединения можно создать тестовую таблицу командой: `ydb workload kv init --init-upserts 1000 --cols 4`. Это создаст тестовую таблицу `kv_test`, состоящую из 4 столбцов и 1000 строк. Проверить, что таблица `kv_test` создалась и заполнилась тестовыми данными, можно с помощью команды `ydb sql -s 'select * from kv_test limit 10;'`.

В терминал будет выведена таблица из 10 строк. Теперь можно выполнять тестирование производительности кластера. В статье Key-Value нагрузка описаны несколько типов рабочих нагрузок (`upsert`, `insert`, `select`, `read-rows`, `mixed`) и параметры их выполнения. Пример выполнения тестовой рабочей нагрузки `upsert` с параметром для вывода времени выполнения `--print-timestamp` и стандартными параметрами выполнения: `ydb workload kv run upsert --print-timestamp`:

```
Window Txs/Sec Retries Errors  p50(ms) p95(ms) p99(ms) pMax(ms)        Timestamp
1          727 0       0       11      27      71      116     2024-02-14T12:56:39Z
2          882 0       0       10      21      29      38      2024-02-14T12:56:40Z
3          848 0       0       10      22      30      105     2024-02-14T12:56:41Z
4          901 0       0       9       20      27      42      2024-02-14T12:56:42Z
5          879 0       0       10      22      31      59      2024-02-14T12:56:43Z
...
```

После завершения тестов таблицу `kv_test` можно удалить командой: `ydb workload kv clean`. Подробнее о параметрах создания тестовой таблицы и тестах можно прочитать в статье Key-Value нагрузка.

См. также

- Дополнительные примеры конфигурации Ansible
- Перезапуск кластеров YDB, развёрнутых с помощью Ansible
- Обновление конфигурации кластеров YDB, развёрнутых с Ansible
- Обновление версии YDB на кластерах, развёрнутых с помощью Ansible

Развёртывание кластера с использованием конфигурации V2

> **Внимание**
>
> Эта статья посвящена кластерам YDB, в которых используется **конфигурация V2**. Данный способ конфигурирования пока является экспериментальным и доступен только для версий YDB начиная с v25.1. Для использования в продакшене мы рекомендуем выбирать конфигурацию V1 — она является основной и официально поддерживаемой для всех кластеров YDB.

Подготовьте окружение

Перед развёртыванием системы обязательно выполните подготовительные действия. Ознакомьтесь с документом Подготовка к развертыванию.

Создайте директорию для работы

```
mkdir deployment
cd deployment
mkdir -p inventory/group_vars/ydb
mkdir files
```

Создайте конфигурационный файл Ansible

Создайте `ansible.cfg` с конфигурацией Ansible, подходящей для целевой среды развёртывания. Подробности в справочнике по конфигурации Ansible. Дальнейшее руководство предполагает, что поддиректория `./inventory` рабочей директории настроена для использования файлов инвентаризации.

Пример стартового ansible.cfg

> **Примечание**
>
> Использование параметра `StrictHostKeyChecking=no` в `ssh_args` повышает удобство автоматизации, но снижает уровень безопасности SSH-соединения (отключает проверку подлинности хоста). Для production-окружений рекомендуется не указывать этот аргумент и настроить доверенные ключи вручную. Используйте этот параметр только для тестовых и временных установок.

```
[defaults]
conditional_bare_variables = False
force_handlers = True
forks = 300
gathering = explicit
host_key_checking = False
interpreter_python = /usr/bin/python3
inventory = ./inventory
module_name = shell
pipelining = True
private_role_vars = True
retry_files_enabled = False
timeout = 5
vault_password_file = ./ansible_vault_password_file
verbosity = 1
log_path = ./ydb.log

[ssh_connection]
retries = 5
ssh_args = -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no -o ControlMaster=auto -o ControlPe
rsist=60s -o ControlPath=/tmp/ssh-%h-%p-%r -o ServerAliveCountMax=3 -o ServerAliveInterval=10
```

Создайте основной файл инвентаризации

Создайте файл `inventory/group_vars/ydb/all.yaml` и заполните его.

mirror-3-dc-3nodes

```
# Ansible
ansible_user: имя_пользователя
ansible_ssh_private_key_file: "/путь/к/вашему/id_rsa"

# Система
system_timezone: UTC
system_ntp_servers: [time.cloudflare.com, time.google.com, ntp.ripe.net, pool.ntp.org]

# База данных
ydb_user: root
ydb_dbname: db

# Хранилище
ydb_disks:
  - name: /dev/vdb
    label: ydb_disk_1
  - name: /dev/vdc
    label: ydb_disk_2
  - name: /dev/vdd
```

```
    label: ydb_disk_3
ydb_pool_kind: ssd
ydbops_local: true
ydb_cores_dynamic: 2
ydb_dynnodes:
  - {"instance": "a", offset: 1}
ydb_cores_static:  2

# YDB
ydb_version: "версия_системы"
ydb_archive: "{{ ansible_config_file | dirname }}/files/ydbd.tar.gz"

# Дополнительные параметры
ydb_allow_format_drives: true
ydb_skip_data_loss_confirmation_prompt: false
```

Обязательные настройки, которые нужно адаптировать под окружение в выбранном шаблоне:

1. **Настройка SSH доступа.** Укажите пользователя `ansible_user` и путь к приватному ключу `ansible_ssh_private_key_file`, которые Ansible будет использовать для подключения к вашим серверам.
2. **Пути к блочным устройствам в файловой системе.** В секции `ydb_disks` шаблон предполагает, что `/dev/vda` предназначен для операционной системы, а следующие диски, такие как `/dev/vdb`, — для слоя хранения YDB. Метки дисков создаются плейбуками автоматически, и их имена могут быть произвольными.
3. **Версия системы.** В параметре `ydb_version` укажите номер версии YDB, которую нужно установить. Список доступных версий вы найдёте на странице загрузок.

Рекомендуемые настройки для адаптации:

- `ydb_domain`. Это будет первый компонент пути для всех объектов схемы в кластере. Например, можно поместить туда название своей компании, регион кластера и т.д.
- `ydb_dbname`. Это будет второй компонент пути для всех объектов схемы в базе данных. Например, можно поместить туда название сценария использования или проекта.

Дополнительные настройки

Существует несколько вариантов указания того, какие именно исполняемые файлы YDB вы хотите использовать для кластера:

- `ydb_version`: автоматически загрузить один из официальных релизов YDB по номеру версии. Например, `23.4.11`.
- `ydb_archive`: локальный путь файловой системы к архиву дистрибутива YDB, загруженному или иным образом подготовленному заранее.

Для использования федеративных запросов может потребоваться установка коннектора. Плейбук может развернуть fq-connector-go на хостах с динамическими узлами. Используйте следующие настройки:

- `ydb_install_fq_connector` — установите в `true` для установки коннектора.
- Выберите один из доступных вариантов развёртывания исполняемых файлов fq-connector-go:
  - `ydb_fq_connector_version`: автоматически загрузить один из официальных релизов fq-connector-go по номеру версии. Например, `v0.7.1`.
  - `ydb_fq_connector_git_version`: автоматически скомпилировать исполняемый файл fq-connector-go из исходного кода, загруженного из официального репозитория GitHub. Значение настройки — это имя ветки, тега или коммита. Например, `main`.
  - `ydb_fq_connector_archive`: локальный путь файловой системы к архиву дистрибутива fq-connector-go, загруженному или иным образом подготовленному заранее.
  - `ydb_fq_connector_binary`: локальные пути файловой системы к исполняемому файлу fq-connector-go, загруженному или иным образом подготовленному заранее.
- `ydb_tls_dir` — укажите локальный путь к папке с TLS-сертификатами, подготовленными заранее. Она должна содержать файл `ca.crt` и подкаталоги с именами, соответствующими именам хостов узлов, содержащие сертификаты для данного узла. Если не указано, самоподписанные TLS-сертификаты будут сгенерированы автоматически для всего кластера YDB.
- `ydb_brokers` — перечислите FQDN узлов брокеров. Например:

  ```
  ydb_brokers:
      - static-node-1.ydb-cluster.com
      - static-node-2.ydb-cluster.com
      - static-node-3.ydb-cluster.com
  ```

Оптимальное значение настройки `ydb_database_groups` в разделе `vars` зависит от доступных дисков. Предполагая только одну базу данных в кластере, используйте следующую логику:

- Для промышленных развёртываний используйте диски ёмкостью более 800 ГБ с высокой производительностью IOPS, затем выберите значение для этой настройки на основе топологии кластера:
  - Для `block-4-2` установите `ydb_database_groups` на 95% от общего количества дисков, округляя вниз.
  - Для `mirror-3-dc` установите `ydb_database_groups` на 84% от общего количества дисков, округляя вниз.
- Для тестирования YDB на небольших дисках установите `ydb_database_groups` в 1 независимо от топологии кластера.

Значения переменных `system_timezone` и `system_ntp_servers` зависят от свойств инфраструктуры, на которой развёртывается кластер YDB. По умолчанию `system_ntp_servers` включает набор NTP-серверов без учёта географического расположения инфраструктуры, на которой будет развёрнут кластер YDB. Мы настоятельно рекомендуем использовать локальный NTP-сервер для on-premise инфраструктуры и следующие NTP-серверы для облачных провайдеров:

AWS

- `system_timezone`: USA/<region_name>

- `system_ntp_servers`: [169.254.169.123, time.aws.com] Подробнее о настройках NTP-серверов AWS.

Azure

- О том, как настраивается синхронизация времени на виртуальных машинах Azure, можно прочитать в этой статье.

Alibaba

- Специфика подключения к NTP-серверам в Alibaba описана в этой статье.

Yandex Cloud

- `system_timezone`: Europe/Moscow
- `system_ntp_servers`: [0.ru.pool.ntp.org, 1.ru.pool.ntp.org, ntp0.NL.net, ntp2.vniiftri.ru, ntp.ix.ru, ntps1-1.cs.tu-berlin.de] Подробнее о настройках NTP-серверов Yandex Cloud.

Измените пароль пользователя root

Создайте файл `ansible_vault_password_file` с содержимым:

```
password
```

Этот файл содержит пароль, который Ansible будет использовать для автоматического шифрования и расшифровки конфиденциальных данных, например, файлов с паролями пользователей. Благодаря этому пароли не хранятся в открытом виде в репозитории. Подробнее о работе механизма Ansible Vault можно прочитать в официальной документации.

Далее необходимо установить пароль для начального пользователя, указанного в настройке `ydb_user` (по умолчанию `root`). Этот пользователь изначально будет иметь полные права доступа в кластере, но при необходимости это можно изменить позже. Создайте `inventory/group_vars/ydb/vault.yaml` со следующим содержимым (замените `<password>` на фактический пароль):

```
ydb_password: <password>
```

Зашифруйте этот файл с помощью команды `ansible-vault encrypt inventory/group_vars/ydb/vault.yaml`.

Подготовка конфигурационного файла YDB

Создайте файл `files/config.yaml` и заполните его.

mirror-3-dc-3nodes

```
metadata:
  kind: MainConfig
  cluster: ""
  version: 0
config:
  yaml_config_enabled: true
  erasure: mirror-3-dc
  fail_domain_type: disk
  self_management_config:
    enabled: true
  default_disk_type: SSD
  host_configs:
  - host_config_id: 1
    drive:
    - path: /dev/disk/by-partlabel/ydb_disk_1
      type: SSD
    - path: /dev/disk/by-partlabel/ydb_disk_2
      type: SSD
    - path: /dev/disk/by-partlabel/ydb_disk_3
      type: SSD
  hosts:
  - host: ydb-node-zone-a.local
    host_config_id: 1
    location:
      body: 1
      data_center: 'zone-a'
      rack: '1'
  - host: ydb-node-zone-b.local
    host_config_id: 1
    location:
      body: 2
      data_center: 'zone-b'
      rack: '1'
  - host: ydb-node-zone-c.local
    host_config_id: 1
    location:
      body: 3
      data_center: 'zone-c'
      rack: '1'
  actor_system_config:
    use_auto_config: true
    cpu_count: 1
  interconnect_config:
    start_tcp: true
    encryption_mode: OPTIONAL
    path_to_certificate_file: "/opt/ydb/certs/node.crt"
    path_to_private_key_file: "/opt/ydb/certs/node.key"
```

```
    path_to_ca_file: "/opt/ydb/certs/ca.crt"
  grpc_config:
    cert: "/opt/ydb/certs/node.crt"
    key: "/opt/ydb/certs/node.key"
    ca: "/opt/ydb/certs/ca.crt"
    services_enabled:
    - legacy
  security_config:
    enforce_user_token_requirement: true
  client_certificate_authorization:
    request_client_certificate: true
    client_certificate_definitions:
    - member_groups: ["databaseNodes@cert"]
      subject_terms:
      - short_name: "O"
        values: ["YDB"]
  domains_config:
    security_config:
      monitoring_allowed_sids:
      - "root"
      - "ADMINS"
      - "DATABASE-ADMINS"
      administration_allowed_sids:
      - "root"
      - "ADMINS"
      - "DATABASE-ADMINS"
      viewer_allowed_sids:
      - "root"
      - "ADMINS"
      - "DATABASE-ADMINS"
      register_dynamic_node_allowed_sids:
      - databaseNodes@cert
      - root@builtin
```

Для ускорения и упрощения первичного развёртывания YDB конфигурационный файл уже содержит большинство настроек для установки кластера. Достаточно заменить стандартные хосты FQDN в разделе `hosts` и пути к дискам на актуальные в разделе `hosts_configs`.

- Раздел `hosts`:

```
...
hosts:
  - host: ydb-node-zone-a.local
  host_config_id: 1
  location:
    body: 1
    data_center: 'zone-a'
    rack: '1'
...
```

- Раздел `hosts_configs`:

```
...
host_configs:
- host_config_id: 1
  drive:
  - path: /dev/disk/by-partlabel/ydb_disk_1
    type: SSD
  - path: /dev/disk/by-partlabel/ydb_disk_2
    type: SSD
  - path: /dev/disk/by-partlabel/ydb_disk_3
    type: SSD
...
```

Остальные секции и настройки конфигурационного файла остаются без изменений.

Создайте файл `inventory/ydb_inventory.yaml` с содержимым:

```
plugin: ydb_platform.ydb.ydb_inventory
ydb_config: "files/config.yaml"
```

Разверните кластер YDB

После завершения всех описанных выше подготовительных действий фактическое первоначальное развёртывание кластера сводится к выполнению следующей команды из рабочей директории:

```
ansible-playbook ydb_platform.ydb.initial_setup
```

Вскоре после начала будет необходимо подтвердить полную очистку настроенных дисков. Затем завершение развёртывания может занять десятки минут в зависимости от окружения и настроек. Этот плейбук выполняет примерно те же шаги, которые описаны в инструкциях для ручного развёртывания кластера YDB.

Проверьте состояние кластера

На последнем этапе плейбук выполнит несколько тестовых запросов с использованием настоящих временных таблиц для проверки корректной работы. При успешном выполнении для каждого сервера будут отображены статус ok, failed=0 и результаты тестовых запросов (3 и 6) при достаточно подробном выводе плейбука.

Пример вывода

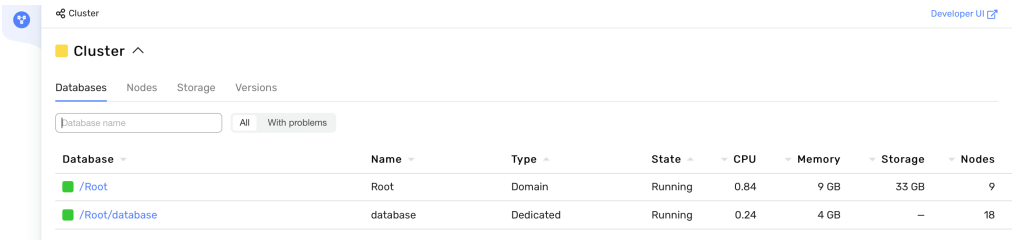
```
...

TASK [ydb_platform.ydb.ydbd_dynamic : run test queries] *************************************************
*****************************************************************************************************
ok: [static-node-1.ydb-cluster.com] => (item={'instance': 'a'}) => {"ansible_loop_var": "item", "change
d": false, "item": {"instance": "a"}, "msg": "all test queries were successful, details: {\"count\":3,\"s
um\":6}\n"}
ok: [static-node-1.ydb-cluster.com] => (item={'instance': 'b'}) => {"ansible_loop_var": "item", "change
d": false, "item": {"instance": "b"}, "msg": "all test queries were successful, details: {\"count\":3,\"s
um\":6}\n"}
ok: [static-node-2.ydb-cluster.com] => (item={'instance': 'a'}) => {"ansible_loop_var": "item", "change
d": false, "item": {"instance": "a"}, "msg": "all test queries were successful, details: {\"count\":3,\"s
um\":6}\n"}
ok: [static-node-2.ydb-cluster.com] => (item={'instance': 'b'}) => {"ansible_loop_var": "item", "change
d": false, "item": {"instance": "b"}, "msg": "all test queries were successful, details: {\"count\":3,\"s
um\":6}\n"}
ok: [static-node-3.ydb-cluster.com] => (item={'instance': 'a'}) => {"ansible_loop_var": "item", "change
d": false, "item": {"instance": "a"}, "msg": "all test queries were successful, details: {\"count\":3,\"s
um\":6}\n"}
ok: [static-node-3.ydb-cluster.com] => (item={'instance': 'b'}) => {"ansible_loop_var": "item", "change
d": false, "item": {"instance": "b"}, "msg": "all test queries were successful, details: {\"count\":3,\"s
um\":6}\n"}
PLAY RECAP *********************************************************************************************
*****************************************************************************************************
static-node-1.ydb-cluster.com : ok=167  changed=80   unreachable=0    failed=0    skipped=167  rescued=0
ignored=0
static-node-2.ydb-cluster.com : ok=136  changed=69   unreachable=0    failed=0    skipped=113  rescued=0
ignored=0
static-node-3.ydb-cluster.com : ok=136  changed=69   unreachable=0    failed=0    skipped=113  rescued=0
ignored=0
```

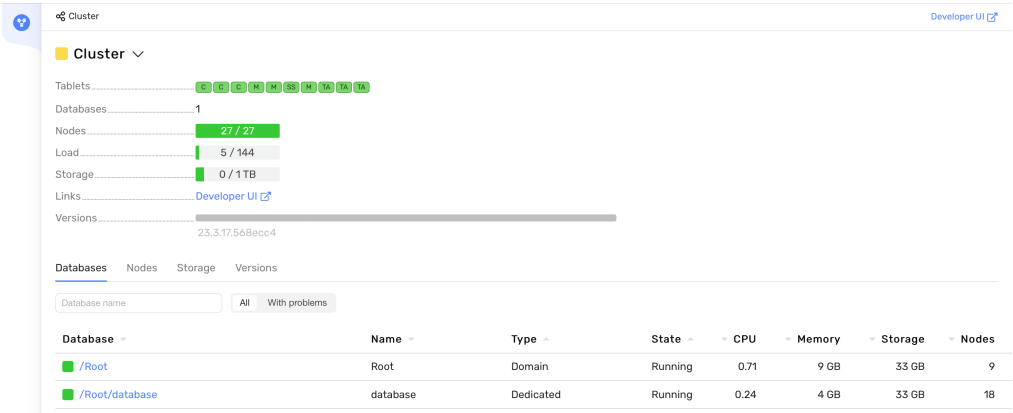
В результате выполнения плейбука `ydb_platform.ydb.initial_setup` будет создан кластер YDB. Он будет содержать домен с именем из настройки `ydb_domain` (по умолчанию `Root`), базу данных с именем из настройки `ydb_dbname` (по умолчанию `db`) и начального пользователя с именем из настройки `ydb_user` (по умолчанию `root`).

Дополнительные шаги

Самый простой способ исследовать только что развёрнутый кластер — использовать YDB UI, работающий на порту 8765 каждого сервера. Если нет прямого доступа к порту из браузера, можно настроить SSH-туннелирование. Для этого выполните команду `ssh -L 8765:localhost:8765 -i <private-key> <user>@<any-ydb-server-hostname>` на локальной машине (при необходимости добавьте дополнительные опции). После успешного установления соединения можно перейти по URL localhost:8765 через браузер. Браузер может попросить принять исключение безопасности. Пример того, как это может выглядеть:



После успешного создания кластера YDB проверьте его состояние, используя следующую страницу YDB UI: http://localhost:8765/monitoring/cluster/tenants. Это может выглядеть так:

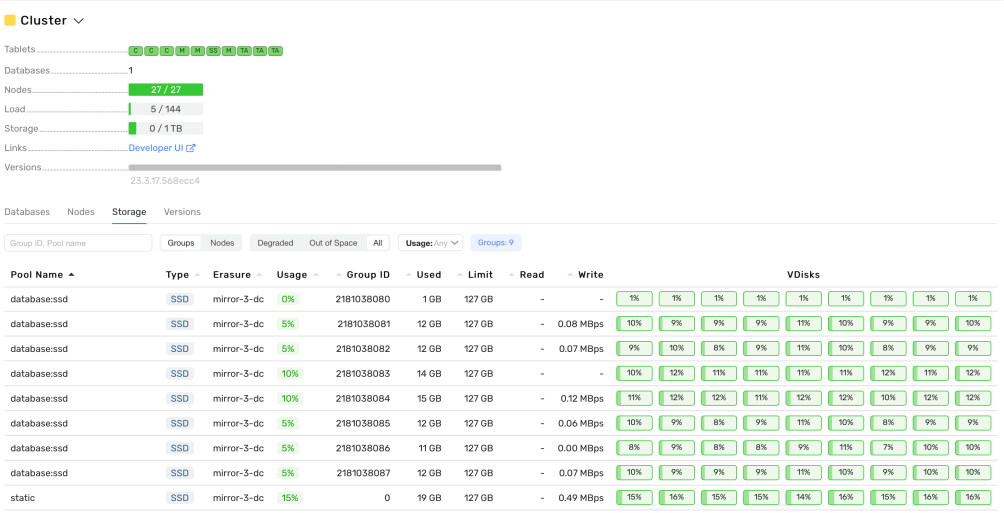


В этом разделе отображаются следующие параметры кластера YDB, отражающие его состояние:

- `Tablets` — список запущенных таблеток. Все индикаторы состояния таблеток должны быть зелёными.
- `Nodes` — количество и состояние узлов хранения и узлов базы данных, запущенных в кластере. Индикатор состояния узлов должен быть зелёным, а количество созданных и запущенных узлов должно быть равным. Например, 18/18 для кластера из девяти узлов с одним узлом базы данных на сервер.

Индикаторы `Load` (количество используемой RAM) и `Storage` (количество используемого дискового пространства) также должны быть зелёными.

Проверить состояние группы хранения можно в разделе `storage` — http://localhost:8765/monitoring/cluster/storage:



Индикаторы `VDisks` должны быть зелёными, а статус `state` (находится в подсказке при наведении на индикатор Vdisk) должен быть `Ok`. Больше об индикаторах состояния кластера и мониторинге можно прочитать в статье YDB Monitoring.

Тестирование кластера

Протестировать кластер можно с помощью встроенных нагрузочных тестов в YDB CLI. Для этого установите YDB CLI и создайте профиль с параметрами подключения, заменив заполнители:

```
ydb \
  config profile create <profile-name> \
  -d /<ydb-domain>/<ydb-database> \
  -e grpcs://<any-ydb-cluster-hostname>:2135 \
  --ca-file $(pwd)/files/TLS/certs/ca.crt \
  --user root \
  --password-file <path-to-a-file-with-password>
```

Параметры команды и их значения:

- `config profile create` — эта команда используется для создания профиля подключения. Укажите имя профиля. Более подробную информацию о том, как создавать и изменять профили, можно найти в статье Создание и изменение профиля.
- `-d` — путь базы данных в виде `/<домен>/<имя_базы>`. Вместо `<ydb-domain>` укажите значение параметра `ydb_domain`, вместо `<ydb-database>` — `ydb_dbname` из `inventory/group_vars/ydb/all.yaml` (секция «База данных»). Для примера из этого руководства путь будет `/Root/db`.
- `-e` — конечная точка, строка в формате `protocol://host:port`. Допускается указать FQDN любого узла кластера и опустить порт. По умолчанию используется порт 2135.
- `--ca-file` — путь к корневому сертификату для подключений к базе данных по `grpcs`.
- `--user` — пользователь для подключения к базе данных.
- `--password-file` — путь к файлу с паролем. Опустите это, чтобы ввести пароль вручную.

Проверить, создался ли профиль, можно с помощью команды `ydb config profile list`, которая отобразит список профилей. После создания профиля его нужно активировать командой `ydb config profile activate <profile-name>`. Чтобы убедиться, что профиль активирован, можно повторно выполнить команду `ydb config profile list` — активный профиль будет иметь отметку `(active)`.

Для выполнения YQL запроса можно использовать команду `ydb sql -s 'SELECT 1;'`, которая вернёт результат запроса `SELECT 1` в табличной форме в терминал. После проверки соединения можно создать тестовую таблицу командой: `ydb workload kv init --init-upserts 1000 --cols 4`. Это создаст тестовую таблицу `kv_test`, состоящую из 4 столбцов и 1000 строк. Проверить, что таблица `kv_test` создалась и заполнилась тестовыми данными, можно с помощью команды `ydb sql -s 'select * from kv_test limit 10;'`.

В терминал будет выведена таблица из 10 строк. Теперь можно выполнять тестирование производительности кластера. В статье Key-Value нагрузка описаны несколько типов рабочих нагрузок (`upsert`, `insert`, `select`, `read-rows`, `mixed`) и параметры их выполнения. Пример выполнения тестовой рабочей нагрузки `upsert` с параметром для вывода времени выполнения `--print-timestamp` и стандартными параметрами выполнения: `ydb workload kv run upsert --print-timestamp`:

```
Window Txs/Sec Retries Errors  p50(ms) p95(ms) p99(ms) pMax(ms)        Timestamp
1          727 0       0       11      27      71      116     2024-02-14T12:56:39Z
2          882 0       0       10      21      29      38      2024-02-14T12:56:40Z
3          848 0       0       10      22      30      105     2024-02-14T12:56:41Z
4          901 0       0       9       20      27      42      2024-02-14T12:56:42Z
5          879 0       0       10      22      31      59      2024-02-14T12:56:43Z
...
```

После завершения тестов таблицу `kv_test` можно удалить командой: `ydb workload kv clean`. Подробнее о параметрах создания тестовой таблицы и тестах можно прочитать в статье Key-Value нагрузка.

См. также

- Дополнительные примеры конфигурации Ansible
- Перезапуск кластеров YDB, развёрнутых с помощью Ansible

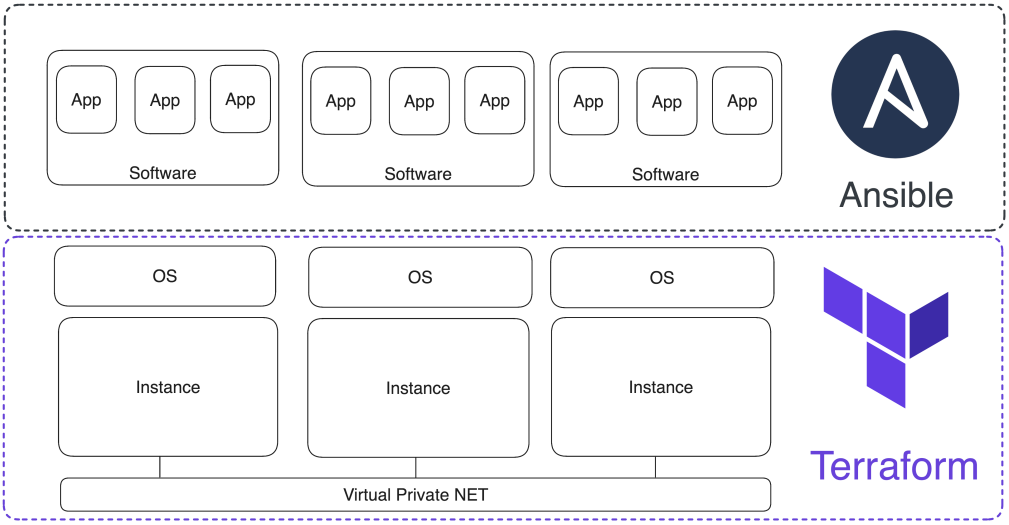
- Обновление конфигурации кластеров YDB, развёрнутых с Ansible
- Обновление версии YDB на кластерах, развёрнутых с помощью Ansible

Развёртывание инфраструктуры для кластера YDB с помощью Terraform

Развернуть кластер YDB для использования в production можно тремя рекомендованными способами: с помощью Ansible, Kubernetes или вручную. Если вариант с Kubernetes практически самодостаточен, то для вариантов с Ansible и вручную нужен SSH-доступ к правильно сконфигурированным серверам или виртуальным машинам.

В статье описывается, как создать и настроить необходимый для работы кластера YDB набор виртуальных машин у различных облачных провайдеров с помощью Terraform.

Terraform – это программное обеспечение с открытым исходным кодом для управления инфраструктурой по модели «инфраструктура как код» (Infrastructure as Code). Такой же подход используется в Ansible, системе управления конфигурациями. Terraform и Ansible работают на разных уровнях: Terraform управляет инфраструктурой, а Ansible настраивает окружения на ВМ:



Конфигурация настройки окружения ВМ описывается в YAML-формате, а инфраструктурный код пишется на HCL (язык конфигурации Terraform). Основной логической единицей записи в HCL является «блок». Блок состоит из ключевого слова, идентифицирующего его тип, названия и фигурных скобок, обозначающих тело блока. Например, так может выглядеть блок управления виртуальным сервером в AWS:

```
resource "aws_instance" "ydb-vm" {
  count                  = var.instance_count
  ami                    = "ami-008fe2fc65df48dac"
  instance_type          = "t2.micro"
  key_name               = var.req_key_pair
  vpc_security_group_ids = [var.input_security_group_id]
  subnet_id              = element(var.input_subnet_ids, count.index % length(var.input_subnet_ids))
  tags = {
    Name                 = "ydb-node-${count.index +1}"
    Username             = "ubuntu"
  }
}
```

Блоки могут располагаться друг за другом в одном файле и быть независимыми, могут ссылаться друг на друга и быть зависимыми, а также могут вкладываться друг в друга.

Основные типы блоков:

- `resource` – блок инициализации ресурса инфраструктуры (ВМ, сеть, подсеть, диск, DNS-зона и т.д.);
- `provider` – блок инициализации провайдера, версии API и данных для аутентификации;
- `variable` – переменная как со значением по умолчанию, так и пустая для хранения данных, введённых пользователем или переданных другими блоками;
- `output` – вывод данных в терминал и сохранение в переменной;
- `data` – переменная для запроса данных от внешних облачных ресурсов, не представленных в создаваемой инфраструктуре;
- `module` – логическая группировка ресурсов, которые можно переиспользовать несколько раз в рамках одного или разных проектов;
- `terraform` – блок настройки поведения самого Terraform, включая версию Terraform и используемых провайдеров, а также настройки бэкенда, который используется для хранения состояния Terraform.

Блоки записываются в файлы с расширением `.tf` и логически группируются в директориях, которые в терминологии Terraform называют модулями. Модуль обычно состоит из следующих файлов:

- `main.tf` – основной файл, в котором находится код инфраструктуры. Может быть несколько файлов, содержащих инфраструктурный код.
- `variables.tf` – локальные переменные модуля, которые принимают данные от других модулей или имеют значения по умолчанию.
- `outputs.tf` – переменные, которые содержат результаты работы ресурса (IP-адреса ВМ, ID сетей/подсетей и т.д.).

Модули подключаются к проекту в корневом файле `main.tf` следующим образом:

```
module "vpc" {
  source                  = "./modules/vpc"
  subnets_count           = var.subnets_count
```

```
  subnets_availability_zones = var.availability_zones
}
```

В примере подключается модуль `vpc` (имя модуля назначается при подключении). Обязательный параметр – это `source`, путь к директории, где располагается модуль. `subnets_count` и `subnets_availability_zones` – это переменные внутри модуля `vpc`, которые принимают значения из переменных глобального уровня `var.subnets_count` и `var.availability_zones`.

Модули, как и блоки, располагаются друг за другом в корневом `main.tf` проекта. Основное преимущество модульного подхода организации проекта – возможность легко управлять логически связанными наборами ресурсов. Поэтому репозиторий с готовыми Terraform-сценариями организован следующим образом:

```
.
├── README.md
├── README_RU.md
├── aws
│   ├── README.md
│   ├── README_RU.md
│   ├── main.tf
│   ├── modules
│   │   ├── dns
│   │   ├── eip
│   │   ├── instance
│   │   ├── key_pair
│   │   ├── security
│   │   └── vpc
│   └── variables.tf
├── azure
│   ├── README.md
│   ├── README_RU.md
│   ├── main.tf
│   ├── modules
│   │   ├── dns
│   │   ├── resource_group
│   │   ├── security
│   │   ├── vm
│   │   └── vpc
│   └── variables.tf
├── ...
```

Поддиректории содержат два `readme`, файл `variables.tf` с локальными переменными модуля и основной файл `main.tf`, который подключает модули из поддиректории `modules`. Набор модулей зависит от облачного провайдера. Базовые модули, функционально одинаковые для всех провайдеров, имеют одинаковые названия:

- `vpc` – модуль управления облачной сетью и подсетями;
- `dns` – модуль управления DNS-зоной и DNS-записями;
- `security` – модуль управления группами безопасности;
- `instance` – модуль управления ВМ.

Чтобы воспользоваться готовыми Terraform-сценариями из репозитория, нужно скачать репозиторий командой `git clone https://github.com/ydb-platform/ydb-terraform.git`, внести изменения в конфигурационный файл Terraform `~/.terraformrc`, задать актуальные значения глобальных переменных сценария и скачать CLI того облачного провайдера, где будет создана инфраструктура.

Если вы планируете использовать несколько провайдеров, можно добавить следующий код в `~/.terraformrc`, который установит пути скачивания для всех провайдеров, описанных ниже:

```
provider_installation {
  network_mirror {
    url     = "https://terraform-mirror.yandexcloud.net/"
    include = ["registry.terraform.io/*/*"]
  }
  direct {
    exclude = ["registry.terraform.io/*/*"]
    exclude = ["terraform.storage.ydb.tech/*/*"]
  }
```

Если уже используются Terraform-провайдеры, представленные в официальном репозитории, они продолжат работать.

Обзор разворачиваемой инфраструктуры

Далее приведены пошаговые инструкции создания инфраструктуры в AWS, Azure, GCP, Yandex Cloud. Предложенные примеры Terraform-сценариев развертывают однотипную инфраструктуру:

- Виртуальные машины в трёх зонах доступности;
- Облачная сеть, публичные и приватные подсети (по подсети на зону доступности);
- Приватная DNS-зона;
- Группы безопасности, разрешающие ICMP и трафик на портах: 22, 65535, 19001, 8765, 2135.

Большинство параметров кластера регулируются (количество ВМ, объём и тип подключаемого диска, количество сетей, домен DNS-зоны и т.д.), однако значения по умолчанию являются минимально рекомендованными. Их изменение в меньшую сторону может привести к проблемам.

Создание инфраструктуры в AWS для развертывания YDB кластера

Создайте аккаунт в AWS и пополните баланс на сумму, достаточную для работы 9 ВМ. Рассчитать примерную стоимость содержания инфраструктуры в зависимости от региона и прочих обстоятельств можно с помощью калькулятора.

Создайте пользователя и ключ подключения в AWS Cloud для работы AWS CLI:

1. Пользователь создаётся в разделе Security credentials → Access management → Users → Create User.
2. На следующем шаге нужно назначить права пользователю. Выберете `AmazonEC2FullAccess`.
3. После создания пользователя – перейдите в него, откройте вкладку `Security credentials` и нажмите кнопку **Create access key** в секции **Access keys**.
4. Из предложенных вариантов наборов опций выберете `Command Line Interface`.
5. Далее придумайте тег для разметки ключа и нажмите кнопку **Create access key**.
6. Скопируйте значения полей `Access key` и `Secret access key`.

Установите AWS CLI и выполните команду `aws configure`. Введите значения полей `Access key` и `Secret access key` сохраненные ранее. Отредактируйте файлы `~/.aws/credentials` и `~/.aws/config` следующим образом:

1. Добавьте `[AWS_def_reg]` в `~/.aws/config` перед `region = ...`.
2. Добавьте `[AWS]` перед секретной информацией о ключах подключения.

Перейдите в директорию `aws` в скачанном репозитории и отредактируйте следующие переменные в файле `variable.tf`:

1. `aws_region` – регион, в котором будет развернута инфраструктура.
2. `aws_profile` – имя профиля безопасности из файла `~/.aws/credentials`.
3. `availability_zones` – список зон доступности региона. Формируется из названия региона и порядковой буквы. Например, для региона `us-west-2` список зон доступности будет выглядеть так: `["us-west-2a", "us-west-2b", "us-west-2c"]`.

Теперь, находясь в поддиректории `aws`, можно выполнить последовательность следующих команд для установки провайдера, инициализации модулей и создания инфраструктуры:

1. `terraform init` – установка провайдера и инициализация модулей.
2. `terraform plan` – создание плана будущей инфраструктуры.
3. `terraform apply` (повторное выполнение) – создание ресурсов в облаке.

Далее используются команды `terraform plan`, `terraform apply` и `terraform destroy` (уничтожение созданной инфраструктуры).

Создание инфраструктуры в Azure для развертывания YDB кластера

Создайте аккаунт в Azure и пополните счёт аккаунта на сумму, достаточную для работы девяти ВМ. Рассчитать примерную стоимость содержания инфраструктуры в зависимости от региона и прочих обстоятельств можно с помощью калькулятора.

Аутентификация в провайдере Azure для Terraform осуществляется через CLI:

1. Скачать, установить и настроить Azure CLI можно, следуя этой инструкции.
2. Авторизоваться в Azure с помощью Azure CLI можно в интерактивном режиме командой `az login`, а для создания пары SSH-ключей (Linux, macOS) проще всего будет воспользоваться командой `ssh-keygen`.

После авторизации в Azure и генерации SSH-ключей нужно изменить дефолтное значение следующих переменных в корневом файле `variables.tf`:

1. `auth_location` – название региона, где будет развернута инфраструктура. Список доступных регионов в зависимости от подписки можно получить командой `az account list-locations | grep "displayName"`.
2. `ssh_key_path` – путь к публичной части сгенерированного SSH-ключа.

Теперь, находясь в поддиректории `azure`, можно выполнить последовательность следующих команд для установки провайдера, инициализации модулей и создания инфраструктуры:

1. `terraform init` – установка провайдера и инициализация модулей.
2. `terraform plan` – создание плана будущей инфраструктуры.
3. `terraform apply` (повторное выполнение) – создание ресурсов в облаке.

Далее используются команды `terraform plan`, `terraform apply` и `terraform destroy` (уничтожение созданной инфраструктуры).

Создание инфраструктуры в Google Cloud Platform для развертывания YDB кластера

Зарегистрируйтесь в Google Cloud Console и создайте проект. Активируйте платежный аккаунт и пополните его средствами для запуска девяти ВМ. Рассчитать стоимость можно в калькуляторе.

Настройте GCP CLI:

1. Актируйте Compute Engine API и Cloud DNS API.
2. Скачайте и установите GCP CLI, следуя этой инструкции.
3. Перейдите в поддиректорию `.../google-cloud-sdk/bin` и выполните команду `./gcloud compute regions list` для получения списка доступных регионов.
4. Выполните команду `./gcloud auth application-default login` для настройки профиля подключения.

Перейдите в поддиректорию `gcp` (находится в скаченном репозитории) и в файле `variables.tf` задайте актуальные значения следующим переменным:

1. `project` – название проекта, которое было задано в облачной консоли Google Cloud.
2. `region` – регион, где будет развернута инфраструктура.
3. `zones` – список зон доступности, в которых будут созданы подсети и ВМ.

Теперь, находясь в поддиректории `gcp`, можно выполнить последовательность следующих команд для установки провайдера, инициализации модулей и создания инфраструктуры:

1. `terraform init` – установка провайдера и инициализация модулей.
2. `terraform plan` – создание плана будущей инфраструктуры.
3. `terraform apply` (повторное выполнение) – создание ресурсов в облаке.

Далее используются команды `terraform plan`, `terraform apply` и `terraform destroy` (уничтожение созданной инфраструктуры).

Создание инфраструктуры в Yandex Cloud для развертывания YDB кластера

Для создания инфраструктуры в Yandex Cloud с помощью Terraform нужно:

1. Подготовить облако к работе:
   - Зарегистрироваться в Yandex Cloud.
   - Подключить платежный аккаунт.
   - Убедится в наличии достаточного количества средств для создания девяти ВМ.
2. Установить и настроить Yandex Cloud CLI:
   - Скачать Yandex Cloud CLI.
   - Создать профиль
3. Создать сервисный аккаунт с помощью CLI.
4. Сгенерировать авторизованный ключ в JSON формате для подключения Terraform к облаку с помощью CLI:

```
yc iam key create --service-account-name <acc name> \
  --output <file name> --folder-id <cloud folder id>
```

В терминал будет выведена информация о созданном ключе:

```
```text
id: ajenap572v8e1l...
service_account_id: aje90em65r69...
created_at: "2024-09-03T15:34:57.495126296Z"
key_algorithm: RSA_2048
```

Авторизованный ключ будет создан в директории, где вызывалась команда.
```

1. Настроить Yandex Cloud Terraform провайдера.
2. Скачать данный репозиторий командой `git clone https://github.com/ydb-platform/ydb-terraform.git`.
3. Перейти в директорию `yandex_cloud` (директория в скаченном репозитории) и внести изменения в следующие переменные, в файле `variables.tf`:
   - `key_path` – путь к сгенерированному авторизованному ключу с помощью CLI.
   - `cloud_id` – ID облака. Можно получить список доступных облаков командой `yc resource-manager cloud list`.
   - `folder_id` – ID Cloud folder. Можно получить командой `yc resource-manager folder list`.

Теперь, находясь в поддиректории `yandex_cloud`, можно выполнить последовательность следующих команд для установки провайдера, инициализации модулей и создания инфраструктуры:

1. `terraform init` – установка провайдера и инициализация модулей.
2. `terraform plan` – создание плана будущей инфраструктуры.
3. `terraform apply` (повторное выполнение) – создание ресурсов в облаке.

Далее используются команды `terraform plan`, `terraform apply` и `terraform destroy` (уничтожение созданной инфраструктуры).

> **Примечание**
>
> С помощью Yandex Cloud провайдера можно не только создавать инфраструктуру для дальнейшего развертывания на ней YDB кластера с помощью Ansible, но и управлять serverless или dedicated версией YDB прямо из Terraform. О возможностях работы с YDB в Yandex Cloud читайте в разделе Работа с YDB через Terraform документации Yandex Cloud.

Перезапуск кластеров YDB, развёрнутых с помощью Ansible

Кластеры YDB обеспечивают высокие гарантии доступности; поэтому необходимо учитывать модель отказоустойчивости кластера во время любого технического обслуживания, включая перезапуск кластера. Существует два типа узлов, которые могут потребовать перезапуска:

- Database узлы (также известные как динамические) не сохраняют состояние; поэтому основное внимание стоит уделять тому, чтобы их было достаточно для обработки нагрузки каждой базы данных. Обычно для динамических узлов достаточно простого поэтапного перезапуска с небольшими задержками.
- Storage узлы (также известные как статические) имеют персистентное состояние и отвечают за безопасное хранение данных. Поэтому они требуют особого обращения для обеспечения доступности данных. Каждый кластер YDB имеет выделенный компонент, который отслеживает все сбои и обслуживание и может определить, безопасно ли сейчас остановить или перезапустить конкретный узел. Поэтому важно запрашивать у него разрешение на каждую операцию, а полный перезапуск узлов хранения часто занимает некоторое время.

Перезапуск через Ansible playbook

Репозиторий ydb-ansible содержит playbook под названием `ydb_platform.ydb.restart`, который можно использовать для перезапуска кластера YDB. Запускайте его из той же директории, которая использовалась для изначального развёртывания.

Перезапуск всех узлов

По умолчанию `ydb_platform.ydb.restart` перезапускает все узлы кластера. Сначала идут статические узлы, затем динамические. Команда для его запуска:

```
ansible-playbook ydb_platform.ydb.restart
```

Фильтр по типу узлов

Задачи в playbook `ydb_platform.ydb.restart` помечены типами узлов, поэтому вы можете использовать функциональность тегов Ansible для фильтрации узлов по их типу.

Эти две команды эквивалентны и перезапустят все узлы хранения:

```
ansible-playbook ydb_platform.ydb.restart --tags storage
ansible-playbook ydb_platform.ydb.restart --tags static
```

Эти две команды эквивалентны и перезапустят все узлы баз данных:

```
ansible-playbook ydb_platform.ydb.restart --tags database
ansible-playbook ydb_platform.ydb.restart --tags dynamic
```

Фильтр по имени хоста

Чтобы перезапустить конкретный хост или подмножество хостов, используйте аргумент `--limit`:

```
ansible-playbook ydb_platform.ydb.restart --limit='<hostname>'
ansible-playbook ydb_platform.ydb.restart --limit='<hostname-1,hostname-2>'
```

Его можно использовать и вместе с тегами:

```
ansible-playbook ydb_platform.ydb.restart --tags database --limit='<hostname>'
```

Перезапуск узлов вручную

Инструмент ydbops реализует различные манипуляции с кластерами YDB, включая перезапуск. Описанный выше playbook `ydb_platform.ydb.restart` использует его за сценой, но использование его вручную тоже допустимо.

Больше рекомендаций и информации о том, как это работает, находится в статье Обслуживание кластера без потери доступности.

Обновление конфигурации кластеров YDB, развёрнутых с Ansible

Во время первоначального развёртывания Ansible playbook использует предоставленный конфигурационный файл для создания начальной конфигурации кластера. Технически, он генерирует два варианта конфигурационного файла на основе исходного и размещает их на всех хостах через механизм Ansible для копирования файлов между серверами. В этой статье рассматриваются доступные способы для изменения конфигурации кластера после первоначального развёртывания.

Обновление конфигурации через Ansible playbook

В репозитории ydb-ansible есть playbook под названием `ydb_platform.ydb.update_config`, который можно использовать для обновления конфигурации кластера YDB. Перейдите в ту же директорию, которая использовалась для первоначального развёртывания кластера, отредактируйте файл `files/config.yaml` по необходимости и затем запустите этот playbook:

```
ansible-playbook ydb_platform.ydb.update_config
```

Этот playbook развернёт новую версию конфигурационных файлов и затем выполнит постепенную перезагрузку кластера.

Фильтрация по типу узла

Задачи в playbook `ydb_platform.ydb.update_config` помечены типами узлов, поэтому вы можете использовать функциональность тегов Ansible для фильтрации узлов по их типу.

Эти две команды эквивалентны и изменят конфигурацию всех узлов хранения:

```
ansible-playbook ydb_platform.ydb.update_config --tags storage
ansible-playbook ydb_platform.ydb.update_config --tags static
```

Эти две команды эквивалентны и изменят конфигурацию всех узлов баз данных:

```
ansible-playbook ydb_platform.ydb.update_config --tags database
ansible-playbook ydb_platform.ydb.update_config --tags dynamic
```

Пропуск перезагрузки

Также есть тег `no_restart`, который позволяет только обновить конфигурационные файлы и пропустить перезагрузку кластера. Это может быть полезно, если кластер будет перезагружен позже вручную или в рамках других задач по обслуживанию. Пример запуска:

```
ansible-playbook ydb_platform.ydb.update_config --tags no_restart
```

Обновление версии YDB на кластерах, развёрнутых с помощью Ansible

Во время начального развёртывания Ansible playbook предоставляет на выбор несколько вариантов, какой именно серверный исполняемый файл YDB (`ydbd`) использовать. В этой статье объясняются доступные варианты изменения версии кластера после начального развёртывания.

> **Важно**
>
> У YDB существуют специфические правила относительно совместимости версий. Важно ознакомиться с руководством по именованию версий YDB и их совместимости и списком изменений, чтобы правильно выбрать новую версию для обновления и подготовиться к возможным нюансам.

Обновление исполняемых файлов через Ansible playbook

Репозиторий ydb-ansible содержит playbook под названием `ydb_platform.ydb.update_executable` , который можно использовать для обновления или понижения версии кластера YDB. Перейдите в ту же директорию, которая использовалась для начального развёртывания, отредактируйте файл `inventory/50-inventory.yaml` , чтобы указать целевую версию YDB для установки (обычно через переменные `ydb_version` или `ydb_git_version`), а затем выполните этот playbook:

```
ansible-playbook ydb_platform.ydb.update_executable
```

Playbook получает новый бинарный файл и затем разворачивает его на кластере с помощью кросс-серверного копирования Ansible. После этого он выполняет постепенную перезагрузку кластера.

Фильтрация по типу узла

Задачи в playbook `ydb_platform.ydb.update_executable` помечены типами узлов, поэтому можно использовать функциональность тегов Ansible для фильтрации узлов по их типу.

Эти две команды эквивалентны и изменят конфигурацию всех узлов хранения:

```
ansible-playbook ydb_platform.ydb.update_executable --tags storage
ansible-playbook ydb_platform.ydb.update_executable --tags static
```

Эти две команды эквивалентны и изменят конфигурацию всех узлов баз данных:

```
ansible-playbook ydb_platform.ydb.update_executable --tags database
ansible-playbook ydb_platform.ydb.update_executable --tags dynamic
```

Пропуск перезагрузки

Существует тег `no_restart` , чтобы только развернуть исполняемые файлы, а перезагрузку кластера пропустить. Это может быть полезно, если кластер будет перезагружен позже вручную или в рамках других задач по обслуживанию. Пример запуска:

```
ansible-playbook ydb_platform.ydb.update_executable --tags no_restart
```

Обеспечение наблюдаемости кластера YDB, развёрнутого с помощью Ansible

В этом разделе документации YDB рассматриваются различные аспекты наблюдаемости (observability), специфичные для кластеров, развёрнутых с помощью Ansible.

- Логирование в кластерах, развёрнутых с помощью Ansible

Также есть отдельный раздел Справка по наблюдаемости YDB, содержащий статьи, не зависящие от выбранного метода развёртывания кластера.

Логирование в кластерах, развёрнутых с помощью Ansible

Во время первоначального развёртывания Ansible playbook настраивает несколько systemd юнитов, которые управляют узлами YDB. Как правило, на каждом физическом сервере или виртуальной машине работает несколько узлов YDB, каждый из которых имеет свой собственный лог. Существует два основных способа просмотра логов такого кластера: через Ansible playbook или через ssh.

Просмотр логов с помощью Ansible playbook

Репозиторий ydb-ansible содержит playbook под названием `ydb_platform.ydb.logs`, который можно использовать для просмотра логов со всех узлов кластера YDB. Этот playbook собирает логи с узлов и выводит их в `stdout`, что позволяет при необходимости дальше их обработать, например, с помощью команд `grep` или `awk`.

Все логи всех узлов

По умолчанию playbook `ydb_platform.ydb.logs` извлекает логи всех узлов YDB. Команда для этого:

```
ansible-playbook ydb_platform.ydb.logs
```

Фильтрация по типу узла

В кластере YDB есть два основных типа узлов:

- Storage (также известные как статические)
- Database (также известные как динамические)

Задачи в playbook `ydb_platform.ydb.logs` размечены тегами по типам узлов, благодаря чему можно использовать функциональность тегов Ansible для фильтрации логов по типу узла.

Следующие две команды эквивалентны и будут выводить логи статических узлов:

```
ansible-playbook ydb_platform.ydb.logs --tags storage
ansible-playbook ydb_platform.ydb.logs --tags static
```

Эти две команды также эквивалентны и будут выводить логи динамических узлов:

```
ansible-playbook ydb_platform.ydb.logs --tags database
ansible-playbook ydb_platform.ydb.logs --tags dynamic
```

Фильтрация по имени хоста

Чтобы отобразить логи определённого хоста или подмножества хостов, используйте аргумент `--limit`:

```
ansible-playbook ydb_platform.ydb.logs --limit='<hostname>'
ansible-playbook ydb_platform.ydb.logs --limit='<hostname-1,hostname-2>'
```

Его также можно использовать вместе с тегами:

```
ansible-playbook ydb_platform.ydb.logs --tags database --limit='<hostname>'
```

Просмотр логов по ssh

Чтобы вручную получить доступ к логам кластера YDB через `ssh`, выполните следующие действия:

1. Сформируйте команду `ssh` для доступа к серверу, на котором запущен узел YDB, для которого вам нужны логи. Базовая версия будет выглядеть как `ssh -i <path-to-ssh-key> <username>@<hostname>`. Возьмите значения для составления этой команды из файла `inventory/50-inventory.yaml`, который вы использовали для развёртывания:
   - `<path-to-ssh-key>` — это `children.ydb.ansible_ssh_private_key_file`;
   - `<username>` — это `children.ydb.ansible_user`;
   - `<hostname>` — это один из `children.ydb.hosts`.
2. Выберите, логи какого systemd юнита вам нужны. Если вам уже известно название юнита, этот шаг можно пропустить. После входа на сервер с помощью команды `ssh`, созданной на предыдущем шаге, получите список связанных с YDB systemd юнитов, используя `systemctl list-units | grep ydb`. Как правило, там будет один статический узел и несколько динамических узлов.

   Пример вывода

   ```
   $ systemctl list-units | grep ydb
   ydb-transparent-hugepages.service                                                 loaded active     exit
   ed    Configure Transparent Huge Pages (THP)
   ydbd-database-a.service                                                           loaded active     runn
   ing   YDB dynamic node / database / a
   ydbd-database-b.service                                                           loaded active     runn
   ing   YDB dynamic node / database / b
   ydbd-storage.service                                                              loaded active     runn
   ing   YDB storage node
   ```

3. Возьмите название systemd юнита из предыдущего шага и используйте его в следующей команде `journalctl -u <systemd-unit>`, чтобы фактически отобразить логи. Вы можете указать `-u` несколько раз, чтобы отобразить логи нескольких юнитов, а также использовать любые другие аргументы из `man journalctl` для настройки вывода команды.

Работа с YDB в Kubernetes

Этот раздел документации YDB содержит коллекцию статей по администрированию кластеров YDB через Kubernetes. Это рекомендуемый подход к запуску production кластеров YDB в контейнерных окружениях. Для работы с кластерами YDB на виртуальных машинах или физических серверов, используйте Ansible вместо него.

Основные статьи для начала работы с этим разделом:

- Первоначальное развертывание

Начало работы с YDB в Kubernetes

Развертывание YDB в Kubernetes — это простой способ установки и эксплуатации YDB кластера. С Kubernetes вы можете использовать универсальный подход к управлению приложением в любом облачном провайдере. Это руководство содержит инструкции для развертывания YDB в AWS EKS или Yandex Managed Service for Kubernetes.

Пререквизиты

YDB поставляется в виде Helm-чарта — пакета, который содержит шаблоны структур Kubernetes. Подробнее о Helm читайте в документации. Чарт YDB может быть развернут в следующем окружении:

1. Кластер Kubernetes версии 1.20 и старше. Он должен поддерживать динамическое предоставление томов (Dynamic Volume Provisioning). Следуйте иструкциям ниже, если у вас ещё не настроен подходящий кластер.
2. Установлена утилита kubectl и настроен доступ к кластеру.
3. Установлен менеджер пакетов Helm версии старше 3.1.0.

Для эффективной работы YDB рекомендуется использовать физические (не виртуальные) диски объемом более 800 ГБ как блочные устройства.

Минимальный объем диска должен быть не менее 80 ГБ, при меньшем объеме узел YDB не сможет использовать устройство. Корректная и бесперебойная работа с дисками минимального объема не гарантируется. Использовать такие диски рекомендуется исключительно в ознакомительных целях.

> **Важно**
>
> Конфигурации с дисками объемом меньше 800 ГБ или с любыми видами виртуализации системы хранения нельзя использовать для сервисов, находящихся в промышленной эксплуатации, а также для тестирования производительности системы.
>
> Мы не рекомендуем использовать для хранения данных YDB диски, которые используются другими процессами (в том числе операционной системой).

Создание Kubernetes кластера

Пропустите этот раздел, если у вас уже имеется подходящий Kubernetes

AWS EKS

1. Настройте утилиты `awscli` и `eksctl` для работы с ресурсами AWS по документации.
2. Настройте `kubectl` для работы с кластером Kubernetes.
3. Выполните следующую команду:

```
eksctl create cluster \
  --name ydb \
  --nodegroup-name standard-workers \
  --node-type c5a.2xlarge \
  --nodes 3 \
  --nodes-min 1 \
  --nodes-max 4
```

   Это команда создаст Kubernetes кластер с именем `ydb`. Флаг `--node-type` указывает, что кластер будет развернут с использованием инстансов `c5a.2xlarge` (8vCPUs, 16 GiB RAM), что соответствует рекомендациям по запуску YDB.

   Создание Kubernetes кластера занимает в среднем от 10 до 15 минут. Дождитесь завершения процесса перед переходом к следующим шагам развертывания YDB. Конфигурация `kubectl` будет автоматически обновлена для работы с кластером после его создания.

Yandex Managed Service for Kubernetes

1. Создайте кластер Kubernetes.

   Вы можете использовать уже работающий кластер Kubernetes или создать новый.

   Как создать кластер Kubernetes в Yandex Managed Service for Kubernetes

   Следуйте инструкциям по началу работы с Yandex Managed Service for Kubernetes.

Обзор YDB Helm-чарта

Helm-чарт устанавливает YDB Kubernetes Operator в Kubernetes кластер. Он представляет собой контроллер, построенный по паттерну Оператор. Он реализует необходимую логику для развертывания и управления компонентами YDB.

Кластер YDB состоит из двух видов узлов:

- **Storage nodes** (ресурс Storage) — обеспечивают слой хранения данных;
- **Dynamic nodes** (ресурс Database) — реализуют доступ к данным и их обработку.

Для развертывания кластера YDB в Kubernetes необходимо создать оба эти ресурса. Этот процесс будет рассмотрен подробнее ниже. Схема этих ресурсов располагается на GitHub.

После обработки чарта контроллером будут созданы следующие ресурсы:

- StatefulSet — контроллер рабочей нагрузки, который предоставляет предсказуемые сетевые имена и дисковые ресурсы для каждого контейнера.
- Service для доступа к созданным базам данных из приложений.
- ConfigMap для хранения конфигурации кластера.

Ознакомиться с исходным кодом оператора можно на GitHub, Helm-чарт расположен в папке deploy.
При разворачивании контейнеров YDB используются образы `cr.yandex/yc/ydb`, на данный момент доступные только как

предсобранные артефакты.

Подготовка рабочего окружения

1. Добавьте в Helm репозиторий для YDB:

   Выполните команду:

   ```
   helm repo add ydb https://charts.ydb.tech/
   ```

   - `ydb` — алиас репозитория;
   - `https://charts.ydb.tech/` — URL репозитория YDB.

   Результат выполнения:

   ```
   "ydb" has been added to your repositories
   ```

2. Обновите индекс чартов Helm:

   Выполните команду:

   ```
   helm repo update
   ```

   Результат выполнения:

   ```
   Hang tight while we grab the latest from your chart repositories...
   ...Successfully got an update from the "ydb" chart repository
   Update Complete. ⎈Happy Helming!⎈
   ```

Развертывание кластера YDB

Установите YDB Kubernetes оператор

Разверните YDB Kubernetes оператор на кластере с помощью `helm`, выполните команду:

```
helm install ydb-operator ydb/ydb-operator
```

- `ydb-operator` — имя установки;
- `ydb/ydb-operator` — название чарта в добавленном ранее репозитории.

Результат выполнения:

```
NAME: ydb-operator
LAST DEPLOYED: Thu Aug 12 19:32:28 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Разверните узлы хранения

YDB поддерживает различные топологии хранения данных. YDB Kubernetes оператор имеет несколько примеров конфигураций для типовых топологий. В данной инструкции они используются без изменений, но их можно подстраивать под свои нужды или написать новый конфигурационный файл с нуля.

Примените манифест для создания узлов хранения:

block-4-2

```
kubectl apply -f https://raw.githubusercontent.com/ydb-platform/ydb-kubernetes-operator/master/samples/storage-block-4-2.yaml
```

Это создаст 8 узлов хранения YDB, сохраняющих данные с использованием erasure coding. В этом случае используется лишь +50% дополнительного дискового пространства для обеспечения отказоустойчивости.

mirror-3-dc

```
kubectl apply -f https://raw.githubusercontent.com/ydb-platform/ydb-kubernetes-operator/master/samples/storage-mirror-3dc.yaml
```

Это создаст 9 узлов хранения YDB, которые сохраняют данные с фактором репликации 3.

Эта команда создаст объект `StatefulSet`, который описывает набор контейнеров с предсказуемыми сетевыми именами и закрепленными за ними дисками, а также необходимые для работы кластера объекты `Service` и `ConfigMap`.

Узлам хранения YDB требуется время для инициализации. За прогрессом инициализации можно следить с помощью команд `kubectl get storages.ydb.tech` или `kubectl describe storages.ydb.tech`. Дождитесь пока статус ресурса `Storage` станет `Ready`.

Важно

Конфигурация кластера статична, контроллер не будет обрабатывать изменения при повторном применении манифеста. Изменение таких параметров кластера, как версия или размер дисков, возможно только через пересоздание кластера.

Создайте базу данных и динамические узлы

YDB база данных представляет собой логическую сущность, обслуживаемую динамическими узлами. Пример конфигурационного файла, который поставляется с YDB Kubernetes оператором, создаёт базу данных с именем `database-sample` и 3 динамическими узлами. Как и в случае с узлами хранения, конфигурацию можно изменять под свои нужды.

Примените манифест для создания базы данных и динамических узлов:

```
kubectl apply -f https://raw.githubusercontent.com/ydb-platform/ydb-kubernetes-operator/master/samples/database.yaml
```

Примечание

Значение по ключу `.spec.storageClusterRef.name` должно совпадать с именем ресурса `Storage` с узлами хранения.

После обработки манифеста будет создан объект `StatefulSet`, который описывает набор динамических нод. Созданная база данных будет доступна изнутри Kubernetes кластера по имени `database-sample`, или по FQDN-имени `database-sample.<namespace>.svc.cluster.local`, где `namespace` — название пространства имен, использовавшегося при установке. Подключиться к базе данных можно по порту 2135.

Посмотрите статус созданного ресурса:

```
kubectl describe database.ydb.tech
```

Результат выполнения команды:

```
Name:         database-sample
Namespace:    default
Labels:       <none>
Annotations:  <none>
API Version:  ydb.tech/v1alpha1
Kind:         Database
...
  Storage Endpoint:  grpc://storage-sample-grpc.default.svc.cluster.local:2135
Status:
  State:  Ready
Events:
  Type     Reason              Age    From          Message
  ----     ------              ----   ----          -------
  Normal   Provisioning        8m10s  ydb-operator  Resource sync is in progress
  Normal   Provisioning        8m9s   ydb-operator  Resource sync complete
  Normal   TenantInitialized   8m9s   ydb-operator  Tenant /root/database-sample created
```

`State: Ready` означает, что база данных готова к работе.

Проверьте работу кластера

Проверьте работоспособность YDB:

1. Проверьте, что все узлы в состоянии `Running`:

   ```
   kubectl get pods
   ```

   Результат:

   ```
   NAME                READY   STATUS    RESTARTS   AGE
   database-sample-0   1/1     Running   0          1m
   database-sample-1   1/1     Running   0          1m
   database-sample-2   1/1     Running   0          1m
   database-sample-3   1/1     Running   0          1m
   database-sample-4   1/1     Running   0          1m
   database-sample-5   1/1     Running   0          1m
   storage-sample-0    1/1     Running   0          1m
   storage-sample-1    1/1     Running   0          1m
   storage-sample-2    1/1     Running   0          1m
   storage-sample-3    1/1     Running   0          1m
   storage-sample-4    1/1     Running   0          1m
   storage-sample-5    1/1     Running   0          1m
   storage-sample-6    1/1     Running   0          1m
   storage-sample-7    1/1     Running   0          1m
   storage-sample-8    1/1     Running   0          1m
   ```

2. Запустите новый под с YDB CLI:

   ```
   kubectl run -it --image=cr.yandex/crptqonuodf51kdj7a7d/ydb:24.4.4.2 --rm ydb-cli bash
   ```

3. Выполните запрос к базе данных YDB (endpoint можно получить из вывода `kubectl describe database.ydb.tech`):

```
/opt/ydb/bin/ydb \
  --endpoint grpc://database-sample-grpc:2135 \
  --database /Root/database-sample \
  sql -s 'SELECT 2 + 2;'
```

   - `--endpoint` — эндпоинт базы данных;
   - `--database` — имя созданной базы данных;
   - `--query` — текст запроса.

   Результат:

```
┌─────────┐
| column0 |
├─────────┤
| 4       |
└─────────┘
```

Дальнейшие шаги

После проверки, что созданный кластер YDB работает нормально, его можно использовать в соответствии с вашими задачами. Например, если вы хотите просто продолжить экспериментировать, можете использовать этот кластер, чтобы пройти YQL туториал.

Также, ниже описаны ещё несколько аспектов, которые можно учесть далее.

Мониторинг

YDB предоставляет стандартные механизмы сбора логов и метрик. Логирование осуществляется в стандартные каналы `stdout` и `stderr` и может быть перенаправлено при помощи популярных решений. Например, можно использовать комбинацию из Fluentd и Elastic Stack.

Для сбора метрик `ydb-controller` предоставляет ресурсы типа `ServiceMonitor`, которые могут быть обработаны с помощью kube-prometheus-stack.

Выделение ресурсов

Каждый под YDB может быть ограничен в потреблении ресурсов. Если оставить значения ограничений пустыми, поду будет доступно все процессорное время и вся память виртуальной машины, что может привести к нежелательным последствиям. Мы рекомендуем всегда явно указывать лимиты ресурсов.

Более детально ознакомиться с принципами распределения и ограничения ресурсов можно в документации Kubernetes.

Освобождение неиспользуемых ресурсов

Если созданный кластер YDB больше не нужен, удалите его с помощью следующих шагов:

1. Чтобы удалить базу данных YDB и её динамические узлы, достаточно удалить сопоставленный с ней ресурс `Database`:

```
kubectl delete database.ydb.tech database-sample
```

2. Чтобы удалить узлы хранения YDB, выполните следующие команды:

```
kubectl delete storage.ydb.tech storage-sample
kubectl delete pvc -l app.kubernetes.io/name=ydb
```

3. Чтобы удалить YDB Kubernetes оператор с кластера, используйте Helm:

```
helm delete ydb-operator
```

Обзор управления кластером YDB вручную

В данном разделе описано развёртывание, конфигурирование, обслуживание, мониторинг и диагностика многоузловых кластеров YDB без использования систем оркестрации и автоматизации (таких как Kubernetes или специализированных инструментов управления). При таком подходе администратор самостоятельно выполняет все операции по установке компонентов, настройке конфигурации и обслуживанию кластера, используя командную строку и напрямую взаимодействуя с узлами кластера.

Основные материалы:

- Развёртывание YDB кластера вручную
- Обзор управления дисковой подсистемой кластера
- Использование YDB UI

Развёртывание YDB кластера вручную

Данный документ описывает способы развёртывания мультитенантного кластера YDB с распределённой архитектурой на нескольких физических или виртуальных серверах.

Основные материалы:

- Подготовка к развертыванию
- Развёртывание кластера с использованием конфигурации V1
- Развёртывание кластера с использованием конфигурации V2

Подготовка к развертыванию

Перед началом работы

Требования

Ознакомьтесь с системными требованиями и топологией кластера.

У вас должен быть SSH доступ на все сервера. Это необходимо для установки артефактов и запуска исполняемого файла YDB.

Сетевая конфигурация должна разрешать TCP соединения по следующим портам (по умолчанию, могут быть изменены настройками):

- 22: сервис SSH;
- 2135, 2136: GRPC для клиент-кластерного взаимодействия;
- 19001, 19002: Interconnect для внутрикластерного взаимодействия узлов;
- 8765, 8766: HTTP интерфейс YDB UI.
- 9092, 9093: порты для работы по Kafka API.

При размещении нескольких динамических узлов на одном сервере потребуются отдельные порты для gRPC, Interconnect, HTTP интерфейса и Kafka API каждого динамического узла в рамках сервера.

Убедитесь в том, что системные часы на всех серверах в составе кластера синхронизированы с помощью инструментов `ntpd` или `chrony`. Желательно использовать единый источник времени для всех серверов кластера, чтобы обеспечить одинаковую обработку секунд координации (leap seconds).

Если применяемый на серверах кластера тип Linux использует `syslogd` для логирования, необходимо настроить ротацию файлов лога с использованием инструмента `logrotate` или его аналогов. Сервисы YDB могут генерировать значительный объем системных логов, в особенности при повышении уровня логирования для диагностических целей, поэтому важно включить ротацию файлов системного лога для исключения ситуаций переполнения файловой системы `/var`.

Выберите серверы и диски, которые будут использоваться для хранения данных:

- Используйте схему отказоустойчивости `block-4-2` для развертывания кластера в одной зоне доступности (AZ), задействуя не менее 8 серверов. Данная схема позволяет переживать отказ 2 серверов.
- Используйте схему отказоустойчивости `mirror-3-dc` для развертывания кластера в трех зонах доступности (AZ), задействуя не менее 9 серверов. Данная схема позволяет переживать отказ 1 AZ и 1 сервера в другой AZ. Количество задействованных серверов в каждой AZ должно быть одинаковым.

> **Примечание**
>
> Запускайте каждый статический узел (узел хранения данных) на отдельном сервере. Возможно совмещение статических и динамических узлов на одном сервере, а также размещение на одном сервере нескольких динамических узлов при наличии достаточных вычислительных ресурсов.

Подробнее о требованиях к оборудованию описано в разделе Системные требования и рекомендации для YDB.

Подготовка ключей и сертификатов TLS

Защита трафика и проверка подлинности серверных узлов YDB осуществляется с использованием протокола TLS. Перед установкой кластера необходимо спланировать состав серверов, определиться со схемой именования узлов и конкретными именами, и подготовить ключи и сертификаты TLS.

Вы можете использовать существующие или сгенерировать новые сертификаты. Следующие файлы ключей и сертификатов TLS должны быть подготовлены в формате PEM:

- `ca.crt` - сертификат центра регистрации (Certification Authority, CA), которым подписаны остальные сертификаты TLS (одинаковые файлы на всех узлах кластера);
- `node.key` - секретные ключи TLS для каждого из узлов кластера (свой ключ на каждый сервер кластера);
- `node.crt` - сертификаты TLS для каждого из узлов кластера (соответствующий ключу сертификат);
- `web.pem` - конкатенация секретного ключа узла, сертификата узла и сертификата центра регистрации для работы HTTP интерфейса мониторинга (свой файл на каждый сервер кластера).

Необходимые параметры формирования сертификатов определяются политикой организации. Обычно сертификаты и ключи для YDB формируются со следующими параметрами:

- ключи RSA длиною 2048 или 4096 бит;
- алгоритм подписи сертификатов SHA-256 с шифрованием RSA;
- срок действия сертификатов узлов не менее 1 года;
- срок действия сертификата центра регистрации не менее 3 лет.

Необходимо, чтобы сертификат центра регистрации был помечен соответствующим образом: должен быть установлен признак CA, а также включены виды использования "Digital Signature, Non Repudiation, Key Encipherment, Certificate Sign".

Для сертификатов узлов важно соответствие фактического имени хоста (или имён хостов) значениям, указанным в поле "Subject Alternative Name". Для сертификатов должны быть включены виды использования "Digital Signature, Key Encipherment" и расширенные виды использования "TLS Web Server Authentication, TLS Web Client Authentication". Необходимо, чтобы сертификаты узлов поддерживали как серверную, так и клиентскую аутентификацию (опция `extendedKeyUsage = serverAuth,clientAuth` в настройках OpenSSL).

Для пакетной генерации или обновления сертификатов кластера YDB с помощью программного обеспечения OpenSSL можно воспользоваться примером скрипта, размещённым в репозитории YDB на GitHub. Скрипт позволяет автоматически сформировать необходимые файлы ключей и сертификатов для всего набора узлов кластера за одну операцию, облегчая подготовку к установке.

Создайте системного пользователя и группу, от имени которых будет работать YDB

На каждом сервере, где будет запущен YDB, выполните:

```
sudo groupadd ydb
sudo useradd -m -g ydb ydb
```

Для того, чтобы сервис YDB имел доступ к блочным дискам для работы, необходимо добавить пользователя, под которым будут запущены процессы YDB, в группу `disk`:

```
sudo usermod -aG disk ydb
```

Настройте лимиты файловых дескрипторов

Для корректной работы YDB, особенно при использовании спиллинга в многоузловых кластерах, рекомендуется увеличить лимит на количество одновременно открытых файловых дескрипторов.

Для изменения лимита файловых дескрипторов добавьте следующие строки в файл `/etc/security/limits.conf`:

```
ydb soft nofile 10000
ydb hard nofile 10000
```

Где `ydb` — имя пользователя, под которым запускается `ydbd`.

После изменения файла необходимо перезагрузить систему или заново залогиниться для применения новых лимитов.

> **Примечание**
>
> Для получения дополнительной информации о конфигурации спиллинга и его связи с файловыми дескрипторами см. раздел «Конфигурация спиллинга».

Установите программное обеспечение YDB на каждом сервере

Скачайте и распакуйте архив с исполняемым файлом `ydbd` и необходимыми для работы YDB библиотеками

OSS

```
mkdir ydbd-stable-linux-amd64
curl -L <binaries_url> | tar -xz --strip-component=1 -C ydbd-stable-linux-amd64
```

Enterprise

```
mkdir ydbd-stable-linux-amd64
curl -L <binaries_url> | tar -xJ --strip-component=1 -C ydbd-stable-linux-amd64
```

где `binaries_url` — ссылка на архив нужной вам версии со страницы загрузок.

В командах выше: `-xz` — для архива `.tar.gz` (OSS), `-xJ` — для `.tar.xz` (Enterprise).

Создайте на сервере директорию

```
sudo mkdir -p  /opt/ydb
```

Скопируйте исполняемый файл и библиотеки в соответствующие директории

```
sudo cp -iR ydbd-stable-linux-amd64/bin /opt/ydb/
sudo cp -iR ydbd-stable-linux-amd64/lib /opt/ydb/
```

Установите владельца файлов и каталогов

```
sudo chown -R root:bin /opt/ydb
```

Подготовьте и очистите диски на каждом сервере

Для эффективной работы YDB рекомендуется использовать физические (не виртуальные) диски объемом более 800 ГБ как блочные устройства.

Минимальный объем диска должен быть не менее 80 ГБ, при меньшем объеме узел YDB не сможет использовать устройство. Корректная и бесперебойная работа с дисками минимального объема не гарантируется. Использовать такие диски рекомендуется исключительно в ознакомительных целях.

> **Важно**
>
> Конфигурации с дисками объемом меньше 800 ГБ или с любыми видами виртуализации системы хранения нельзя использовать для сервисов, находящихся в промышленной эксплуатации, а также для тестирования производительности системы.
>
> Мы не рекомендуем использовать для хранения данных YDB диски, которые используются другими процессами (в том числе операционной системой).

Получить список блочных устройств на сервере можно командой `lsblk`. Пример вывода:

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
loop0    7:0    0  63.3M  1 loop /snap/core20/1822
...
vda    252:0    0    40G  0 disk
├─vda1 252:1    0     1M  0 part
└─vda2 252:2    0    40G  0 part /
vdb    252:16   0   186G  0 disk
└─vdb1 252:17   0   186G  0 part
```

Названия блочных устройств зависят от настроек операционной системы, заданных базовым образом или настроенных вручную. Обычно имена устройств состоят из трех частей:

- фиксированный префикс, указывающий на тип устройства — например, `vd` в `vdb` (virtio), `sd` в `sda` (SATA/SCSI) или `nvme` в `nvme0n1` (NVMe);
- идентификатор устройства — буква или число (`b` в `vdb`, `a` в `sda`, `0n1` в `nvme0n1`);
- номер раздела — обычно число (`1` в `vdb1`).

В выводе `lsblk` выше системный диск — `vda`, диск для данных YDB — `vdb`. В командах ниже используется `/dev/vdb`; на вашем сервере подставьте путь к целому диску без номера раздела (например, `/dev/sda` или `/dev/nvme0n1`).

1. Создайте разделы на выбранных дисках:

   Внимание

   Следующая операция удалит все разделы на указанном диске! Убедитесь, что вы указали диск, на котором нет других данных!

   ```
   DISK=/dev/vdb
   sudo parted ${DISK} mklabel gpt -s
   sudo parted -a optimal ${DISK} mkpart primary 0% 100%
   sudo parted ${DISK} name 1 ydb_disk_ssd_01
   sudo partx --u ${DISK}
   ```

   Выполните команду `ls -l /dev/disk/by-partlabel/`, чтобы убедиться что в системе появился диск с меткой `/dev/disk/by-partlabel/ydb_disk_ssd_01`.

   Если вы планируете использовать более одного диска на каждом сервере, укажите для каждого свою уникальную метку вместо `ydb_disk_ssd_01`. Метки дисков должны быть уникальны в рамках каждого сервера, и используются в конфигурационных файлах, как показано в последующих инструкциях.

   Для упрощения последующей настройки удобно использовать одинаковые метки дисков на серверах кластера, имеющих идентичную конфигурацию дисков.
2. Очистите диск встроенной в исполняемый файл `ydbd` командой:

   Важно

   После выполнения команды данные на диске сотрутся.

   ```
   sudo LD_LIBRARY_PATH=/opt/ydb/lib /opt/ydb/bin/ydbd admin bs disk obliterate /dev/disk/by-partlabel/ydb_
   disk_ssd_01
   ```

   Проделайте данную операцию для каждого диска, который будет использоваться для хранения данных YDB.

Пример полной команды для разметки 3-х дисков

```
DISK=/dev/vdb
sudo parted ${DISK} mklabel gpt -s
sudo parted -a optimal ${DISK} mkpart primary 0% 100%
sudo parted ${DISK} name 1 ydb_disk_ssd_01
sudo partx --u ${DISK}
sleep 5
sudo LD_LIBRARY_PATH=/opt/ydb/lib /opt/ydb/bin/ydbd admin bs disk obliterate /dev/disk/by-partlabel/ydb_disk_
ssd_01

DISK=/dev/vdc
sudo parted ${DISK} mklabel gpt -s
sudo parted -a optimal ${DISK} mkpart primary 0% 100%
sudo parted ${DISK} name 1 ydb_disk_ssd_02
sudo partx --u ${DISK}
sleep 5
sudo LD_LIBRARY_PATH=/opt/ydb/lib /opt/ydb/bin/ydbd admin bs disk obliterate /dev/disk/by-partlabel/ydb_disk_
ssd_02

DISK=/dev/vdd
sudo parted ${DISK} mklabel gpt -s
sudo parted -a optimal ${DISK} mkpart primary 0% 100%
sudo parted ${DISK} name 1 ydb_disk_ssd_03
sudo partx --u ${DISK}
sleep 5
sudo LD_LIBRARY_PATH=/opt/ydb/lib /opt/ydb/bin/ydbd admin bs disk obliterate /dev/disk/by-partlabel/ydb_disk_
ssd_03
```

Проверьте подготовку дисков

Для проверки корректной разметки дисков выполните команду на каждом сервере кластера:

```
ls -al /dev/disk/by-partlabel/
```

В выводе команды должны быть созданные и размеченные вами диски

```
lrwxrwxrwx 1 root root    10 Nov 26 12:54 ydb_disk_ssd_01 -> ../../vdb1
lrwxrwxrwx 1 root root    10 Nov 26 12:54 ydb_disk_ssd_02 -> ../../vdc1
lrwxrwxrwx 1 root root    10 Nov 26 12:54 ydb_disk_ssd_03 -> ../../vdd1
```

После выполнения подготовительных мероприятий можно переходить к развёртыванию системы. Выберите инструкцию в соответствии с вашей конфигурацией:

- Развёртывание кластера с использованием конфигурации V1
- Развёртывание кластера с использованием конфигурации V2

Развёртывание кластера с использованием конфигурации V1

Подготовьте окружение

Перед развёртыванием системы обязательно выполните подготовительные действия. Ознакомьтесь с документом Подготовка к развертыванию.

Подготовьте конфигурационные файлы

Подготовьте конфигурационный файл YDB в зависимости от выбранной вами топологии (см. выбор топологии). Примеры для каждой поддерживаемой топологии приведены ниже во вкладках — выберите и используйте подходящий для вашего случая. Также при необходимости включить работу по Kafka API с топиками, добавьте в конфигурационный файл секцию kafka_proxy_config (см. конфигурирование Kafka API).

mirror-3-dc-3nodes

```
static_erasure: mirror-3-dc
host_configs:
- drive:
  - path: /dev/disk/by-partlabel/ydb_disk_ssd_01
    type: SSD
  - path: /dev/disk/by-partlabel/ydb_disk_ssd_02
    type: SSD
  - path: /dev/disk/by-partlabel/ydb_disk_ssd_03
    type: SSD
  host_config_id: 1
hosts:
- host: static-node-1.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 1
    data_center: 'zone-a'
    rack: '1'
- host: static-node-2.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 2
    data_center: 'zone-b'
    rack: '2'
- host: static-node-3.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 3
    data_center: 'zone-c'
    rack: '3'
domains_config:
  security_config:
    enforce_user_token_requirement: true
    monitoring_allowed_sids:
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
    administration_allowed_sids:
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
    viewer_allowed_sids:
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
    register_dynamic_node_allowed_sids:
    - databaseNodes@cert
    - root@builtin
  domain:
  - name: Root
    storage_pool_types:
    - kind: ssd
      pool_config:
        box_id: 1
        erasure_species: mirror-3-dc
        kind: ssd
        geometry:
          realm_level_begin: 10
          realm_level_end: 20
          domain_level_begin: 10
          domain_level_end: 256
        pdisk_filter:
        - property:
          - type: SSD
        vdisk_kind: Default
  state_storage:
  - ring:
      node: [1, 2, 3]
      nto_select: 3
    ssid: 1
actor_system_config:
  executor:
  - name: System
    threads: 2
    type: BASIC
  - name: User
```

```
    threads: 3
    type: BASIC
  - name: Batch
    threads: 2
    type: BASIC
  - name: IO
    threads: 1
    time_per_mailbox_micro_secs: 100
    type: IO
  - name: IC
    spin_threshold: 10
    threads: 1
    time_per_mailbox_micro_secs: 100
    type: BASIC
  scheduler:
    progress_threshold: 10000
    resolution: 256
    spin_threshold: 0
blob_storage_config:
  service_set:
    groups:
    - erasure_species: mirror-3-dc
      rings:
      - fail_domains:
        - vdisk_locations:
          - node_id: static-node-1.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_01
        - vdisk_locations:
          - node_id: static-node-1.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_02
        - vdisk_locations:
          - node_id: static-node-1.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_03
      - fail_domains:
        - vdisk_locations:
          - node_id: static-node-2.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_01
        - vdisk_locations:
          - node_id: static-node-2.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_02
        - vdisk_locations:
          - node_id: static-node-2.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_03
      - fail_domains:
        - vdisk_locations:
          - node_id: static-node-3.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_01
        - vdisk_locations:
          - node_id: static-node-3.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_02
        - vdisk_locations:
          - node_id: static-node-3.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_03
channel_profile_config:
  profile:
  - channel:
    - erasure_species: mirror-3-dc
      pdisk_category: 0
      storage_pool_kind: ssd
    - erasure_species: mirror-3-dc
      pdisk_category: 0
      storage_pool_kind: ssd
    - erasure_species: mirror-3-dc
      pdisk_category: 0
      storage_pool_kind: ssd
    profile_id: 0
interconnect_config:
    start_tcp: true
    encryption_mode: OPTIONAL
    path_to_certificate_file: "/opt/ydb/certs/node.crt"
    path_to_private_key_file: "/opt/ydb/certs/node.key"
    path_to_ca_file: "/opt/ydb/certs/ca.crt"
grpc_config:
    cert: "/opt/ydb/certs/node.crt"
    key: "/opt/ydb/certs/node.key"
    ca: "/opt/ydb/certs/ca.crt"
    services_enabled:
    - legacy
client_certificate_authorization:
  request_client_certificate: true
  client_certificate_definitions:
    - member_groups: ["databaseNodes@cert"]
```

```
      subject_terms:
      - short_name: "O"
        values: ["YDB"]
```

mirror-3-dc-9nodes

```
static_erasure: mirror-3-dc
host_configs:
- drive:
  - path: /dev/disk/by-partlabel/ydb_disk_ssd_01
    type: SSD
  - path: /dev/disk/by-partlabel/ydb_disk_ssd_02
    type: SSD
  host_config_id: 1
hosts:
- host: static-node-1.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 1
    data_center: 'zone-a'
    rack: '1'
- host: static-node-2.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 2
    data_center: 'zone-a'
    rack: '2'
- host: static-node-3.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 3
    data_center: 'zone-a'
    rack: '3'
- host: static-node-4.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 4
    data_center: 'zone-b'
    rack: '4'
- host: static-node-5.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 5
    data_center: 'zone-b'
    rack: '5'
- host: static-node-6.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 6
    data_center: 'zone-b'
    rack: '6'
- host: static-node-7.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 7
    data_center: 'zone-c'
    rack: '7'
- host: static-node-8.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 8
    data_center: 'zone-c'
    rack: '8'
- host: static-node-9.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 9
    data_center: 'zone-c'
    rack: '9'
domains_config:
  security_config:
    enforce_user_token_requirement: true
    monitoring_allowed_sids:
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
    administration_allowed_sids:
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
    viewer_allowed_sids:
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
    register_dynamic_node_allowed_sids:
    - databaseNodes@cert
    - root@builtin
  domain:
  - name: Root
    storage_pool_types:
    - kind: ssd
```

```
      pool_config:
        box_id: 1
        erasure_species: mirror-3-dc
        kind: ssd
        pdisk_filter:
        - property:
          - type: SSD
        vdisk_kind: Default
  state_storage:
  - ring:
      node: [1, 2, 3, 4, 5, 6, 7, 8, 9]
      nto_select: 9
    ssid: 1
actor_system_config:
  executor:
  - name: System
    spin_threshold: 0
    threads: 2
    type: BASIC
  - name: User
    spin_threshold: 0
    threads: 3
    type: BASIC
  - name: Batch
    spin_threshold: 0
    threads: 2
    type: BASIC
  - name: IO
    threads: 1
    time_per_mailbox_micro_secs: 100
    type: IO
  - name: IC
    spin_threshold: 10
    threads: 1
    time_per_mailbox_micro_secs: 100
    type: BASIC
  scheduler:
    progress_threshold: 10000
    resolution: 256
    spin_threshold: 0
blob_storage_config:
  service_set:
    groups:
    - erasure_species: mirror-3-dc
      rings:
      - fail_domains:
        - vdisk_locations:
          - node_id: static-node-1.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_01
        - vdisk_locations:
          - node_id: static-node-2.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_01
        - vdisk_locations:
          - node_id: static-node-3.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_01
      - fail_domains:
        - vdisk_locations:
          - node_id: static-node-4.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_01
        - vdisk_locations:
          - node_id: static-node-5.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_01
        - vdisk_locations:
          - node_id: static-node-6.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_01
      - fail_domains:
        - vdisk_locations:
          - node_id: static-node-7.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_01
        - vdisk_locations:
          - node_id: static-node-8.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_01
        - vdisk_locations:
          - node_id: static-node-9.ydb-cluster.com
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_01
channel_profile_config:
  profile:
  - channel:
    - erasure_species: mirror-3-dc
      pdisk_category: 1
      storage_pool_kind: ssd
    - erasure_species: mirror-3-dc
```

```
      pdisk_category: 1
      storage_pool_kind: ssd
    - erasure_species: mirror-3-dc
      pdisk_category: 1
      storage_pool_kind: ssd
    profile_id: 0
interconnect_config:
    start_tcp: true
    encryption_mode: OPTIONAL
    path_to_certificate_file: "/opt/ydb/certs/node.crt"
    path_to_private_key_file: "/opt/ydb/certs/node.key"
    path_to_ca_file: "/opt/ydb/certs/ca.crt"
grpc_config:
    cert: "/opt/ydb/certs/node.crt"
    key: "/opt/ydb/certs/node.key"
    ca: "/opt/ydb/certs/ca.crt"
    services_enabled:
    - legacy
client_certificate_authorization:
  request_client_certificate: true
  client_certificate_definitions:
      - member_groups: ["databaseNodes@cert"]
        subject_terms:
        - short_name: "O"
          values: ["YDB"]
```

block-4-2

```
static_erasure: block-4-2
host_configs:
- drive:
  - path: /dev/disk/by-partlabel/ydb_disk_ssd_01
    type: SSD
  host_config_id: 1
hosts:
- host: static-node-1.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 1
    data_center: 'zone-a'
    rack: '1'
- host: static-node-2.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 2
    data_center: 'zone-a'
    rack: '2'
- host: static-node-3.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 3
    data_center: 'zone-a'
    rack: '3'
- host: static-node-4.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 4
    data_center: 'zone-a'
    rack: '4'
- host: static-node-5.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 5
    data_center: 'zone-a'
    rack: '5'
- host: static-node-6.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 6
    data_center: 'zone-a'
    rack: '6'
- host: static-node-7.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 7
    data_center: 'zone-a'
    rack: '7'
- host: static-node-8.ydb-cluster.com
  host_config_id: 1
  walle_location:
    body: 8
    data_center: 'zone-a'
    rack: '8'
domains_config:
  security_config:
    enforce_user_token_requirement: true
    monitoring_allowed_sids:
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
    administration_allowed_sids:
```

```
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
    viewer_allowed_sids:
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
    register_dynamic_node_allowed_sids:
    - databaseNodes@cert
    - root@builtin
  domain:
  - name: Root
    storage_pool_types:
    - kind: ssd
      pool_config:
        box_id: 1
        erasure_species: block-4-2
        kind: ssd
        pdisk_filter:
        - property:
          - type: SSD
        vdisk_kind: Default
  state_storage:
  - ring:
      node: [1, 2, 3, 4, 5, 6, 7, 8]
      nto_select: 5
    ssid: 1
actor_system_config:
  executor:
  - name: System
    threads: 2
    type: BASIC
  - name: User
    threads: 3
    type: BASIC
  - name: Batch
    threads: 2
    type: BASIC
  - name: IO
    threads: 1
    time_per_mailbox_micro_secs: 100
    type: IO
  - name: IC
    spin_threshold: 10
    threads: 1
    time_per_mailbox_micro_secs: 100
    type: BASIC
  scheduler:
    progress_threshold: 10000
    resolution: 256
    spin_threshold: 0
blob_storage_config:
  service_set:
    groups:
    - erasure_species: block-4-2
      rings:
      - fail_domains:
        - vdisk_locations:
          - node_id: "static-node-1.ydb-cluster.com"
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_01
      - fail_domains:
        - vdisk_locations:
          - node_id: "static-node-2.ydb-cluster.com"
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_01
      - fail_domains:
        - vdisk_locations:
          - node_id: "static-node-3.ydb-cluster.com"
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_01
      - fail_domains:
        - vdisk_locations:
          - node_id: "static-node-4.ydb-cluster.com"
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_01
      - fail_domains:
        - vdisk_locations:
          - node_id: "static-node-5.ydb-cluster.com"
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_01
      - fail_domains:
        - vdisk_locations:
          - node_id: "static-node-6.ydb-cluster.com"
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_01
      - fail_domains:
        - vdisk_locations:
          - node_id: "static-node-7.ydb-cluster.com"
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_01
```

```
      - fail_domains:
        - vdisk_locations:
          - node_id: "static-node-8.ydb-cluster.com"
            pdisk_category: SSD
            path: /dev/disk/by-partlabel/ydb_disk_ssd_01
channel_profile_config:
  profile:
  - channel:
    - erasure_species: block-4-2
      pdisk_category: 0
      storage_pool_kind: ssd
    - erasure_species: block-4-2
      pdisk_category: 0
      storage_pool_kind: ssd
    - erasure_species: block-4-2
      pdisk_category: 0
      storage_pool_kind: ssd
    profile_id: 0
interconnect_config:
    start_tcp: true
    encryption_mode: OPTIONAL
    path_to_certificate_file: "/opt/ydb/certs/node.crt"
    path_to_private_key_file: "/opt/ydb/certs/node.key"
    path_to_ca_file: "/opt/ydb/certs/ca.crt"
grpc_config:
    cert: "/opt/ydb/certs/node.crt"
    key: "/opt/ydb/certs/node.key"
    ca: "/opt/ydb/certs/ca.crt"
    services_enabled:
    - legacy
client_certificate_authorization:
  request_client_certificate: true
  client_certificate_definitions:
    - member_groups: ["databaseNodes@cert"]
      subject_terms:
      - short_name: "O"
        values: ["YDB"]
```

Для ускорения и упрощения первичного развёртывания YDB конфигурационный файл уже содержит большинство настроек для установки кластера. Достаточно заменить стандартные хосты FQDN на актуальные в разделах `hosts` и `blob_storage_config`.

- Раздел `hosts`:

```
...
hosts:
  - host: static-node-1.ydb-cluster.com #FQDN BM
    host_config_id: 1
    walle_location:
      body: 1
      data_center: 'zone-a'
      rack: '1'
...
```

- Раздел `blob_storage_config`:

```
...
- fail_domains:
  - vdisk_locations:
    - node_id: static-node-1.ydb-cluster.com #FQDN BM
      pdisk_category: SSD
      path: /dev/disk/by-partlabel/ydb_disk_1
...
```

Остальные секции и настройки конфигурационного файла остаются без изменений.

Сохраните конфигурационный файл YDB под именем `/opt/ydb/cfg/config.yaml` на каждом сервере кластера.

Более подробная информация по созданию файла конфигурации приведена в разделе Параметры конфигурации кластера.

Скопируйте ключи и сертификаты TLS на каждый сервер

Подготовленные ключи и сертификаты TLS необходимо скопировать в защищённый каталог на каждом из узлов кластера YDB.

Подробности подготовки — в разделе Подготовка ключей и сертификатов TLS. Если сертификаты сгенерированы скриптом `tls_cert_gen`, возьмите `ca.crt` из каталога `CA/certs/YYYY-MM-DD_hh-mi-ss` (подставьте метку времени из вывода скрипта или выберите нужный каталог в `CA/certs/`), а `node.crt`, `node.key` и `web.pem` — из подкаталога с именем FQDN этого сервера в `ydb-ca-nodes.txt`. Перенесите эти четыре файла на соответствующий сервер кластера (например, через `scp`) в один локальный каталог.

Для административных команд в разделах Инициализируйте кластер, Создайте базу данных, Настройка учётных записей и Протестируйте работу с созданной базой в примерах используется путь `~/CA/certs/`. Чтобы не менять пути в командах, на каждом сервере создайте каталог `~/CA/certs/` в домашнем каталоге того пользователя, от имени которого вы выполняете эти команды, и поместите в него четыре файла: `ca.crt`, `node.crt`, `node.key` и `web.pem`.

Каталог `/opt/ydb/certs/` (см. ниже) нужен для запуска `ydbd` от пользователя `ydb`; `~/CA/certs/` — для административных команд из последующих разделов.

На каждом сервере выполните команды ниже: сначала разместите файлы в `~/CA/certs/`, затем скопируйте их в `/opt/ydb/certs/`.

```
mkdir -p ~/CA/certs
cp -v ca.crt ~/CA/certs/
cp -v node.crt ~/CA/certs/
cp -v node.key ~/CA/certs/
cp -v web.pem ~/CA/certs/
sudo mkdir -p /opt/ydb/certs
sudo cp -v ~/CA/certs/ca.crt /opt/ydb/certs/
sudo cp -v ~/CA/certs/node.crt /opt/ydb/certs/
sudo cp -v ~/CA/certs/node.key /opt/ydb/certs/
sudo cp -v ~/CA/certs/web.pem /opt/ydb/certs/
sudo chown -R ydb:ydb /opt/ydb/certs
sudo chmod 700 /opt/ydb/certs
```

Перед запуском `ydbd` убедитесь, что процессы от пользователя `ydb` смогут прочитать сертификат мониторинга (иначе при `--mon-cert /opt/ydb/certs/web.pem` будет ошибка `Permission denied`):

```
sudo -u ydb test -r /opt/ydb/certs/web.pem
```

При успешной проверке команда завершается **без вывода**, код возврата `0` (проверить: `echo $?` → `0`). Если файл отсутствует или права настроены неверно — код `1`; повторите `chown` / `chmod` из блока выше и убедитесь, что на этот узел скопирован `web.pem` этого хоста.

Запустите статические узлы

Вручную

Запустите сервис хранения данных YDB на каждом статическом узле кластера. Поочередно выполните команды ниже. Не вставляйте все строки разом вместе с `sudo su - ydb` — иначе следующие команды могут выполниться не от пользователя `ydb`.

После шага 1 дождитесь приглашения `ydb@...$`.

На шаге 4 дождитесь вывода `ydbd` в этом же терминале.

1. Переключитесь на пользователя `ydb`:

   ```
   sudo su - ydb
   ```

2. Перейдите в каталог установки:

   ```
   cd /opt/ydb
   ```

3. Укажите путь к библиотекам:

   ```
   export LD_LIBRARY_PATH=/opt/ydb/lib
   ```

4. Запустите статический узел:

   ```
   /opt/ydb/bin/ydbd server --log-level 3 --syslog --tcp --yaml-config /opt/ydb/cfg/config.yaml --grpcs-por
   t 2135 --ic-port 19001 --mon-port 8765 --mon-cert /opt/ydb/certs/web.pem --node static
   ```

   При успешном старте в выводе появится, в частности, `Determined node ID: ...`. Если проверка `web.pem` не пройдена, здесь будет ошибка `Permission denied` для `/opt/ydb/certs/web.pem`.

С использованием systemd

Создайте на каждом сервере, где будет размещен статический узел кластера, конфигурационный файл systemd `/etc/systemd/system/ydbd-storage.service` по приведенному ниже образцу. Образец файла также можно скачать из репозитория.

```
[Unit]
Description=YDB storage node
After=network-online.target rc-local.service
Wants=network-online.target
StartLimitInterval=10
StartLimitBurst=15

[Service]
Restart=always
RestartSec=1
User=ydb
PermissionsStartOnly=true
StandardOutput=syslog
StandardError=syslog
SyslogIdentifier=ydbd
SyslogFacility=daemon
SyslogLevel=err
Environment=LD_LIBRARY_PATH=/opt/ydb/lib
ExecStart=/opt/ydb/bin/ydbd server --log-level 3 --syslog --tcp \
    --yaml-config  /opt/ydb/cfg/config.yaml \
    --grpcs-port 2135 --ic-port 19001 --mon-port 8765 --kafka-port 9092 \
    --mon-cert /opt/ydb/certs/web.pem --node static
```

```
LimitNOFILE=65536
LimitCORE=0
LimitMEMLOCK=3221225472

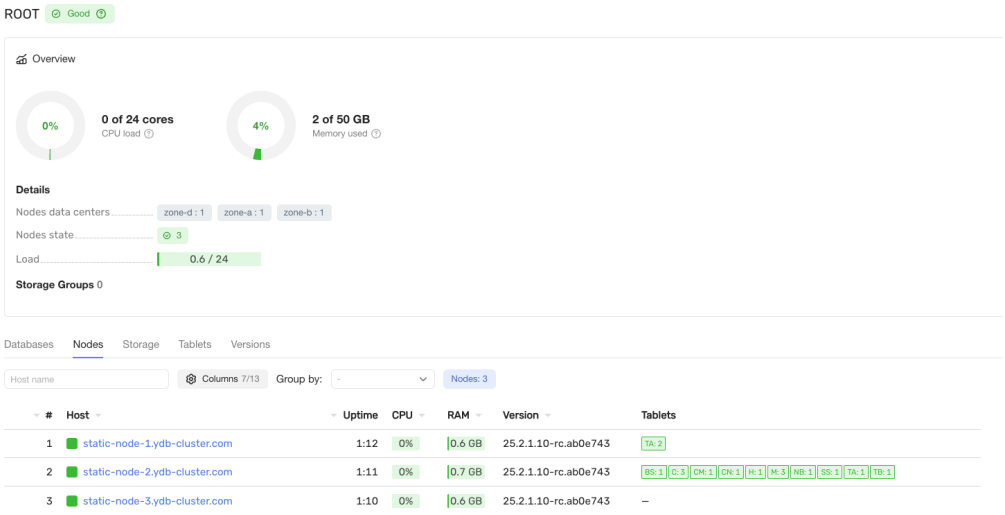
[Install]
WantedBy=multi-user.target
```

Запустите сервис на каждом статическом узле YDB:

```
sudo systemctl start ydbd-storage
```

После запуска статических узлов проверьте их работоспособность через YDB UI:

1. Откройте в браузере адрес `https://<static-node.ydb.tech>:8765`, где `<static-node.ydb.tech>` — FQDN сервера, на котором запущен любой статический узел;
2. Если браузер запросит логин и пароль, войдите под учётной записью `root` с пустым паролем — на этом этапе развёртывания пароль для `root` ещё не задан;
3. Перейдите на вкладку **Nodes**;
4. Убедитесь, что в списке отображаются все 3 статических узла.



Инициализируйте кластер

Операция инициализации кластера осуществляет настройку набора статических узлов, перечисленных в конфигурационном файле кластера, для хранения данных YDB.

Для инициализации кластера потребуется файл сертификата центра регистрации `ca.crt` из каталога `~/CA/certs/` (см. раздел Скопируйте ключи и сертификаты TLS на каждый сервер).

На одном из серверов хранения в составе кластера выполните команды:

Сначала получите авторизационный токен для регистрации запросов. Для этого выполните приведённую ниже команду.

```
/opt/ydb/bin/ydb --ca-file ~/CA/certs/ca.crt -e grpcs://`hostname -f`:2135 -d /Root --user root --no-password
auth get-token -f > auth_token
```

Инициализируйте кластер, используя полученный токен

```
export LD_LIBRARY_PATH=/opt/ydb/lib
/opt/ydb/bin/ydbd --ca-file ~/CA/certs/ca.crt -s grpcs://`hostname -f`:2135 -f auth_token \
    admin blobstorage config init --yaml-file  /opt/ydb/cfg/config.yaml
echo $?
```

При успешном выполнении инициализации кластера выведенный на экран код завершения команды инициализации кластера должен быть нулевым.

Создайте базу данных

Для работы со строковыми или колоночными таблицами необходимо создать как минимум одну базу данных и запустить процесс или процессы, обслуживающие эту базу данных (динамические узлы).

Для выполнения административной команды создания базы данных потребуется файл `ca.crt` из каталога `~/CA/certs/` (см. раздел Скопируйте ключи и сертификаты TLS на каждый сервер).

При создании базы данных устанавливается первоначальное количество используемых групп хранения, определяющее доступную пропускную способность ввода-вывода и максимальную емкость хранения. Количество групп хранения может быть при необходимости увеличено после создания базы данных.

На одном из серверов хранения в составе кластера выполните команды:

```
export LD_LIBRARY_PATH=/opt/ydb/lib
/opt/ydb/bin/ydbd --ca-file ~/CA/certs/ca.crt -s grpcs://`hostname -f`:2135 -f auth_token \
    admin database /Root/testdb create ssd:8
echo $?
```

При успешном создании базы данных, выведенный на экран код завершения команды должен быть нулевым.

В приведенном выше примере команд используются следующие параметры:

- `/Root` - имя корневого домена, сгенерированного автоматически при инициализации кластера;
- `testdb` - имя создаваемой базы данных;
- `ssd:8` - задает пул хранения для базы данных и количество групп в нем. Имя пула (`ssd`) должно соответствовать типу диска, указанному в конфигурации кластера (например, в `default_disk_type`), и является регистронезависимым. Число после двоеточия — это количество выделяемых групп хранения.

Запустите динамические узлы

Первый динамический узел запустите на одном из серверов кластера, где уже подготовлены каталоги `/opt/ydb/certs` и `/opt/ydb/cfg` и работает статический узел. Совмещение статического и динамического узла на одном сервере допустимо (см. Требования).

Для топологий из примера (`mirror-3-dc-3nodes` — три сервера, `mirror-3-dc-9nodes` — девять) для отказоустойчивости и масштабирования запустите дополнительные динамические узлы на тех же серверах, что и статические — по одному на каждом из оставшихся хостов.

Вручную

Запустите динамический узел YDB для базы `/Root/testdb`. Поочередно выполните команды ниже. Не вставляйте все строки разом вместе с `sudo su - ydb` — иначе следующие команды могут выполниться не от пользователя `ydb`.

После шага 1 дождитесь приглашения `ydb@...$`.

На шаге 4 дождитесь вывода `ydbd` в этом же терминале. В команде ниже замените `<ydb-static-node1>`, `<ydb-static-node2>`, `<ydb-static-node3>` на FQDN трёх любых серверов со статическими узлами.

1. Переключитесь на пользователя `ydb`:

   ```
   sudo su - ydb
   ```

2. Перейдите в каталог установки:

   ```
   cd /opt/ydb
   ```

3. Укажите путь к библиотекам:

   ```
   export LD_LIBRARY_PATH=/opt/ydb/lib
   ```

4. Запустите динамический узел:

   ```
   /opt/ydb/bin/ydbd server --grpcs-port 2136 --grpc-ca /opt/ydb/certs/ca.crt --ic-port 19002 --ca /opt/yd
   b/certs/ca.crt --mon-port 8766 --mon-cert /opt/ydb/certs/web.pem --yaml-config /opt/ydb/cfg/config.yaml
   --tenant /Root/testdb --grpc-cert /opt/ydb/certs/node.crt --grpc-key /opt/ydb/certs/node.key --node-brok
   er grpcs://<ydb-static-node1>:2135 --node-broker grpcs://<ydb-static-node2>:2135 --node-broker grpcs://<
   ydb-static-node3>:2135
   ```

   Если проверка `web.pem` не пройдена, будет ошибка `Permission denied` для `/opt/ydb/certs/web.pem`.

С использованием systemd

Создайте конфигурационный файл systemd `/etc/systemd/system/ydbd-testdb.service` по приведенному ниже образцу. Образец файла также можно скачать из репозитория.

```
[Unit]
Description=YDB testdb dynamic node
After=network-online.target rc-local.service
Wants=network-online.target
StartLimitInterval=10
StartLimitBurst=15

[Service]
Restart=always
RestartSec=1
User=ydb
PermissionsStartOnly=true
StandardOutput=syslog
StandardError=syslog
SyslogIdentifier=ydbd
SyslogFacility=daemon
SyslogLevel=err
Environment=LD_LIBRARY_PATH=/opt/ydb/lib
ExecStart=/opt/ydb/bin/ydbd server \
    --grpcs-port 2136 --grpc-ca /opt/ydb/certs/ca.crt \
    --ic-port 19002 --ca /opt/ydb/certs/ca.crt \
    --mon-port 8766 --mon-cert /opt/ydb/certs/web.pem \
    --kafka-port 9093 \
    --yaml-config  /opt/ydb/cfg/config.yaml \
    --tenant /Root/testdb \
    --grpc-cert /opt/ydb/certs/node.crt \
    --grpc-key /opt/ydb/certs/node.key \
    --node-broker grpcs://<ydb-static-node1>:2135 \
    --node-broker grpcs://<ydb-static-node2>:2135 \
    --node-broker grpcs://<ydb-static-node3>:2135
```

```
LimitNOFILE=65536
LimitCORE=0
LimitMEMLOCK=32212254720

[Install]
WantedBy=multi-user.target
```

В примере команды выше `<ydb-static-node1>`, `<ydb-static-node2>`, `<ydb-static-node3>` — FQDN трех любых серверов, на которых запущены статические узлы кластера.

Запустите динамический узел YDB для базы `/Root/testdb`:

```
sudo systemctl start ydbd-testdb
```

Настройка учетных записей

1. Установите YDB CLI, как описано в документации.

В командах ниже `<node.ydb.tech>` — FQDN сервера с запущенным динамическим узлом базы `/Root/testdb` (порт gRPC `2136`). При выполнении команд по SSH на этом сервере можно указать endpoint: `grpcs://$(hostname -f):2136`.

1. Установите пароль для учетной записи `root`, используя полученный ранее токен:

   ```
   ydb --ca-file ~/CA/certs/ca.crt -e grpcs://<node.ydb.tech>:2136 -d /Root/testdb --token-file auth_token
   \
       yql -s 'ALTER USER root PASSWORD "passw0rd"'
   ```

   Вместо значения `passw0rd` подставьте необходимый пароль. Сохраните пароль в отдельный файл. Последующие команды от имени пользователя `root` будут выполняться с использованием пароля, передаваемого с помощью ключа `--password-file <path_to_user_password>`. Также пароль можно сохранить в профиле подключения, как описано в документации YDB CLI.
2. Создайте дополнительные учетные записи:

   ```
   ydb --ca-file ~/CA/certs/ca.crt -e grpcs://<node.ydb.tech>:2136 -d /Root/testdb --user root --password-f
   ile <path_to_root_pass_file> \
       yql -s 'CREATE USER user1 PASSWORD "passw0rd"'
   ```
3. Установите права учетных записей, включив их во встроенные группы:

   ```
   ydb --ca-file ~/CA/certs/ca.crt -e grpcs://<node.ydb.tech>:2136 -d /Root/testdb --user root --password-f
   ile <path_to_root_pass_file> \
       yql -s 'ALTER GROUP `ADMINS` ADD USER user1'
   ```

При выполнении команд создания учётных записей и присвоения групп клиент YDB CLI будет запрашивать ввод пароля пользователя `root`. Избежать многократного ввода пароля можно, создав профиль подключения, как описано в документации YDB CLI.

Протестируйте работу с созданной базой

1. Установите YDB CLI, как описано в документации.

В командах ниже `<node.ydb.tech>` — тот же FQDN динамического узла базы `/Root/testdb`, что в разделе Настройка учетных записей.

1. Создайте тестовую строковую (`test_row_table`) или колоночную таблицу (`test_column_table`):

Создание строковой таблицы

```
ydb --ca-file ~/CA/certs/ca.crt -e grpcs://<node.ydb.tech>:2136 -d /Root/testdb --user root \
    yql -s 'CREATE TABLE `testdir/test_row_table` (id Uint64, title Utf8, PRIMARY KEY (id));'
```

Создание колоночной таблицы

```
ydb --ca-file ~/CA/certs/ca.crt -e grpcs://<node.ydb.tech>:2136 -d /Root/testdb --user root \
    yql -s 'CREATE TABLE `testdir/test_column_table` (id Uint64 NOT NULL, title Utf8, PRIMARY KEY (id)) WITH
(STORE = COLUMN);'
```

Проверка доступа к YDB UI

Для проверки доступа к YDB UI достаточно открыть в Web-браузере страницу с адресом `https://<static-node.ydb.tech>:8765`, где `<static-node.ydb.tech>` — FQDN сервера, на котором запущен любой статический узел YDB.

В Web-браузере должно быть настроено доверие в отношении центра регистрации, выпустившего сертификаты для кластера YDB, в противном случае будет отображено предупреждение об использовании недоверенного сертификата.

Если в кластере включена аутентификация, в Web-браузере должен отобразиться запрос логина и пароля. После ввода верных данных аутентификации должна отобразиться начальная страница YDB UI. Описание доступных функций и пользовательского интерфейса приведено в разделе Использование YDB UI.

Примечание

Обычно для обеспечения доступа к YDB UI настраивают отказоустойчивый HTTP-балансировщик на базе программного обеспечения `haproxy`, `nginx` или аналогов. Детали настройки HTTP-балансировщика выходят за рамки стандартной инструкции по установке YDB.

Особенности установки YDB в незащищенном режиме

Важно

Мы не рекомендуем использовать незащищенный режим работы YDB ни при эксплуатации, ни при разработке приложений.

Описанная выше процедура установки предусматривает развёртывание YDB в стандартном защищенном режиме.

Незащищённый режим работы YDB предназначен для решения тестовых задач, преимущественно связанных с разработкой и тестированием программного обеспечения YDB. В незащищенном режиме:

- трафик между узлами кластера, а также между приложениями и кластером использует незашифрованные соединения;
- не используется аутентификация пользователей (включение аутентификации при отсутствии шифрования трафика не имеет смысла, поскольку логин и пароль в такой конфигурации передавались бы через сеть в открытом виде).

Установка YDB для работы в незащищенном режиме производится в порядке, описанном выше, со следующими исключениями:

1. При подготовке к установке не требуется формировать сертификаты и ключи TLS, и не выполняется копирование сертификатов и ключей на узлы кластера.
2. Из конфигурационных файлов кластерных узлов исключаются секции `security_config`, `interconnect_config` и `grpc_config`.
3. Используются упрощенный вариант команд запуска статических и динамических узлов кластера: исключаются опции с именами файлов сертификатов и ключей, используется протокол `grpc` вместо `grpcs` при указании точек подключения.
4. Пропускается ненужный в незащищенном режиме шаг по получению токена аутентификации перед выполнением инициализации кластера и созданием базы данных.
5. Команда инициализации кластера выполняется в следующей форме:

```
export LD_LIBRARY_PATH=/opt/ydb/lib
/opt/ydb/bin/ydbd admin blobstorage config init --yaml-file  /opt/ydb/cfg/config.yaml
echo $?
```

6. Команда создания базы данных выполняется в следующей форме:

```
export LD_LIBRARY_PATH=/opt/ydb/lib
/opt/ydb/bin/ydbd admin database /Root/testdb create ssd:1
```

7. При обращении к базе данных из YDB CLI и приложений используется протокол grpc вместо grpcs, и не используется аутентификация.

Развёртывание кластера с использованием конфигурации V2

> **Внимание**
>
> Эта статья посвящена кластерам YDB, в которых используется **конфигурация V2**. Данный способ конфигурирования пока является экспериментальным и доступен только для версий YDB начиная с v25.1. Для использования в продакшене мы рекомендуем выбирать конфигурацию V1 — она является основной и официально поддерживаемой для всех кластеров YDB.

Подготовьте окружение

Перед развёртыванием системы обязательно выполните подготовительные действия. Ознакомьтесь с документом Подготовка к развертыванию.

Подготовьте конфигурационные файлы

Подготовьте конфигурационный файл YDB:
Также при необходимости включить работу по Kafka API с топиками, добавьте в конфигурационный файл секцию kafka_proxy_config (см. конфигурирование Kafka API).

```
metadata:
  kind: MainConfig
  cluster: ""
  version: 0
config:
  erasure: mirror-3-dc
  fail_domain_type: disk
  self_management_config:
    enabled: true
  default_disk_type: SSD
  host_configs:
  - host_config_id: 1
    drive:
    - path: /dev/disk/by-partlabel/ydb_disk_ssd_01
      type: SSD
    - path: /dev/disk/by-partlabel/ydb_disk_ssd_02
      type: SSD
    - path: /dev/disk/by-partlabel/ydb_disk_ssd_03
      type: SSD
  hosts:
  - host: ydb-node-zone-a.local
    host_config_id: 1
    location:
      body: 1
      data_center: 'zone-a'
      rack: '1'
  - host: ydb-node-zone-b.local
    host_config_id: 1
    location:
      body: 2
      data_center: 'zone-b'
      rack: '2'
  - host: ydb-node-zone-c.local
    host_config_id: 1
    location:
      body: 3
      data_center: 'zone-c'
      rack: '3'
  actor_system_config:
    use_auto_config: true
    cpu_count: 8
  interconnect_config:
    start_tcp: true
    encryption_mode: OPTIONAL
    path_to_certificate_file: "/opt/ydb/certs/node.crt"
    path_to_private_key_file: "/opt/ydb/certs/node.key"
    path_to_ca_file: "/opt/ydb/certs/ca.crt"
  grpc_config:
    cert: "/opt/ydb/certs/node.crt"
    key: "/opt/ydb/certs/node.key"
    ca: "/opt/ydb/certs/ca.crt"
    services_enabled:
    - legacy
  security_config:
    enforce_user_token_requirement: true
    monitoring_allowed_sids:
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
    administration_allowed_sids:
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
    viewer_allowed_sids:
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
    register_dynamic_node_allowed_sids:
    - databaseNodes@cert
```

```
    - root@builtin
    bootstrap_allowed_sids:
    - "root"
    - "ADMINS"
    - "DATABASE-ADMINS"
  client_certificate_authorization:
    request_client_certificate: true
    client_certificate_definitions:
        - member_groups: ["ADMINS", "databaseNodes@cert"]
          subject_terms:
          - short_name: "O"
            values: ["YDB"]
```

Примечание

Приведенные выше настройки подготовлены для запуска YDB на 3х физических или виртуальных серверах с использованием режима работы кластера mirror-3-dc.

Для ускорения и упрощения первичного развёртывания YDB предлагаемый пример конфигурационного файла уже содержит большинство настроек для установки кластера. Достаточно заменить стандартные хосты FQDN в разделе `hosts` и пути к дискам на актуальные в разделе `hosts_configs`.

- Раздел `hosts`:

```
...
hosts:
- host: ydb-node-zone-a.local
  host_config_id: 1
  location:
    body: 1
    data_center: 'zone-a'
    rack: '1'
...
```

- Раздел `hosts_configs`:

```
...
host_configs:
- host_config_id: 1
  drive:
  - path: /dev/disk/by-partlabel/ydb_disk_ssd_01
    type: SSD
  - path: /dev/disk/by-partlabel/ydb_disk_ssd_02
    type: SSD
  - path: /dev/disk/by-partlabel/ydb_disk_ssd_03
    type: SSD
...
```

Остальные секции и настройки конфигурационного файла остаются без изменений.

Сохраните конфигурационный файл YDB под именем `/tmp/config.yaml` на каждом сервере кластера.

Более подробная информация по созданию файла конфигурации приведена в разделе Параметры конфигурации кластера.

Скопируйте ключи и сертификаты TLS на каждый сервер

Подготовленные ключи и сертификаты TLS необходимо скопировать в защищённый каталог на каждом из узлов кластера YDB.

Подробности подготовки — в разделе Подготовка ключей и сертификатов TLS. Если сертификаты сгенерированы скриптом `tls_cert_gen`, возьмите `ca.crt` из каталога `CA/certs/YYYY-MM-DD_hh-mi-ss` (подставьте метку времени из вывода скрипта или выберите нужный каталог в `CA/certs/`), а `node.crt`, `node.key` и `web.pem` — из подкаталога с именем FQDN этого сервера в `ydb-ca-nodes.txt`. Перенесите эти четыре файла на соответствующий сервер кластера (например, через `scp`) в один локальный каталог.

Для административных команд в разделах Инициализируйте кластер, Создайте базу данных, Настройка учётных записей и Протестируйте работу с созданной базой в примерах используется путь `~/CA/certs/`. Чтобы не менять пути в командах, на каждом сервере создайте каталог `~/CA/certs/` в домашнем каталоге того пользователя, от имени которого вы выполняете эти команды, и поместите в него четыре файла: `ca.crt`, `node.crt`, `node.key` и `web.pem`.

Каталог `/opt/ydb/certs/` (см. ниже) нужен для запуска `ydbd` от пользователя `ydb`; `~/CA/certs/` — для административных команд из последующих разделов.

На каждом сервере выполните команды ниже: сначала разместите файлы в `~/CA/certs/`, затем скопируйте их в `/opt/ydb/certs/`.

```
mkdir -p ~/CA/certs
cp -v ca.crt ~/CA/certs/
cp -v node.crt ~/CA/certs/
cp -v node.key ~/CA/certs/
cp -v web.pem ~/CA/certs/
sudo mkdir -p /opt/ydb/certs
sudo cp -v ~/CA/certs/ca.crt /opt/ydb/certs/
sudo cp -v ~/CA/certs/node.crt /opt/ydb/certs/
sudo cp -v ~/CA/certs/node.key /opt/ydb/certs/
sudo cp -v ~/CA/certs/web.pem /opt/ydb/certs/
```

```
sudo chown -R ydb:ydb /opt/ydb/certs
sudo chmod 700 /opt/ydb/certs
```

Перед запуском `ydbd` убедитесь, что процессы от пользователя `ydb` смогут прочитать сертификат мониторинга (иначе при `--mon-cert /opt/ydb/certs/web.pem` будет ошибка `Permission denied`):

```
sudo -u ydb test -r /opt/ydb/certs/web.pem
```

При успешной проверке команда завершается без вывода, код возврата `0` (проверить: `echo $?` → `0`). Если файл отсутствует или права настроены неверно — код `1`; повторите `chown` / `chmod` из блока выше и убедитесь, что на этот узел скопирован `web.pem` этого хоста.

Подготовьте конфигурацию на статических узлах кластера

Создайте на каждой машине пустую директорию `opt/ydb/cfg` для работы кластера с конфигурацией. В случае запуска нескольких узлов кластера на одной машине создайте отдельные директории под каждый узел.

```
sudo mkdir -p /opt/ydb/cfg
sudo chown -R ydb:ydb /opt/ydb/cfg
```

Выполнив специальную команду на каждой машине, инициализируйте эту директорию файлом конфигурации.

```
sudo /opt/ydb/bin/ydb admin node config init --config-dir /opt/ydb/cfg --from-config /tmp/config.yaml
```

Исходный файл `/tmp/config.yaml` после выполнения этой команды больше не используется, его можно удалить.

Запустите статические узлы

Вручную

Запустите сервис хранения данных YDB на каждом статическом узле кластера. Поочередно выполните команды ниже. Не вставляйте все строки разом вместе с `sudo su - ydb` — иначе следующие команды могут выполниться не от пользователя `ydb`.

После шага 1 дождитесь приглашения `ydb@...$`.

На шаге 4 дождитесь вывода `ydbd` в этом же терминале.

1. Переключитесь на пользователя `ydb`:

   ```
   sudo su - ydb
   ```

2. Перейдите в каталог установки:

   ```
   cd /opt/ydb
   ```

3. Укажите путь к библиотекам:

   ```
   export LD_LIBRARY_PATH=/opt/ydb/lib
   ```

4. Запустите статический узел:

   ```
   /opt/ydb/bin/ydbd server --log-level 3 --syslog --tcp --yaml-config /opt/ydb/cfg/config.yaml --grpcs-por
   t 2135 --ic-port 19001 --mon-port 8765 --mon-cert /opt/ydb/certs/web.pem --node static
   ```

   При успешном старте в выводе появится, в частности, `Determined node ID: ...`. Если проверка `web.pem` не пройдена, здесь будет ошибка `Permission denied` для `/opt/ydb/certs/web.pem`.

С использованием systemd

Создайте на каждом сервере, где будет размещен статический узел кластера, конфигурационный файл systemd `/etc/systemd/system/ydbd-storage.service` по приведенному ниже образцу. Образец файла также можно скачать из репозитория.

```
[Unit]
Description=YDB storage node
After=network-online.target rc-local.service
Wants=network-online.target
StartLimitInterval=10
StartLimitBurst=15

[Service]
Restart=always
RestartSec=1
User=ydb
PermissionsStartOnly=true
StandardOutput=syslog
StandardError=syslog
SyslogIdentifier=ydbd
SyslogFacility=daemon
SyslogLevel=err
Environment=LD_LIBRARY_PATH=/opt/ydb/lib
ExecStart=/opt/ydb/bin/ydbd server --log-level 3 --syslog --tcp \
    --yaml-config  /opt/ydb/cfg/config.yaml \
```

```
    --grpcs-port 2135 --ic-port 19001 --mon-port 8765 --kafka-port 9092 \
    --mon-cert /opt/ydb/certs/web.pem --node static
LimitNOFILE=65536
LimitCORE=0
LimitMEMLOCK=3221225472

[Install]
WantedBy=multi-user.target
```

Запустите сервис на каждом статическом узле YDB:

```
sudo systemctl start ydbd-storage
```

В конфигурации V2 YDB UI доступен для входа только **после** инициализации кластера. Сразу после запуска статических узлов попытка входа на экране логина завершится ошибкой — это нормально. Проверку UI выполните после раздела Инициализируйте кластер.

Инициализируйте кластер

Операция инициализации кластера осуществляет настройку набора статических узлов, перечисленных в конфигурационном файле кластера, для хранения данных YDB.

Для инициализации кластера потребуются файлы `ca.crt`, `node.crt` и `node.key` из каталога `~/CA/certs/` (см. раздел Скопируйте ключи и сертификаты TLS на каждый сервер). Пути к ним указываются в командах ниже (для bootstrap — все три, для получения токена — `ca.crt`).

На одном из серверов хранения в составе кластера выполните команды:

Инициализируйте кластер:

```
export LD_LIBRARY_PATH=/opt/ydb/lib
/opt/ydb/bin/ydb --ca-file ~/CA/certs/ca.crt \
    --client-cert-file ~/CA/certs/node.crt \
    --client-cert-key-file ~/CA/certs/node.key \
    -e grpcs://`hostname -f`:2135 \
    admin cluster bootstrap --uuid <строка>
echo $?
```

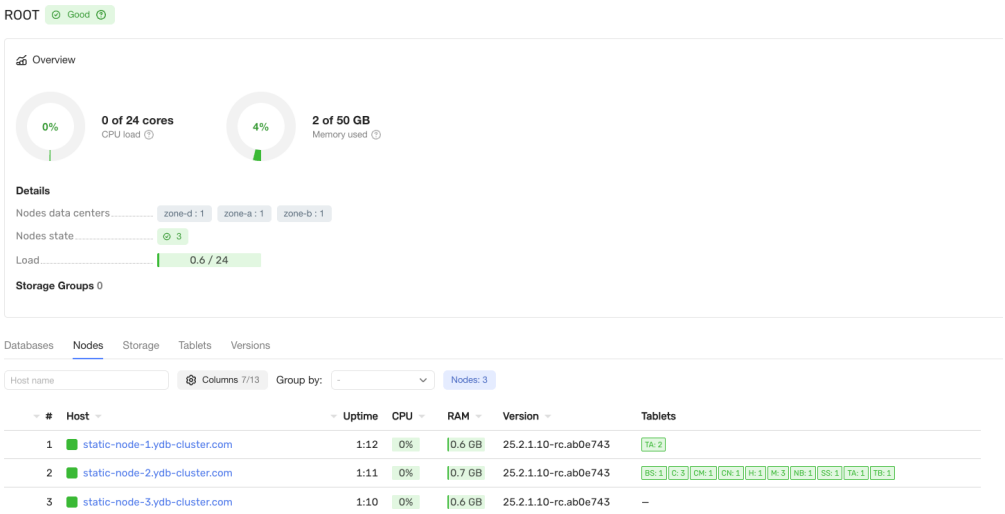
После инициализации кластера для выполнения дальнейших административных команд необходимо предварительно получить аутентификационный токен.

```
/opt/ydb/bin/ydb -e grpcs://`hostname -f`:2135 -d /Root --ca-file ~/CA/certs/ca.crt \
--user root --no-password auth get-token --force > auth_token
```

При успешном выполнении инициализации кластера выведенный на экран код завершения команды инициализации кластера должен быть нулевым.

После успешной инициализации кластера проверьте работоспособность статических узлов через YDB UI:

1. Откройте в браузере адрес `https://<static-node.ydb.tech>:8765`, где `<static-node.ydb.tech>` — FQDN сервера с запущенным статическим узлом;
2. Войдите под учётной записью `root` с пустым паролем (пароль задаётся позже, в разделе Настройка учётных записей);
3. Перейдите на вкладку **Nodes**;
4. Убедитесь, что в списке отображаются все статические узлы кластера.



Создайте базу данных

Для работы со строковыми или колоночными таблицами необходимо создать как минимум одну базу данных и запустить процесс или процессы, обслуживающие эту базу данных (динамические узлы).

Для выполнения административной команды создания базы данных потребуется файл `ca.crt` из каталога `~/CA/certs/` (см. раздел Скопируйте ключи и сертификаты TLS на каждый сервер).

При создании базы данных устанавливается первоначальное количество используемых групп хранения, определяющее доступную пропускную способность ввода-вывода и максимальную емкость хранения. Количество групп хранения может быть

при необходимости увеличено после создания базы данных.

На одном из серверов хранения в составе кластера выполните команды:

```
export LD_LIBRARY_PATH=/opt/ydb/lib
/opt/ydb/bin/ydbd --ca-file ~/CA/certs/ca.crt -s grpcs://`hostname -f`:2135 -f auth_token \
    admin database /Root/testdb create ssd:8
echo $?
```

При успешном создании базы данных, выведенный на экран код завершения команды должен быть нулевым.

В приведенном выше примере команд используются следующие параметры:

- `/Root` - имя корневого домена, сгенерированного автоматически при инициализации кластера;
- `testdb` - имя создаваемой базы данных;
- `ssd:8` - задает пул хранения для базы данных и количество групп в нем. Имя пула (`ssd`) должно соответствовать типу диска, указанному в конфигурации кластера (например, в `default_disk_type`), и является регистронезависимым. Число после двоеточия — это количество выделяемых групп хранения.

Запустите динамические узлы

Первый динамический узел запустите на одном из серверов кластера, где уже подготовлены каталоги `/opt/ydb/certs` и `/opt/ydb/cfg` и работает статический узел. Совмещение статического и динамического узла на одном сервере допустимо (см. Требования).

Для топологий из примера (`mirror-3-dc-3nodes` — три сервера, `mirror-3-dc-9nodes` — девять) для отказоустойчивости и масштабирования запустите дополнительные динамические узлы на тех же серверах, что и статические — по одному на каждом из оставшихся хостов.

Вручную

Запустите динамический узел YDB для базы `/Root/testdb`. Поочередно выполните команды ниже. Не вставляйте все строки разом вместе с `sudo su - ydb` — иначе следующие команды могут выполниться не от пользователя `ydb`.

После шага 1 дождитесь приглашения `ydb@...$`.

На шаге 4 дождитесь вывода `ydbd` в этом же терминале. В команде ниже замените `<ydb-static-node1>`, `<ydb-static-node2>`, `<ydb-static-node3>` на FQDN трёх любых серверов со статическими узлами.

1. Переключитесь на пользователя `ydb`:

   ```
   sudo su - ydb
   ```

2. Перейдите в каталог установки:

   ```
   cd /opt/ydb
   ```

3. Укажите путь к библиотекам:

   ```
   export LD_LIBRARY_PATH=/opt/ydb/lib
   ```

4. Запустите динамический узел:

   ```
   /opt/ydb/bin/ydbd server --grpcs-port 2136 --grpc-ca /opt/ydb/certs/ca.crt --ic-port 19002 --ca /opt/yd
   b/certs/ca.crt --mon-port 8766 --mon-cert /opt/ydb/certs/web.pem --yaml-config /opt/ydb/cfg/config.yaml
   --tenant /Root/testdb --grpc-cert /opt/ydb/certs/node.crt --grpc-key /opt/ydb/certs/node.key --node-brok
   er grpcs://<ydb-static-node1>:2135 --node-broker grpcs://<ydb-static-node2>:2135 --node-broker grpcs://<
   ydb-static-node3>:2135
   ```

   Если проверка `web.pem` не пройдена, будет ошибка `Permission denied` для `/opt/ydb/certs/web.pem`.

С использованием systemd

Создайте конфигурационный файл systemd `/etc/systemd/system/ydbd-testdb.service` по приведенному ниже образцу. Образец файла также можно скачать из репозитория.

```
[Unit]
Description=YDB testdb dynamic node
After=network-online.target rc-local.service
Wants=network-online.target
StartLimitInterval=10
StartLimitBurst=15

[Service]
Restart=always
RestartSec=1
User=ydb
PermissionsStartOnly=true
StandardOutput=syslog
StandardError=syslog
SyslogIdentifier=ydbd
SyslogFacility=daemon
SyslogLevel=err
Environment=LD_LIBRARY_PATH=/opt/ydb/lib
ExecStart=/opt/ydb/bin/ydbd server \
    --grpcs-port 2136 --grpc-ca /opt/ydb/certs/ca.crt \
    --ic-port 19002 --ca /opt/ydb/certs/ca.crt \
```

```
    --mon-port 8766 --mon-cert /opt/ydb/certs/web.pem \
    --kafka-port 9093 \
    --yaml-config  /opt/ydb/cfg/config.yaml \
    --tenant /Root/testdb \
    --grpc-cert /opt/ydb/certs/node.crt \
    --grpc-key /opt/ydb/certs/node.key \
    --node-broker grpcs://<ydb-static-node1>:2135 \
    --node-broker grpcs://<ydb-static-node2>:2135 \
    --node-broker grpcs://<ydb-static-node3>:2135
LimitNOFILE=65536
LimitCORE=0
LimitMEMLOCK=32212254720

[Install]
WantedBy=multi-user.target
```

В примере команды выше `<ydb-static-node1>`, `<ydb-static-node2>`, `<ydb-static-node3>` - FQDN трех любых серверов, на которых запущены статические узлы кластера.

Запустите динамический узел YDB для базы `/Root/testdb`:

```
sudo systemctl start ydbd-testdb
```

Настройка учетных записей

В командах ниже `<node.ydb.tech>` — FQDN сервера с запущенным динамическим узлом базы `/Root/testdb` (порт gRPC `2136`). При выполнении команд по SSH на этом сервере можно указать endpoint: `grpcs://$(hostname -f):2136`.

1. Установите пароль для учетной записи `root`, используя полученный ранее токен:

   ```
   ydb --ca-file ~/CA/certs/ca.crt -e grpcs://<node.ydb.tech>:2136 -d /Root/testdb --token-file auth_token
   \
       yql -s 'ALTER USER root PASSWORD "passw0rd"'
   ```

   Вместо значения `passw0rd` подставьте необходимый пароль. Сохраните пароль в отдельный файл. Последующие команды от имени пользователя `root` будут выполняться с использованием пароля, передаваемого с помощью ключа `--password-file <path_to_user_password>`. Также пароль можно сохранить в профиле подключения, как описано в документации YDB CLI.
2. Создайте дополнительные учетные записи:

   ```
   ydb --ca-file ~/CA/certs/ca.crt -e grpcs://<node.ydb.tech>:2136 -d /Root/testdb --user root --password-f
   ile <path_to_root_pass_file> \
       yql -s 'CREATE USER user1 PASSWORD "passw0rd"'
   ```
3. Установите права учетных записей, включив их во встроенные группы:

   ```
   ydb --ca-file ~/CA/certs/ca.crt -e grpcs://<node.ydb.tech>:2136 -d /Root/testdb --user root --password-f
   ile <path_to_root_pass_file> \
       yql -s 'ALTER GROUP `ADMINS` ADD USER user1'
   ```

При выполнении команд создания учётных записей и присвоения групп клиент YDB CLI будет запрашивать ввод пароля пользователя `root`. Избежать многократного ввода пароля можно, создав профиль подключения, как описано в документации YDB CLI.

Протестируйте работу с созданной базой

1. Установите YDB CLI, как описано в документации.

В командах ниже `<node.ydb.tech>` — тот же FQDN динамического узла базы `/Root/testdb`, что в разделе Настройка учетных записей.

1. Создайте тестовую строковую (`test_row_table`) или колоночную таблицу (`test_column_table`):

Создание строковой таблицы

```
ydb --ca-file ~/CA/certs/ca.crt -e grpcs://<node.ydb.tech>:2136 -d /Root/testdb --user root \
    yql -s 'CREATE TABLE `testdir/test_row_table` (id Uint64, title Utf8, PRIMARY KEY (id));'
```

Создание колоночной таблицы

```
ydb --ca-file ~/CA/certs/ca.crt -e grpcs://<node.ydb.tech>:2136 -d /Root/testdb --user root \
    yql -s 'CREATE TABLE `testdir/test_column_table` (id Uint64 NOT NULL, title Utf8, PRIMARY KEY (id)) WITH
(STORE = COLUMN);'
```

Проверка доступа к YDB UI

Для проверки доступа к YDB UI достаточно открыть в Web-браузере страницу с адресом `https://<static-node.ydb.tech>:8765`, где `<static-node.ydb.tech>` - FQDN сервера, на котором запущен любой статический узел YDB.

В Web-браузере должно быть настроено доверие в отношении центра регистрации, выпустившего сертификаты для кластера YDB, в противном случае будет отображено предупреждение об использовании недоверенного сертификата.

Если в кластере включена аутентификация, в Web-браузере должен отобразиться запрос логина и пароля. После ввода верных данных аутентификации должна отобразиться начальная страница YDB UI. Описание доступных функций и

пользовательского интерфейса приведено в разделе Использование YDB UI.

Примечание

Обычно для обеспечения доступа к YDB UI настраивают отказоустойчивый HTTP-балансировщик на базе программного обеспечения `haproxy`, `nginx` или аналогов. Детали настройки HTTP-балансировщика выходят за рамки стандартной инструкции по установке YDB.

Особенности установки YDB в незащищенном режиме

Важно

Мы не рекомендуем использовать незащищенный режим работы YDB ни при эксплуатации, ни при разработке приложений.

Описанная выше процедура установки предусматривает развёртывание YDB в стандартном защищенном режиме.

Незащищённый режим работы YDB предназначен для решения тестовых задач, преимущественно связанных с разработкой и тестированием программного обеспечения YDB. В незащищенном режиме:

- трафик между узлами кластера, а также между приложениями и кластером использует незашифрованные соединения;
- не используется аутентификация пользователей (включение аутентификации при отсутствии шифрования трафика не имеет смысла, поскольку логин и пароль в такой конфигурации передавались бы через сеть в открытом виде).

Установка YDB для работы в незащищенном режиме производится в порядке, описанном выше, со следующими исключениями:

1. При подготовке к установке не требуется формировать сертификаты и ключи TLS, и не выполняется копирование сертификатов и ключей на узлы кластера.
2. Из конфигурационных файлов кластерных узлов исключаются секции `security_config`, `interconnect_config` и `grpc_config`.
3. Используются упрощенный вариант команд запуска статических и динамических узлов кластера: исключаются опции с именами файлов сертификатов и ключей, используется протокол `grpc` вместо `grpcs` при указании точек подключения.
4. Пропускается ненужный в незащищенном режиме шаг по получению токена аутентификации перед выполнением инициализации кластера и созданием базы данных.
5. Команда инициализации кластера выполняется в следующей форме:

   ```
   export LD_LIBRARY_PATH=/opt/ydb/lib
   ydb admin cluster bootstrap --uuid <строка>
   echo $?
   ```

6. Команда создания базы данных выполняется в следующей форме:

   ```
   export LD_LIBRARY_PATH=/opt/ydb/lib
   /opt/ydb/bin/ydbd admin database /Root/testdb create ssd:1
   ```

7. При обращении к базе данных из YDB CLI и приложений используется протокол grpc вместо grpcs, и не используется аутентификация.

Обновление конфигурации кластеров YDB, развёрнутых вручную

При ручном развертывании кластера YDB управление конфигурацией осуществляется через YDB CLI. В этой статье рассматриваются способы изменения конфигурации кластера после первоначального развертывания.

Базовые операции с конфигурацией

Получение текущей конфигурации

Для получения текущей конфигурации кластера используется команда:

```
ydb -e grpcs://<endpoint>:2135 admin cluster config fetch > config.yaml
```

В качестве `<endpoint>` указывается адрес любого из узлов кластера.

Применение новой конфигурации

Для загрузки обновленной конфигурации на кластер используется следующая команда:

```
ydb -e grpcs://<endpoint>:2135 admin cluster config replace -f config.yaml
```

Некоторые параметры конфигурации применяются на ходу после выполнения команды, однако для некоторых требуется выполнение процедуры перезапуска кластера.

Обслуживание кластера

Утилита ydbops использует CMS для проведения обслуживания кластера без потери доступности. Также CMS можно использовать напрямую через gRPC API.

Rolling restart

Чтобы выполнить rolling restart всего кластера можно воспользоваться командой:

```
$ ydbops restart
```

По умолчанию, будет использован режим доступности `strong`, минимизирующий риск потери доступности. Его можно переопределить с помощью параметра `--availability-mode`.

Утилита `ydbops` автоматически создаст задачу обслуживания на рестарт всего кластера, используя указанный режим доступности. По ходу продвижения `ydbops` будет обновлять задачу обслуживания и получать эксклюзивные блокировки на узлы в CMS, пока все узлы не будут перезапущены.

Вывести хост для обслуживания

Чтобы вывести хост для обслуживания, выполните следующие шаги:

1. Создать задачу обслуживания с помощью команды:

   ```
   $ ydbops maintenance create --hosts=<fqdn> --duration=<seconds>
   ```

   Эта команда создаст задачу обслуживания, в рамках которой на хост с полным доменным именем `<fqdn>` будет взята эксклюзивная блокировка на `<seconds>` секунд.
2. После создания задачи необходимо обновлять ее состояние, пока блокировка не будет взята, с помощью команды:

   ```
   $ ydbops maintenance refresh --task-id=<id>
   ```

   Эта команда обновит задачу с идентификатором `<id>` и попытается взять требуемую блокировку. При получении ответа `PERFORMED` можно переходить к следующему пункту.
3. Провести обслуживание хоста пока взята блокировка.
4. После завершения работ необходимо отпустить блокировку на хост с помощью команды:

   ```
   $ ydbops maintenance complete --task-id=<id> --hosts=<fqdn>
   ```

Обновление исполняемого файла YDB

YDB распределенная система, поддерживающая плавное обновление (rolling restart) без даунтайма и деградации производительности.

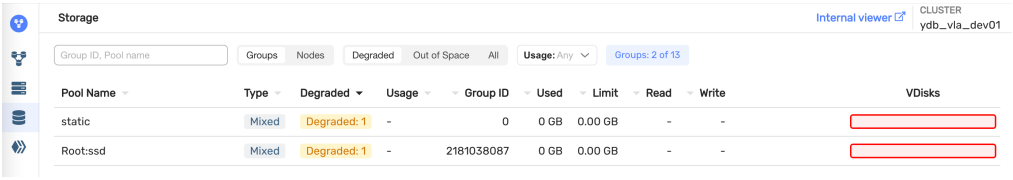
> **Важно**
>
> У YDB существуют специфические правила относительно совместимости версий. Важно ознакомиться с руководством по именованию версий YDB и их совместимости и списком изменений, чтобы правильно выбрать новую версию для обновления и подготовиться к возможным нюансам.

Порядок обновления

Базовым сценарием является обновление исполняемого файла и затем последовательный рестарт каждого узла:

1. Обновление и рестарт узлов хранения;
2. Обновление и рестарт узлов баз данных.

Процесс остановки и запуска описан на странице Безопасные рестарт и выключение узлов.
Узлы YDB следует обновлять последовательно по одному, после каждого шага контролировать состояние кластера через YDB Monitoring - на вкладке `Storage` не должно быть пулов в состоянии `Degraded` (как на примере ниже). В противном случае обновление необходимо остановить.



Проверка результата обновления

Проверить версии узлов после обновления можно на странице `Nodes` мониторинга.

Управление кластером в режиме bridge

> **Функциональность Корпоративной СУБД Яндекса**
>
> Данная функциональность доступна только в Корпоративной СУБД Яндекса. В open-source версии YDB она отсутствует.

Ниже приведены типовые операции для кластера в режиме bridge с использованием соответствующих команд YDB CLI.

Посмотреть текущее состояние

Показывает текущее состояние каждого pile, настроенного на кластере YDB.

```
ydb admin cluster bridge list
```

Пример вывода:

```
pile-a: PRIMARY
pile-b: SYNCHRONIZED
```

Плановая смена `PRIMARY` (switchover)

Если известно, что в обозримом будущем запланированы плановые работы в датацентре или на оборудовании, на котором работает текущий `PRIMARY` pile, то рекомендуется заранее переключить кластер на использование другого pile в роли `PRIMARY`. Выберите другой pile в состоянии `SYNCHRONIZED`, чтобы переключить его в состояние `PRIMARY` следующей командой:

```
ydb admin cluster bridge switchover --new-primary <pile>
```

Переключение выполняется плавно: роли проходят через `PRIMARY/PROMOTED` и завершаются в состоянии `SYNCHRONIZED/PRIMARY`.

Плановое отключение pile (takedown)

Если плановые работы приведут к недоступности одного из pile, его необходимо вывести из кластера перед их началом с помощью следующей команды:

```
ydb admin cluster bridge takedown --pile <pile>
# если отключаете текущий PRIMARY:
ydb admin cluster bridge takedown --pile <current-primary> --new-primary <synchronized-pile>
```

При выполнении операции pile переводится в `SUSPENDED`, затем — в `DISCONNECTED`; дальнейшие операции с кластером выполняются без участия отключённого pile.

Если отключается текущий `PRIMARY` и не было возможности сменить его заранее, то эти операции можно совместить, указав новый `PRIMARY` в аргументе `--new-primary`, который должен быть в состоянии `SYNCHRONIZED`.

```
ydb admin cluster bridge takedown --pile <pile>
# если отключаете текущий PRIMARY:
ydb admin cluster bridge takedown --pile <current-primary> --new-primary <synchronized-pile>
```

> **Важно**
>
> Перед началом плановых работ обязательно убедитесь через команду list, что операция вывода pile из кластера завершилась успешно и все pile находятся в ожидаемом состоянии.

Аварийное отключение недоступного pile (failover)

Так как между pile работает синхронная репликация, то при неожиданном выходе одного из них из строя работа кластера по умолчанию останавливается, и необходимо принять решение, продолжать ли работу кластера без этого pile. Это решение может принимать как человек (например, дежурный DevOps-инженер), так и внешняя по отношению к кластеру YDB автоматизация.

В случае положительного решения о продолжении работы кластера необходимо выполнить следующую команду:

```
ydb admin cluster bridge failover --pile <unavailable-pile>
```

Если недоступен текущий `PRIMARY`, необходимо добавить параметр `--new-primary` с указанием имени pile в состоянии `SYNCHRONIZED`. Если параметр не указан или указан некорректно, команда завершится с ошибкой без изменений в кластере.

```
ydb admin cluster bridge failover --pile <unavailable-pile>
# если недоступен текущий PRIMARY:
ydb admin cluster bridge failover --pile <unavailable-primary> --new-primary <synchronized-pile>
```

Если одновременно отказали несколько pile, перечислите их в одной команде, повторив `--pile`. Например, если отказали `pile-a` и `pile-b`, а `pile-c` остаётся `SYNCHRONIZED`, подключитесь к узлу в `pile-c` и выполните:

```
ydb admin cluster bridge failover --pile pile-a --pile pile-b --new-primary pile-c
```

Одно обновление конфигурации отключает оба недоступных pile и выбирает новый `PRIMARY`. Отключение по одному может не сработать: пока второй отказавший pile участвует в кворуме конфигурации, для первого обновления невозможно собрать этот кворум. В оставшемся pile по-прежнему должны быть доступны кворумы узлов и хранилища, необходимые для применения новой конфигурации и обработки запросов.

Перед выполнением команды убедитесь, что отказавшие pile остановлены или изолированы и не смогут самостоятельно возобновить обработку запросов. Не выполняйте failover независимо по разные стороны сетевого разделения; см. восстановление после split-brain. Команда не включает автоматическое переключение и не позволяет назначить несинхронизированный pile новым `PRIMARY`. После успешного выполнения проверьте результат командой list. После устранения отказа верните каждый pile командой rejoin.

Недоступный pile будет переведён в состояние `DISCONNECTED`, а при указании нового `PRIMARY` произойдёт переключение этой роли. Если остальные pile находятся в состояниях, отличных от `SYNCHRONIZED`, аварийное отключение также может быть выполнено. Допустимые переходы зависят от текущей пары состояний и приведены на диаграмме состояний и в таблице переходов.

> **Важно**
>
> Если включена обязательная аутентификация, то в аварийном состоянии кластера аутентификация по логину и паролю не может быть использована — используйте аутентификацию по клиентским сертификатам. Подробнее см. Особенности аутентификации в аварийном состоянии кластера.

Вернуть pile в кластер (rejoin)

После завершения плановых работ или устранения причин отказа ранее отключённые pile необходимо явным образом вводить обратно в эксплуатацию следующей командой:

```
ydb admin cluster bridge rejoin --pile <pile>
```

Сразу после запуска операции pile переходит в состояние `NOT_SYNCHRONIZED` и запускается фоновый процесс синхронизации данных; по её завершении pile автоматически становится `SYNCHRONIZED`. Дождавшись этого состояния, при необходимости можно переключить роль `PRIMARY` на данный pile.

Особенности аутентификации в аварийном состоянии кластера

Команды управления кластером в режиме bridge требуют прав администратора кластера. Если включена обязательная аутентификация (`enforce_user_token_requirement: true`), то в штатном режиме работы кластера доступны все способы аутентификации, включая аутентификацию по логину и паролю.

Однако при отказе pile кластер приостанавливает обслуживание запросов до выполнения failover. В этом состоянии аутентификация по логину и паролю не может быть использована: для проверки учётных данных нужен доступ к SchemeShard, который недоступен на кластере в аварийном состоянии. Попытка входа завершается ошибкой:

```
SchemeShard is unreachable
```

Единственный способ аутентификации, доступный в аварийном состоянии кластера, — аутентификация по клиентским сертификатам. Клиентский сертификат проверяется локально узлом, принявшим запрос, поэтому такая проверка не зависит от доступности кластера в целом.

Аутентификация по клиентским сертификатам должна быть настроена на кластере заранее:

1. Секция `client_certificate_authorization` конфигурации кластера должна присваивать подключениям с доверенным клиентским сертификатом SID административной группы:

   ```
   client_certificate_authorization:
     request_client_certificate: true
     client_certificate_definitions:
       - member_groups: ["ADMINS"]
         subject_terms:
         - short_name: "O"
           values: ["YDB"]
   ```

2. Этот SID должен входить в список `administration_allowed_sids` секции `security_config`:

   ```
   security_config:
     enforce_user_token_requirement: true
     administration_allowed_sids:
     - "root"
     - "ADMINS"
   ```

Если кластер развёрнут по инструкции по развёртыванию, эти настройки уже включены в конфигурацию, а в качестве клиентского сертификата подойдут файлы `node.crt` и `node.key` из каталога `~/CA/certs/` на любом узле кластера.

Для аутентификации по клиентскому сертификату укажите его в глобальных опциях YDB CLI `--client-cert-file` и `--client-cert-key-file`. Подключаться необходимо к узлу работоспособного pile. Пример выполнения failover:

```
ydb -e grpcs://<node.ydb.tech>:2135 \
    --ca-file ca.crt \
    --client-cert-file node.crt \
    --client-cert-key-file node.key \
    admin cluster bridge failover --pile <unavailable-pile> --new-primary <synchronized-pile>
```

где:

- `<node.ydb.tech>` — FQDN узла в работоспособном pile;

- `ca.crt` — сертификат доверенного центра сертификации кластера;
- `node.crt`, `node.key` — клиентский сертификат и приватный ключ к нему, удовлетворяющие требованиям секции `client_certificate_authorization`.

Важно

Проверьте работоспособность аутентификации по клиентским сертификатам заранее, до возникновения аварии. Если аутентификация по клиентским сертификатам не была настроена до аварии, вывод кластера из аварийного состояния существенно усложнится и потребует ручного вмешательства на узлах кластера.

Обзор управления дисковой подсистемой кластера

Управление дисковой подсистемой кластера включает следующие действия:

- Изменение конфигурации:
  - Расширение кластера.
  - Добавление групп хранения.
- Обслуживание:
  - Замена FQDN статического узла.
  - Безопасный рестарт и выключение узлов.
  - Включение и выключение Scrubbing.
  - SelfHeal.
  - Декомиссия ЦОД.
  - Декомиссия групп с использованием виртуальных групп.
  - Перевоз VDisk'ов.
  - Блобовница.
  - Декомиссия групп.
- Решение проблем:
  - Предотвращение выхода за модель отказа.
  - Балансировка нагрузки на диски.
  - Освобождение места на физических устройствах.

Добавление групп хранения

По мере роста объема хранимых данных может понадобиться добавление дисков в кластер YDB. Диски могут быть добавлены в уже существующие узлы или вместе с новыми узлами. Для того, чтобы ресурсы новых дисков стали доступны БД, необходимо добавить группы хранения.

Чтобы добавить новые группы хранения, воспользуйтесь утилитой YDB DSTool.

Просмотрите список пулов хранения кластера:

```
ydb-dstool -e <bs_endpoint> pool list
```

`<bs_endpoint>` - эндпоинт произвольного узла хранения кластера.

> Пример результата:
>
> ```
> ┌──────────────┬──────────────────┬─────────────────┬──────┬──────────────┬──────────────┐
> | BoxId:PoolId | PoolName         | ErasureSpecies | Kind | Groups_TOTAL | VDisks_TOTAL |
> ├──────────────┼──────────────────┼─────────────────┼──────┼──────────────┼──────────────┤
> | [1:1]        | /Root/testdb:ROT | mirror-3-dc     | ROT  | 1            | 9            |
> └──────────────┴──────────────────┴─────────────────┴──────┴──────────────┴──────────────┘
> ```

Следующая команда добавит 10 групп в пул `/Root/testdb:ROT`:

```
ydb-dstool -e <bs_endpoint> group add --pool-name /Root/testdb:ROT --groups 10
```

В случае успеха команда вернет нулевой `exit status`. Иначе команда вернет ненулевой статус и выведет сообщение об ошибке в `stderr`.

Чтобы проверить возможность добавления групп без фактического добавления, используйте глобальный параметр `--dry-run`. Следующая команда проверит возможность добавления 100 групп в пул `/Root/testdb:ROT`:

```
ydb-dstool --dry-run -e <bs_endpoint> group add --pool-name /Root/testdb:ROT --groups 100
```

Параметр `--dry-run` позволяет оценить, какое максимальное число групп можно добавить в пул.

Безопасный рестарт и выключение узлов

Остановка/рестарт процесса YDB на узле

Чтобы убедиться, что процесс можно остановить, надо выполнить следующие шаги.

1. Перейти в узел по ssh.
2. Выполнить команду

   ```
   ydbd cms request restart host {node_id} --user {user} --duration 60 --dry --reason 'some-reason'
   ```

   При разрешение выведет `ALLOW`.
3. Остановить процесс

   ```
   sudo service ydbd stop
   ```

4. Если потребуется, запустить процесс

   ```
   sudo service ydbd start
   ```

Замена оборудования

Перед заменой нужно убедиться, что процесс YDB можно остановить.
При длительном отсутствии стоит перед этим перевезти все VDisk'и с данного узла и дождаться окончания репликации.
После окончания репликации узел можно безопасно выключать.

Чтобы отключение динамического узла не оказало влияние на обработку запросов, необходимо перед отключением выполнить мягкий перенос (drain) таблеток с этого узла.

Стоит перейти на страницу Hive web-viewer.
После нажатия на кнопку "View Nodes" отобразится список всех узлов.

Перед отключением узла, сначала требуется отключить перевоз таблеток через кнопку Active, после чего нажать Drain и дождаться увоза всех таблеток.

Включение и выключение Scrubbing

Scrubbing — процесс читает данные, проверяет их на целостность и, если нужно, восстанавливает ее. Процесс запущен по умолчанию, интервал между окончанием проверки и запуском следующей — 1 месяц. Вы можете изменить интервал с помощью утилиты YDB DSTool. Проверяются данные, последний доступ к которым был раньше времени предыдущей проверки. Scrubbing-процесс запускается или останавливается для всего кластера YDB. Проверка выполняется в фоне и не перегружает систему.

Чтобы задать интервал в 48 часов, выполните команду:

```
ydb-dstool -e <bs_endpoint> cluster set --scrub-periodicity 48h
```

`<bs_endpoint>` - эндпоинт произвольного узла хранения кластера.

Так же можно указать максимальное число дисков кластера, которые будут проверяться одновременно. Например, чтобы проверять одновременно не более одного диска, выполните команду:

```
ydb-dstool -e <bs_endpoint> cluster set --max-scrubbed-disks-at-once
```

Чтобы остановить scrubbing-процесс на кластере, выполните команду:

```
ydb-dstool -e <bs_endpoint> cluster set --scrub-periodicity disable
```

Декомиссия ЦОД

> **Примечание**
>
> Описанную процедуру можно применять только для декомиссии целого центра обработки данных (ЦОД).

Декомиссия — это процедура перемещения VDisk с PDisk, который требуется вывести из эксплуатации.

Данные перемещаются на PDisk кластера, на которых достаточно свободного пространства для создания новых слотов. Перемещение выполняется только в случае, когда есть возможность перемещения с хотя бы частичным сохранением модели отказа. Строгое соблюдение модели отказа во время декомиссии может быть нарушено, однако система стремится к обеспечению в процессе декомиссии отказоустойчивости не хуже, чем при штатной работе. Например, при кодировании `mirror-3-dc` может сложиться ситуация, когда группа находится в 4 ЦОД вместо 3.

Декомиссия выполняется асинхронно, то есть при обработке команды не происходит немедленного перемещения данных. Механизм SelfHeal в фоновом режиме, когда позволяет модель отказа, перемещает слоты по одному, добиваясь полного освобождения указанных PDisk.

Управление декомиссией

Управление состоянием декомиссии осуществляется заданием параметра `DecommitStatus` для PDisk. Параметр может принимать следующие значения:

- `DECOMMIT_NONE` — диск не участвует в декомиссии и работает штатно, согласно его состоянию.
- `DECOMMIT_PENDING` — запланирована декомиссия диска. Данные с диска не переносятся, однако слоты для новых групп создаваться не будут, а также не будут перемещаться слоты созданных ранее групп.
- `DECOMMIT_IMMINENT` — необходимо провести декомиссию диска. Данные переносятся в фоновом режиме на диски, которые имеют статус `DECOMMIT_NONE` и удовлетворяют модели отказа.

Значения `DECOMMIT_PENDING` и `DECOMMIT_IMMINENT` в случае штатной декомиссии снимать не нужно, т.к. оборудование удаляется из кластера через команду `DefineBox`.

Чтобы отменить декомиссию, достаточно изменить состояние дисков в `DECOMMIT_NONE`. При этом никаких дополнительный действий BS_CONTROLLER предпринимать не будет — уже перевезенные VDisk останутся на своих местах. Если требуется их вернуть, можно использовать команды для точечного перемещения слотов в зависимости от конкретной ситуации.

Управление состояниями `DECOMMIT_PENDING` и `DECOMMIT_IMMINENT` позволяют провести декомиссию кластера по частями.

Например, нужно переместить оборудование из ЦОД-1 в ЦОД-2:

1. В ЦОД-2 размещается буферное оборудование, на которое будет перемещена первая порция данных.
2. Все диски ЦОД-1 единовременно переводятся в состояние `DECOMMIT_PENDING`, чтобы исключить перемещение данных внутри ЦОД-1. Статус необходимо задать сразу на все диски датацентра: если перевести в `DECOMMIT_PENDING` только часть дисков ЦОД, это может привести к нарушению модели отказа групп хранения.
3. В ЦОД-1 все диски оборудования, эквивалентного буферному, переводятся в состояние `DECOMMIT_IMMINENT`.

   Необходимо дождаться освобождения всех дисков в состоянии `DECOMMIT_IMMINENT`.
4. Освободившееся оборудование из ЦОД-1 перемещается в ЦОД-2, его диски переводятся в состояние `DECOMMIT_NONE`.

Описанные шаги повторяются для следующего набора оборудования в ЦОД-1 до тех пор, пока все оборудование не будет перемещено.

Чтобы задать нужное состояние декомиссии диска, воспользуйтесь утилитой YDB DSTool. Следующая команда задаст статус `DECOMMIT_IMMINENT` диску с ID `1000` на ноде с ID `1`:

```
ydb-dstool -e  <bs_endpoint> pdisk set --decommit-status DECOMMIT_IMMINENT --pdisk-ids "[1:1000]"
```

Декомиссия групп с использованием виртуальных групп

Внимание

Операция декомиссии является крайне опасной и может привести к необратимой потере данных. Перед выполнением декомиссии **обязательно** создайте резервную копию всех данных кластера.

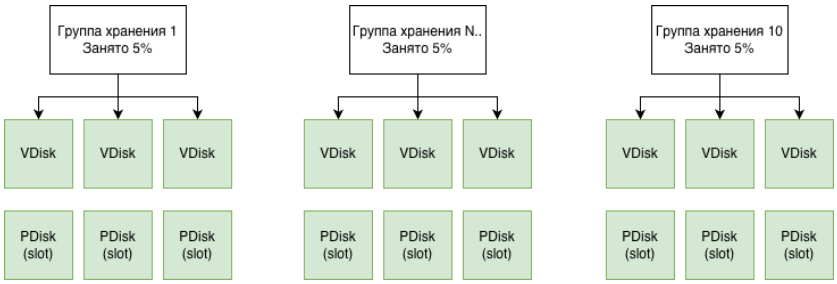
Внимание

Возможность откатить декомиссию группы хранения находится в разработке. В данный момент операция необратима. Рекомендуется использовать декомиссию только в исключительных случаях, когда требуется разово освободить ресурсы.

Описание работы

Физические группы хранения являются ограниченным ресурсом в кластере: их количество определяется размером кластера, и их нельзя удалить, пока в них хранятся данные. Это может привести к ситуации, когда для создания новой базы данных или расширения существующей не хватает групп, а освободить пространство на физических дисках нельзя, поскольку старые группы содержат данные.

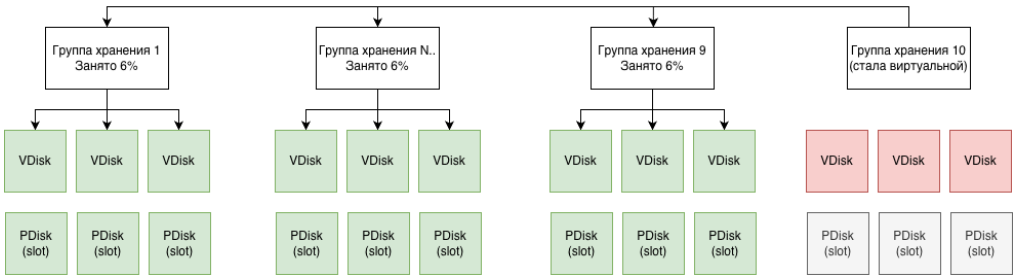
Рассмотрим пример: в базе данных используется 10 групп хранения. Каждая из этих групп содержит некоторый объём пользовательских данных, но при этом заполнены они только на 5%. Пропускная способность также используется не полностью. В результате все группы формально заняты, и при попытке создать новую базу данных новая группа хранения не может быть выделена — свободных слотов на PDisk больше нет. При этом существующие группы используются неэффективно.



В такой ситуации возникает задача освободить часть групп хранения (декомиссовать их) у текущей базы данных, чтобы вернуть их в пул доступных ресурсов для последующего использования.

Для решения этой задачи применяется механизм декомиссии групп хранения. В рамках этого механизма для исходной физической группы создаётся одна новая виртуальная группа хранения, которая заменяет эту физическую группу. После выполнения операции система переходит в следующее состояние:

- Исходная физическая группа хранения заменяется на виртуальную.
- Данные копируются из исходной группы в оставшиеся физические группы кластера (увеличивая их заполненность)
- Виртуальная группа хранит данные в оставшихся физических группах кластера.
- Ресурсы, используемые исходной группой, освобождаются (VDisk'и удаляются, слоты PDisk'ов освобождаются).



Этапы процесса декомиссии

Примечание

Во время декомиссии доступ к данным сохраняется — пользователи могут читать и записывать данные, как обычно. Все перемещения данных происходят в фоновом режиме параллельно с основной работой приложения. Стоит отметить, что во во время декомиссии все же создается повышенная нагрузка на систему хранения.

В дальнейшем под исходной физической группой будем подразумевать группу, над которой проходит процесс декомиссии.

Инициализация виртуальной группы хранения и блокировка записи в исходную физическую группу

Что происходит:

- Создается виртуальная группа.
- Блокируется запись новых данных в исходную физическую группу.
- Все новые записи начинают идти в виртуальную группу.

Статус: `DecommitStatus: PENDING` → `IN_PROGRESS`

Время: Обычно доли секунды

Копирование метаданных

Что происходит:

- Копируются метаданные всех блобов (информация о том, какие данные есть, но не сами данные) из исходной физической группы в оставшиеся физические группы хранения

Статус: `DecommitStatus: IN_PROGRESS`

Время: Обычно секунды

Копирование данных

Что происходит:

- Читаются данные из исходной физической группы.
- Записываются в оставшиеся физические группы кластера, через виртуальную группу.
- Данные распределяются по нескольким группам для балансировки.

Статус: `DecommitStatus: IN_PROGRESS`

Время: Зависит от объема данных (могут быть часы для больших групп)

Освобождение ресурсов исходной физической группы

Что происходит:

- Все данные скопированы
- Удаляется исходная физическая группа и её VDisk'и
- Освобождаются ресурсы PDisk
- Группа помечается как декомиссированная.

Статус: `DecommitStatus: IN_PROGRESS` → `DONE`

Время: Минуты (удаление VDisk)

Предварительные требования

Инструкция по декомиссии групп работает при выполнении следующих условий:

1. Развернутый кластер YDB. Инструкции по развертыванию доступны в разделе Первоначальное развёртывание.
2. Установленная утилита `ydb-dstool`. Инструкции по установке доступны в документации по установке YDB DSTool.
3. Сгенерированный токен аутентификации для доступа к кластеру:

```
ydb -e grpc://<cluster-endpoint> -d <db_path> --user <user> auth get-token -f > ~/ydb_token
```

Замените `<cluster-endpoint>` на адрес вашего кластера, `<db_path>` - путь к базе данных, `<user>` на имя пользователя для авторизации.

Примечание

Декомиссию можно проводить даже на пустом кластере, наличие данных не обязательно. Однако если в кластере есть данные, это позволит проверить их консистентность после декомиссии. При необходимости можно загрузить тестовые данные с помощью утилиты `ydb workload tpcc`:

```
ydb workload tpcc --path tpcc/10wh init -w 100
ydb workload tpcc --path tpcc/10wh import -w 100
```

И запустить тест TPC-C до и после декомиссии:

```
ydb workload tpcc --path tpcc/10wh run -w 100
```

Подробнее о работе с тестовой нагрузкой в документации по ydb workload tpcc.

Запуск декомиссии

Выбор группы для декомиссии

Перед запуском декомиссии необходимо выбрать группу (group ID), которую вы хотите вывести из эксплуатации. Выбирать можно любую группу, однако рекомендуется выбирать менее занятую группу — это поможет завершить процесс декомиссии быстрее. Это можно сделать через веб-интерфейс кластера или с помощью CLI:

UI (Веб-интерфейс)

1. Откройте веб-интерфейс управления кластером `https://<cluster-endpoint>:8765`.
2. Откройте вкладку `Storage`.
3. Выберите нужную вам группу и сохраните `Group ID`.

CLI

1. Выполните команду получения списка групп хранения

```
ydb-dstool -e <cluster-endpoint> \
  --ca-file <path-to-ca-cert> \
  --token-file ~/ydb_token \
  group list
```

2. В выводе команды найдите нужную группу и сохраните её `Group ID`.

Команда для запуска

Для запуска декомиссии выполните команду `ydb-dstool` с параметром `group decommit`, она автоматически создаст виртуальную группу и начнёт процесс декомиссии:

```
ydb-dstool -e grpcs://<cluster-endpoint>:2135 \
  --ca-file <path-to-ca-cert> \
  --token-file ~/ydb_token \
  group decommit \
  --group-ids <GROUP_ID> \
  --database /Root/<database-name> \
  --wait
```

Параметры команды

- `-e <endpoint>` — адрес кластера (например, `grpcs://static-node-1.ydb-cluster.com:2135`);
- `--ca-file <path>` — путь к файлу сертификата CA для TLS-соединения;
- `--token-file <path>` — путь к файлу с токеном аутентификации;
- `--group-ids <GROUP_ID>` — идентификатор группы для декомиссии (можно указать несколько групп через пробел);
- `--database=<DB>` — путь к базе данных или домену, в котором выполняется декомиссия (например, `/Root/database`);
- `--wait` — дождаться запуска декомиссии; в случае возникновения ошибки запуска ошибка выводится на экран, декомиссия отменяется автоматически (только при указании этой опции).

Пример команды

```
ydb-dstool -e grpcs://static-node-1.ydb-cluster.com:2135 \
  --ca-file /home/ubuntu/ansible-11-nov-244/3-nodes-mirror-3-dc/files/TLS/certs/ca.crt \
  --token-file ~/ydb_token \
  group decommit \
  --group-ids 2181038081 \
  --database=/Root/database \
  --wait
```

Проверка статуса декомиссии

После запуска декомиссии можно проверить её статус с помощью команды:

```
ydb-dstool -e grpcs://<cluster-endpoint>:2135 \
  --ca-file <path-to-ca-cert> \
  --token-file ~/ydb_token \
  group list --columns GroupId PoolName \
  ErasureSpecies OperatingStatus VDisks_TOTAL BlobDepotId DecommitStatus
```

В выводе команды для групп, для которых была начата процедура декомиссии, появляется дополнительное поле `DecommitStatus`, которое может принимать одно из следующих значений:

- `NONE` — декомиссия для указанной группы не производится;
- `PENDING` — ожидается декомиссия группы, но ещё не выполняется (создаётся виртуальная группа);
- `IN_PROGRESS` — производится декомиссия группы (все записи уже попадают в виртуальную группу, чтения — в виртуальную группу и в старую группу);
- `DONE` — декомиссия завершена полностью (физическая группа превратилась в виртуальную).

Пример вывода команды

```
┌─────────────┬──────────────────────┬─────────────────┬──────────────────┬──────────────┬──────────────────────┬──────
──────────────┐
| GroupId     | PoolName             | ErasureSpecies | OperatingStatus | VDisks_TOTAL | BlobDepotId          | Dec
ommitStatus |
├─────────────┼──────────────────────┼─────────────────┼──────────────────┼──────────────┼──────────────────────┼──────
──────────────┤
| 2181038080 | /Root/database:ssd | mirror-3-dc    | FULL            | 9            | 0                    | NON
E           |
| 2181038081 | /Root/database:ssd | mirror-3-dc    | FULL            | 0            | 72075186224038889    | DON
E           |
| 2181038082 | /Root/database:ssd | mirror-3-dc    | FULL            | 9            | 0                    | NON
E           |
| 2181038083 | /Root/database:ssd | mirror-3-dc    | FULL            | 9            | 0                    | NON
E           |
| 2181038084 | /Root/database:ssd | mirror-3-dc    | FULL            | 9            | 0                    | NON
E           |
| 2181038085 | /Root/database:ssd | mirror-3-dc    | FULL            | 9            | 0                    | NON
E           |
| 2181038086 | /Root/database:ssd | mirror-3-dc    | FULL            | 9            | 0                    | NON
E           |
| 2181038087 | /Root/database:ssd | mirror-3-dc    | FULL            | 9            | 0                    | NON
E           |
└─────────────┴──────────────────────┴─────────────────┴──────────────────┴──────────────┴──────────────────────┴──────
──────────────┘
```

Обратите внимание, что в приведённом выше примере вывода для группы `2181038081` изменилось количество VDisks (оно стало равно 0), появилась id виртуальной группы в поле `BlobDepotId`, а статус декомиссии (`DecommitStatus`) изменился на

`DONE`. Это отражает успешное завершение процедуры декомиссии для данной группы.

Перевоз VDisk'ов

Иногда бывает нужно освободить блочное устройство для замены оборудования. Или один из VDisk'ов интенсивно используется и влияет на производительность остальных VDisk'ов, находящихся на том же PDisk'е. В этих случаях необходимо выполнить перевоз VDisk'ов.

Перевезти один из VDisk'ов с блочного устройства

Получите список идентификаторов VDisk'ов с помощью утилиты YDB DSTool:

```
ydb-dstool -e <bs_endpoint> vdisk list --format tsv --columns VDiskId --no-header
```

Чтобы перевезти VDisk'и с блочного устройства, выполните на узле следующие команды:

```
ydb-dstool -e <bs_endpoint> vdisk evict --vdisk-ids VDISK_ID1 ... VDISK_IDN
ydbd admin bs config invoke --proto 'Command { ReassignGroupDisk { GroupId: <ID группы хранения> GroupGenerat
ion: <Поколение группы хранения> FailRealmIdx: <FailRealm> FailDomainIdx: <FailDomain> VDiskIdx: <Номер слота
> } }'
```

- `VDISK_ID1 ... VDISK_IDN` — список идентификаторов VDisk'ов, вида `[GroupId:GroupGeneration:FailRealmIdx:FailDomainIdx:VDiskIdx]`. Идентификаторы разделяются пробелами.
- `GroupId` — ID группы хранения.
- `GroupGeneration` — поколение группы хранения.
- `FailRealmIdx` — номер fail realm.
- `FailDomainIdx` — номер fail domain.
- `VDiskIdx` — номер слота.

Перевезти VDisk'и со сломанного/отсутствующего устройства

В случае, если SelfHeal выключен или не перевозит VDisk'и автоматически, перевоз нужно выполнить вручную:

1. Откройте мониторинг и убедитесь, что VDisk в нерабочем состоянии.
2. Получите `[NodeId:PDiskId]` нужного диска с помощью утилиты YDB DSTool:

   ```
   ydb-dstool -e <bs_endpoint> vdisk list | fgrep VDISK_ID
   ```

3. Перевезите VDisk:

   ```
   ydb-dstool -e <bs_endpoint> pdisk set --status BROKEN --pdisk-ids "[NodeId:PDiskId]"
   ```

Вернуть PDisk после развоза

Чтобы вернуть PDisk после развоза:

1. Откройте мониторинг и убедитесь, что PDisk в рабочем состоянии.
2. Получите `[NodeId:PDiskId]` нужного диска с помощью утилиты YDB DSTool:

   ```
   ydb-dstool -e <bs_endpoint> pdisk list
   ```

3. Верните PDisk:

   ```
   ydb-dstool -e <bs_endpoint> pdisk set --status ACTIVE --pdisk-ids "[NodeId:PDiskId]"
   ```

Предотвращение выхода за модель отказа

Отказал один VDisk в группе хранения

При включенном SelfHeal данная ситуация считается штатной. SelfHeal перевезет VDisk за указанное в настройках время, после чего начнется процесс репликации данных на другом PDisk'е.

Если же SelfHeal отключен, то перевоз VDisk'а потребуется провести вручную. Для это перед перевозом требуется убедиться в отказе **только одного** VDisk'а в группе хранения.
После чего воспользоваться инструкцией.

Отказало более одного VDisk'а в одной группе хранения без превышения модели отказа

При таком отказе потери данных не происходит, система сохраняет работоспособность, успешно выполняются запросы на чтение и запись. Возможно падение производительности вызванное переносом нагрузки обрабатываемой отказавшими дисками на оставшиеся в строю.

При отказе более одного VDisk'а в группе SelfHeal перестает перевозить VDisk'и. В случае если количество отказавших VDisk'ов находится на грани модели отказа, то рекомендуется восстановить работоспособность хотя бы одного из VDisk'ов, прежде чем начинать перевоз VDisk'ов. Также возможно потребуется более осторожно перевозить VDisk'и по одному.

Количество отказавших VDisk'ов превысило модель отказа

Доступность и работоспособность системы может быть нарушена. Необходимо восстановить работоспособность хотя бы одного из VDisk'ов без потери хранившихся на нем данных.

Балансировка нагрузки на диски

В YDB балансировать нагрузку на диски можно двумя способами:

- распределить нагрузку равномерно по группам;
- распределить VDisk'и равномерно по устройствам.

Распределить нагрузку равномерно по группам

На странице Hive web-viewer, в нижней части экрана есть кнопка "Reassign Groups".

Распределить VDisk'и равномерно по устройствам

В результате некоторых операций, например декомиссии, VDisk'и могут быть распределены на блочных устройствах неравномерно. Улучшить равномерность распределения можно одним из способов:

- Перевезти VDisk'и по одному с перегруженных устройств.
- Воспользоваться утилитой YDB DSTool. Следующая команда перевезет VDisk с перегруженного устройства на менее нагруженное:

  ```
  ydb-dstool -e <bs_endpoint> cluster balance
  ```

  `<bs_endpoint>` - эндпоинт произвольного узла хранения кластера.
  Команда перевозит не более одного VDisk'а за один запуск.

Изменение количества слотов для VDisk'ов на PDisk'ах

Для добавления групп хранения требуется переопределить конфиг хоста, увеличив для него количество слотов на PDisk'ах. Это можно осуществить, проделав следующие шаги:

1. Получить текущую конфигурацию кластера:

   ```
   ydb -e <endpoint> admin cluster config fetch > config.yaml
   ```

   `<endpoint>` - grpc/grpcs эндпоинт произвольного узла кластера.
2. Добавить (или изменить) поле `expected_slot_count` для нужного устройства `drive` в секции `host_configs`.

   Примерный вид секции конфигурации:

   ```
   config:
   host_configs:
   - host_config_id: 1
       drive:
       - path: <path_to_device>
         type: <type>
         expected_slot_count: <number>
       - path: ...
   - host_config_id: 2
       ...
   ```

3. Загрузить обновленный конфигурационный файл на кластер:

   ```
   ydb -e endpoint admin cluster config replace -f config.yaml
   ```

Освобождение места на физических устройствах

При исчерпании места на дисках база данных может отвечать ошибками на все запросы. Для сохранения работоспособности рекомендуется удалить часть данных или расширить кластер блочными устройствами.

Ниже приведены инструкции которые могут помочь добавить или освободить место на дисках.

Дефрагментировать VDisk

В ходе эксплуатации возникает внутренняя фрагментация VDisk'а. Узнать степень фрагментации можно на странице мониторинга VDisk'а. Дефрагментация VDisk'ов, фрагментированных на 20 и менее процентов не рекомендуется.

По модели отказа кластер переживает потерю двух VDisk'ов одной группы без потери данных. Если в группе все VDisk'и работоспособны, нет VDisk'ов в состоянии ошибки или репликации, удаление данных с одного из VDisk'ов приведет к восстановлению VDisk'ом данных в компактном виде. Следует понимать, что избыточность хранения данных будет снижена до завершения автоматической репликации данных.

В процессе репликации данных нагрузка на все VDisk'и группы будет увеличена, возможно ухудшение времени отклика.

1. Посмотреть коэффициент фрагментации на странице VDisk'а во вьювере.

   Если значение превышает 20%, то дефрагментация позволит освободить место на VDisk'е.
2. Проверить состояние группы, в которую входит VDisk. В группе не должно быть недоступных VDisk'ов, VDisk'ов в состоянии ошибки или репликации.

   Посмотреть состояние группы можно во вьювере.
3. Выполнить команду wipe для VDisk'а.

   Все данные, хранимые VDisk'ом, будут необратимо удалены, после чего VDisk начнет восстанавливать данные, читая их с остальных VDisk'ов группы.

   ```
   ydb-dstool -e <bs_endpoint> vdisk wipe --vdisk-ids <VDiskId> --run
   ```

   `<bs_enpoint>` - эндпоинт произвольного узла хранения кластера.
   `<VDiskId>` - идентификатор VDisk-а, в формате [GroupId:_:FailRealm:FailDomain:VDiskIdx].
   Узнать `VDiskId` можно в веб-интерейсе кластера, во вкладке Storage. Необходимо выбрать нужный VDisk и перейти на его страницу в Developer UI.
   Команда допускает множественное указание VDisk-ов, в таком случае их ID вводятся через пробел, в кавычках.

В случае заканчивающегося места на блочном устройстве можно применить дефрагментацию на все устройство.

1. Проверить состояние групп в кластере. Не должно быть проблемных групп, которые находятся на том же узле, что и проблемное устройство.
2. Зайти по ssh на узел, где находится этот устройство
3. Проверить, можно ли перезапустить процесс.
4. Остановить процесс

   ```
   sudo systemctl stop ydbd
   ```
5. Форматировать устройство

   ```
   sudo ydbd admin blobstorage disk obliterate <путь до партлейбла устройства>
   ```
6. Запустить процесс

   ```
   sudo systemctl start ydbd
   ```

Перевоз отдельных VDisk'ов с заполненных устройств

Если дефрагментация не помогает освободить место на устройстве, то можно перевозить отдельные VDisk'и.

Блобовница

Блобовница расширяет функциональность подсистемы хранения, добавляя функциональность виртуальной группы.

Виртуальная группа наравне с физической группой является единицей отказоустойчивости подсистемы хранения в кластере, однако свои данные виртуальная группа хранит в других группах (в отличие от физической, которая данные хранит на VDisk'ax).

Такой способ хранения данных даёт возможность более гибко использовать подсистему хранилища YDB, в частности:

- использовать более "тяжёлые" таблетки в условиях, когда размер одной физической группы ограничен;
- предоставлять таблеткам более широкую полосу на запись за счёт балансировки записи сразу по всем группам, поверх которых запущена блобовница;
- обеспечивать прозрачную для клиентских таблеток миграцию данных между разными группами.

Блобовница также может использоваться для декомиссии групп, то есть для удаления VDisk'а внутри физической группы с сохранением всех данных, которые были записаны в эту группу. В таком сценарии использования данные из физической группы прозрачным для клиента образом переносятся в виртуальную, затем все VDisk'и физической группы удаляются для освобождения занимаемых ими ресурсов.

Режим виртуальной группы

В режиме виртуальной группы блобовница позволяет объединять несколько групп в единое пространство для хранения большого объёма данных. Предусмотрена балансировка по занятому месту, а также увеличение пропускной способности за счёт распределения записей по разным группам. При этом возможен фоновый (при этом полностью прозрачный для клиента) перенос данных.

Виртуальные группы создаются также внутри Storage Pool'ов, как и физические группы, но для виртуальных групп рекомендуется создавать отдельный пул командой `dstool pool create virtual`. Эти пулы могут быть указаны в Hive для создания других таблеток. Однако во избежание ухудшения latency каналы 0 и 1 таблеток рекомендуется селить на физические группы, а на виртуальные группы с блобовницей селить только каналы данных.

Как запускать

Виртуальная группа создаётся через BS_CONTROLLER путём передачи специальной команды. Команда создания виртуальной группы идемпотентна, поэтому для исключения создания лишних блобовниц каждой виртуальной группе ставится в соответствие название. Название должно быть уникально в рамках всего кластера. В случае повторного исполнения команды вернётся ошибка с заполненным полем `Already: true` и указанием номера уже созданной ранее виртуальной группы.

```
dstool -e ... --direct group virtual create --name vg1 vg2 --hive-id=72057594037968897 --storage-pool-name=/R
oot:virtual --log-channel-sp=/Root:ssd --data-channel-sp=/Root:ssd*8
```

Параметры командной строки:

- --name уникальное имя для виртуальной группы (или нескольких виртуальных групп со схожими параметрами);
- --hive-id=N номер таблетки Hive, которая будет управлять данной блобовницей; необходимо указывать Hive того тенанта, внутри которого запускается блобовница;
- --storage-pool-name=POOL_NAME название Storage Pool, внутри которого необходимо создать блобовницу;
- --storage-pool-id=BOX:POOL альтернатива `--storage-pool-name`, в которой можно указать явный числовой идентификатор пула;
- --log-channel-sp=POOL_NAME название пула, в котором будет размещён канал 0 таблетки блобовницы;
- --snapshot-channel-sp=POOL_NAME название пула, в котором будет размещён кана 0 таблетки блобовницы; если не указан, то используется значение из --log-channel-sp;
- --data-channel-sp=POOL_NAME[*COUNT] название пула, в котором размещаются каналы данных; если указан параметр COUNT (после знака "звёздочка"), то создаётся COUNT каналов данных в указанном пуле; рекомендуется заводить большое количество каналов данных для блобовницы в режиме виртуальной группы (64..250), чтобы наиболее эффективно использовать хранилище;
- --wait дождаться завершения создания блобовниц; если эта опция не указана, то команда завершается сразу же после ответа на запрос о создании блобовницы, не дожидаясь создания и запуска самих таблеток.

Как проверить, что всё запустилось

Посмотреть результат создания виртуальной группы можно следующими способами:

- через страницу мониторинга BS_CONTROLLER;
- через команду `dstool group list --virtual-groups-only`.

В обоих случаях контролировать создание нужно через поле VirtualGroupName, которое должно совпадать с тем, что было передано в параметре --name. Если команда `dstool group virtual create` завершилась успехом, виртуальная группа безусловно появляется в списке групп, но при этом поле VirtualGroupState может принимать одно из следующих значений:

- NEW — группа ожидает инициализации (идёт создание таблетки через Hive, её настройка и запуск);
- WORKING — группа создана и работает, готова выполнять пользовательские запросы;
- CREATE_FAILED — при создании группы случилась ошибка, текстовое описание которой можно увидеть в поле ErrorReason.

```
$ dstool --cluster=$CLUSTER --direct group list --virtual-groups-only
┌─────────────┬──────────────┬──────────────┬────────────┬─────────────────┬──────────────────┬──────────────┐
──────────────────────┬──────────────┬──────────────┬──────────────┬──────────────────┐
| GroupId     | BoxId:PoolId | PoolName     | Generation | ErasureSpecies | OperatingStatus | VDisks_TOTAL |
VirtualGroupState | VirtualGroupName | BlobDepotId       | ErrorReason | DecommitStatus |
├─────────────┼──────────────┼──────────────┼────────────┼─────────────────┼──────────────────┼──────────────┼─
──────────────────────┼──────────────┼──────────────┼──────────────┼──────────────────┤
| 4261412864 | [1:2]        | /Root:virtual | 0          | none           | DISINTEGRATED   | 0            |
WORKING           | vg1              | 72075186224037888 |             | NONE           |
| 4261412865 | [1:2]        | /Root:virtual | 0          | none           | DISINTEGRATED   | 0            |
WORKING           | vg2              | 72075186224037890 |             | NONE           |
| 4261412866 | [1:2]        | /Root:virtual | 0          | none           | DISINTEGRATED   | 0            |
```

```
WORKING           | vg3              | 72075186224037889 |             | NONE           |
| 4261412867 | [1:2]        | /Root:virtual | 0           | none           | DISINTEGRATED   | 0           |
WORKING           | vg4              | 72075186224037891 |             | NONE           |
└──────────────┴──────────────┴──────────────┴──────────────┴──────────────┴──────────────┴──────────────┴──
──────────────┴──────────────┴──────────────┴──────────────┴──────────────┘
```

Устройство

Блобовница является таблеткой, которая в дополнение к двум системным каналам (0 и 1) содержит также набор дополнительных каналов, в которых хранятся сами данные, записанные в блобовницу. Клиентские данные записываются в эти дополнительные каналы.

Блобовница как таблетка может быть запущена на любом узле кластера.

Когда блобовница работает в режиме виртуальной группы, то для доступа к ней используются агенты (BlobDepotAgent). Это акторы, выполняющие функции, аналогичные DS proxy — они запускаются на каждом узле, который использует виртуальную группу с блобовницей. Эти же акторы преобразуют запросы к хранилищу в команды для блобовницы и обеспечивают обмен данными с ней.

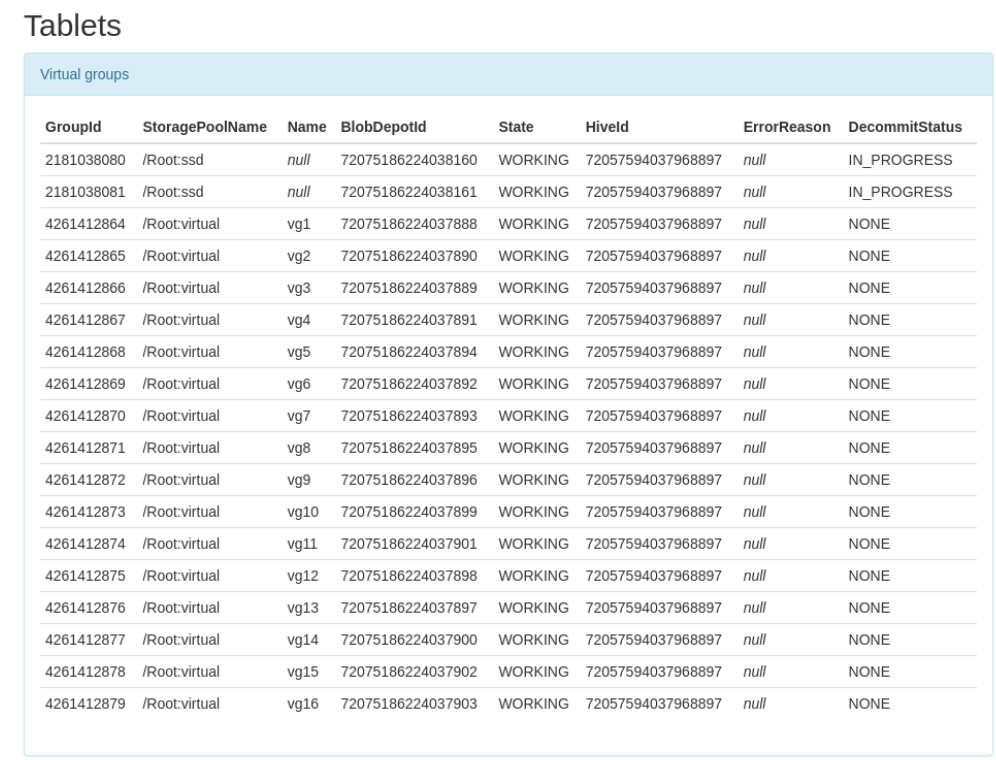
Механизмы диагностики

Для диагностики работоспособности блобовницы предусмотрен следующий ряд механизмов:

- страница мониторинга BS_CONTROLLER;
- страница мониторинга блобовницы;
- internal viewer;
- журнал событий;
- графики.

Страница мониторинга BS_CONTROLLER

На странице мониторинга BS_CONTROLLER существует специальная вкладка Virtual groups, в которой показаны все группы, которые используют блобовницу:



Tablets

Virtual groups

GroupId	StoragePoolName	Name	BlobDepotId	State	HiveId	ErrorReason	DecommitStatus
2181038080	/Root:ssd	*null*	72075186224038160	WORKING	72057594037968897	*null*	IN_PROGRESS
2181038081	/Root:ssd	*null*	72075186224038161	WORKING	72057594037968897	*null*	IN_PROGRESS
4261412864	/Root:virtual	vg1	72075186224037888	WORKING	72057594037968897	*null*	NONE
4261412865	/Root:virtual	vg2	72075186224037890	WORKING	72057594037968897	*null*	NONE
4261412866	/Root:virtual	vg3	72075186224037889	WORKING	72057594037968897	*null*	NONE
4261412867	/Root:virtual	vg4	72075186224037891	WORKING	72057594037968897	*null*	NONE
4261412868	/Root:virtual	vg5	72075186224037894	WORKING	72057594037968897	*null*	NONE
4261412869	/Root:virtual	vg6	72075186224037892	WORKING	72057594037968897	*null*	NONE
4261412870	/Root:virtual	vg7	72075186224037893	WORKING	72057594037968897	*null*	NONE
4261412871	/Root:virtual	vg8	72075186224037895	WORKING	72057594037968897	*null*	NONE
4261412872	/Root:virtual	vg9	72075186224037896	WORKING	72057594037968897	*null*	NONE
4261412873	/Root:virtual	vg10	72075186224037899	WORKING	72057594037968897	*null*	NONE
4261412874	/Root:virtual	vg11	72075186224037901	WORKING	72057594037968897	*null*	NONE
4261412875	/Root:virtual	vg12	72075186224037898	WORKING	72057594037968897	*null*	NONE
4261412876	/Root:virtual	vg13	72075186224037897	WORKING	72057594037968897	*null*	NONE
4261412877	/Root:virtual	vg14	72075186224037900	WORKING	72057594037968897	*null*	NONE
4261412878	/Root:virtual	vg15	72075186224037902	WORKING	72057594037968897	*null*	NONE
4261412879	/Root:virtual	vg16	72075186224037903	WORKING	72057594037968897	*null*	NONE

В таблице предусмотрены следующие столбцы:

Поле	Описание
GroupId	Номер группы.
StoragePoolName	Название пула, в котором находится группа.
Name	Название виртуальной группы; оно уникальное на весь кластер. Для декомиссуемых групп здесь будет null.
BlobDepotId	Номер таблетки блобовницы, которая отвечает за обслуживание данной группы.
State	Состояние блобовницы; может быть NEW, WORKING, CREATED_FAILED.
HiveId	Номер таблетки Hive, внутри которой была создана указанная блобовница.
ErrorReason	При состоянии CREATE_FAILED содержит текстовое описание причины ошибки создания.
DecommitStatus	Состояние декомиссии группы; может быть NONE, PENDING, IN_PROGRESS, DONE.

Страница мониторинга блобовницы

На странице мониторинга блобовницы представлены основные параметры работы таблетки, которые сгруппированы по вкладкам, доступным по ссылке "Contained data":

- data
- refcount
- trash
- barriers
- blocks
- storage

Кроме этого, на основной странице приведены краткие сведения о состоянии блобовницы:

Contained data

Stats

Parameter	Value
Data, bytes	21.83 GiB
Trash in flight, bytes	0 B
Trash pending, bytes	0 B
Data in GroupId# 2181038088, bytes	826.47 MiB
Data in GroupId# 2181038089, bytes	816.17 MiB
Data in GroupId# 2181038090, bytes	736.13 MiB
Data in GroupId# 2181038092, bytes	797.28 MiB
Data in GroupId# 2181038093, bytes	622.74 MiB
Data in GroupId# 2181038094, bytes	734.44 MiB
Data in GroupId# 2181038095, bytes	727.78 MiB
Data in GroupId# 2181038096, bytes	843.88 MiB
Data in GroupId# 2181038098, bytes	724.19 MiB
Data in GroupId# 2181038099, bytes	757.32 MiB
Data in GroupId# 2181038100, bytes	725.58 MiB
Data in GroupId# 2181038101, bytes	673.08 MiB
Data in GroupId# 2181038102, bytes	715.34 MiB
Data in GroupId# 2181038104, bytes	748.80 MiB
Data in GroupId# 2181038105, bytes	710.01 MiB
Data in GroupId# 2181038106, bytes	770.66 MiB
Data in GroupId# 2181038107, bytes	705.25 MiB
Data in GroupId# 2181038108, bytes	898.86 MiB
Data in GroupId# 2181038110, bytes	725.41 MiB
Data in GroupId# 2181038111, bytes	788.05 MiB
Data in GroupId# 2181038112, bytes	639.93 MiB
Data in GroupId# 2181038113, bytes	720.35 MiB
Data in GroupId# 2181038114, bytes	657.61 MiB

В данной таблице приведены следующие данные:

- Data, bytes — количество сохранённых байт данных (TotalStoredDataSize).
- Trash in flight, bytes — количество байт ненужных данных, которые ожидают завершения транзакций, чтобы стать мусором (InFlightTrashSize).
- Trash pending, bytes — количество байт мусора, которые ещё не переданы в сборку мусора (TotalStoredTrashSize).
- Data in GroupId# XXX, bytes — количество байт данных в группе XXX (как полезные данные, так и ещё не собранный мусор).

Data

Main

Parameter	Value
Loaded	true
Last assimilated blob id	
Data size, number of keys	284398
RefCount size, number of blobs	284398
Total stored data size, bytes	21.83 GiB
Keys made certain, number of keys	0

Uncertainty resolver

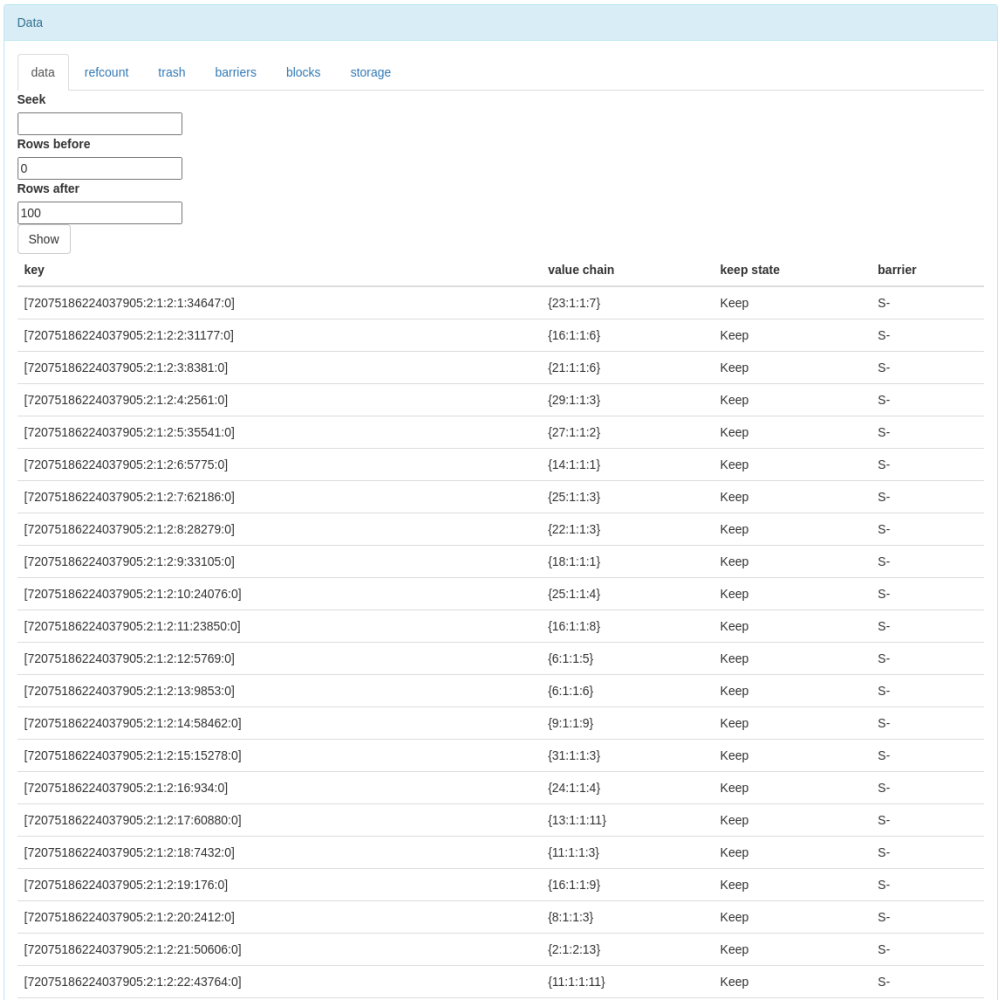
Parameter	Value
Keys queried	0
Gets issued	0
Keys resolved	0
Keys unresolved	0
Keys dropped	498590
Keys being processed	0
Blobs in flight	0

Назначения параметров следующие:

- Loaded — булевое значение, показывающее, загружены ли все метаданные из локальной базы таблетки в память.
- Last assimilated blob id — BlobId последнего вычитанного блоба (копирование метаданных при декомиссии).
- Data size, number of keys — количество сохранённых ключей данных.
- RefCount size, number of blobs — количество уникальных блобов данных, которые хранит блобовница в своём пространстве имён.
- Total stored data size, bytes — аналогично "Data, bytes" из таблицы выше.
- Keys made certain, number of keys — количество недописанных ключей, которые были затем подтверждены чтением.

Раздел "Uncertainty resolver" относится к компоненту, который работает с записанными, но не подтверждённым в блобовницу данными.

data



key	value chain	keep state	barrier
[72075186224037905:2:1:2:1:34647:0]	{23:1:1:7}	Keep	S-
[72075186224037905:2:1:2:2:31177:0]	{16:1:1:6}	Keep	S-
[72075186224037905:2:1:2:3:8381:0]	{21:1:1:6}	Keep	S-
[72075186224037905:2:1:2:4:2561:0]	{29:1:1:3}	Keep	S-
[72075186224037905:2:1:2:5:35541:0]	{27:1:1:2}	Keep	S-
[72075186224037905:2:1:2:6:5775:0]	{14:1:1:1}	Keep	S-
[72075186224037905:2:1:2:7:62186:0]	{25:1:1:3}	Keep	S-
[72075186224037905:2:1:2:8:28279:0]	{22:1:1:3}	Keep	S-
[72075186224037905:2:1:2:9:33105:0]	{18:1:1:1}	Keep	S-
[72075186224037905:2:1:2:10:24076:0]	{25:1:1:4}	Keep	S-
[72075186224037905:2:1:2:11:23850:0]	{16:1:1:8}	Keep	S-
[72075186224037905:2:1:2:12:5769:0]	{6:1:1:5}	Keep	S-
[72075186224037905:2:1:2:13:9853:0]	{6:1:1:6}	Keep	S-
[72075186224037905:2:1:2:14:58462:0]	{9:1:1:9}	Keep	S-
[72075186224037905:2:1:2:15:15278:0]	{31:1:1:3}	Keep	S-
[72075186224037905:2:1:2:16:934:0]	{24:1:1:4}	Keep	S-
[72075186224037905:2:1:2:17:60880:0]	{13:1:1:11}	Keep	S-
[72075186224037905:2:1:2:18:7432:0]	{11:1:1:3}	Keep	S-
[72075186224037905:2:1:2:19:176:0]	{16:1:1:9}	Keep	S-
[72075186224037905:2:1:2:20:2412:0]	{8:1:1:3}	Keep	S-
[72075186224037905:2:1:2:21:50606:0]	{2:1:2:13}	Keep	S-
[72075186224037905:2:1:2:22:43764:0]	{11:1:1:11}	Keep	S-

Таблица data содержит следующие столбцы:

- key — идентификатор ключа (BlobId в пространстве имён клиента);
- value chain — значение ключа, которое формируется конкатенацией фрагментов блобов из пространства имён блобовницы (в этом поле перечислены эти самые блобы);
- keep state — значение keep-флагов для данного блоба глазами клиента (Default, Keep, DoNotKeep);
- barrier — поле, показывающее, под какой барьер попадает данный блоб (S — под soft-барьер, H — под hard-барьер; фактически H никогда не бывает, т.к. блобы синхронно удаляются из таблицы в момент выставления hard-барьера).

Учитывая потенциально большой размер таблицы, на странице мониторинга показывается только её часть. Для поиска нужного блоба можно заполнить поле "seek", введя в нём BlobId искомого блоба, а затем указать число интересующих строк до этого блоба и после него и нажать кнопку "Show".

refcount

Data

data | refcount | trash | barriers | blocks | storage

blob id	refcount
[72075186224037894:1:116:7:33:30016:0]	1
[72075186224037894:1:110:26:1057:28880:0]	1
[72075186224037894:1:361:2:705:39373:0]	1
[72075186224037894:1:148:30:2097:5588:0]	1
[72075186224037894:1:503:20:81:54961:0]	1
[72075186224037894:1:442:16:817:48656:0]	1
[72075186224037894:1:162:8:705:17513:0]	1
[72075186224037894:1:284:7:577:63292:0]	1
[72075186224037894:1:462:13:337:4355:0]	1
[72075186224037894:1:401:9:1073:6248:0]	1
[72075186224037894:1:120:26:17:54953:0]	1
[72075186224037894:1:444:19:145:13229:0]	1
[72075186224037894:1:488:4:161:55863:0]	1
[72075186224037894:1:584:12:209:47719:0]	1
[72075186224037894:1:260:19:81:51993:0]	1
[72075186224037894:1:228:6:897:58958:0]	1
[72075186224037894:1:367:24:17:28839:0]	1
[72075186224037894:1:332:11:577:52762:0]	1
[72075186224037894:1:492:23:145:58541:0]	1
[72075186224037894:1:168:30:17:56918:0]	1
[72075186224037894:1:527:27:929:50666:0]	1
[72075186224037894:1:477:7:225:60133:0]	1
[72075186224037894:1:573:15:273:64158:0]	1
[72075186224037894:1:479:23:881:34206:0]	1
[72075186224037894:1:100:7:465:17313:0]	1
[72075186224037894:1:599:6:97:75:0]	1
[72075186224037894:1:400:12:97:29934:0]	1

Таблица refcount содержит два столбца: "blob id" и "refcount". Blob id — это идентификатор хранимого блоба, который записан от имени блобовницы в хранилище. Refcount — количество ссылок на этот блоб из таблицы data (из столбца value chain).

Метрика TotalStoredDataSize формируется из суммы размеров всех блобов в этой таблице, каждый из которых учитывается ровно один раз, без учёта поля refcount.

trash

Data

data | refcount | trash | barriers | blocks | storage

group id	blob id	in flight
2181038119	[72075186224037894:1:602:28:81:75:0]	
2181038119	[72075186224037894:1:602:28:129:75:0]	
2181038119	[72075186224037894:1:602:28:385:71:0]	
2181038119	[72075186224037894:1:602:28:401:75:0]	
2181038119	[72075186224037894:1:602:28:417:75:0]	
2181038119	[72075186224037894:1:602:28:865:75:0]	
2181038119	[72075186224037894:1:602:28:881:75:0]	
2181038106	[72075186224037894:1:597:17:81:75:0]	
2181038106	[72075186224037894:1:597:17:97:75:0]	
2181038106	[72075186224037894:1:597:17:433:75:0]	
2181038100	[72075186224037894:1:603:12:49:75:0]	
2181038099	[72075186224037894:1:607:11:113:75:0]	
2181038099	[72075186224037894:1:607:11:401:75:0]	
2181038099	[72075186224037894:1:607:11:449:75:0]	
2181038118	[72075186224037894:1:617:27:625:75:0]	
2181038120	[72075186224037894:1:607:29:113:75:0]	
2181038123	[72075186224037894:1:583:31:193:75:0]	
2181038123	[72075186224037894:1:583:31:449:77:0]	
2181038093	[72075186224037894:1:615:6:1025:71:0]	
2181038108	[72075186224037894:1:609:19:209:71:0]	
2181038108	[72075186224037894:1:609:19:945:75:0]	
2181038092	[72075186224037894:1:612:5:113:75:0]	
2181038092	[72075186224037894:1:612:5:337:75:0]	
2181038092	[72075186224037894:1:612:5:353:75:0]	
2181038092	[72075186224037894:1:612:5:369:75:0]	
2181038092	[72075186224037894:1:612:5:385:75:0]	
2181038116	[72075186224037894:1:618:25:225:75:0]	

Таблица содержит три столбца: "group id", "blob id" и "in flight". Group id — номер группы, в которой сохранён более не нужный блоб. Blob id — идентификатор самого блоба. In flight — признак того, чтоб блоб ещё проходит через транзакцию, только по результатам которой его можно будет передавать в сборщик мусора.

Метрики TotalStoredTrashSize и InFlightTrashSize формируются этой этой таблицы суммированием размеров блобов без признака in flight и с ним, соответственно.

barriers

Data

data | refcount | trash | barriers | blocks | storage

tablet id	channel	soft	hard
72075186224038155	0	2:157 => 2:39350	0:0 => 0:0
72075186224037909	2	2:1 => 1:4294967295	2:0 => 1:4294967295
72075186224038059	2	2:1 => 1:4294967295	2:0 => 1:4294967295
72075186224038050	1	2:160 => 2:40178	0:0 => 0:0
72075186224038021	2	2:1 => 1:4294967295	2:0 => 1:4294967295
72075186224038005	2	2:1 => 1:4294967295	2:0 => 1:4294967295
72075186224038144	1	2:157 => 2:39352	0:0 => 0:0
72075186224038149	2	2:1 => 1:4294967295	2:0 => 1:4294967295
72075186224038091	0	2:157 => 2:39371	0:0 => 0:0
72075186224038000	1	2:157 => 2:39371	0:0 => 0:0
72075186224038085	2	2:1 => 1:4294967295	2:0 => 1:4294967295
72075186224038011	0	3:122 => 3:30448	0:0 => 0:0
72075186224038128	1	2:157 => 2:39372	0:0 => 0:0
72075186224038027	0	2:151 => 2:37872	0:0 => 0:0
72075186224037931	0	2:154 => 2:38678	0:0 => 0:0
72075186224038139	0	2:157 => 2:39374	0:0 => 0:0
72075186224038133	2	2:1 => 1:4294967295	2:0 => 1:4294967295
72075186224038016	1	2:158 => 2:39689	0:0 => 0:0
72075186224037925	2	2:1 => 1:4294967295	2:0 => 1:4294967295
72075186224037915	0	2:155 => 2:38870	0:0 => 0:0
72075186224038080	1	2:155 => 2:38889	0:0 => 0:0
72075186224037905	2	4:1 => 3:4294967295	4:0 => 2:0
72075186224038072	0	2:156 => 2:39187	0:0 => 0:0
72075186224037920	1	2:158 => 2:39692	0:0 => 0:0

Таблица barriers содержит информацию о клиентских барьерах, которые были переданы в блобовницу. Она состоит из столбцов "tablet id" (номер таблетки), "channel" (номер канала, для которого записан барьер), а также значений барьеров: "soft" и "hard". Значение имеет формат gen:counter => collect_gen:collect_step, где gen — номер поколения таблетки, в котором был выставлен данный барьер, counter — порядковый номер команды сборки мусора, collect_gen:collect_step — значение барьера (удаляются все блобы, у которых поколение и шаг внутри поколения меньше либо равны указанному барьеру).

blocks

Data

data refcount trash barriers blocks storage

tablet id	blocked generation
72075186224038011	2
72075186224038128	1
72075186224037925	1
72075186224038072	1
72075186224038016	1
72075186224038133	1
72075186224037931	1
72075186224038050	1
72075186224038021	1
72075186224037905	3
72075186224038080	1
72075186224038139	1
72075186224037909	1
72075186224038085	1
72075186224038027	1
72075186224038144	1
72075186224038000	1
72075186224038059	1
72075186224037915	1
72075186224038149	1
72075186224038091	1
72075186224038005	1
72075186224037920	1
72075186224038155	1

Таблица blocks содержит список блокировок клиентских таблеток и состоит из столбцов "tablet id" (номер таблетки) и "blocked generation" (номер поколения этой таблетки, в котором уже нельзя ничего писать).

storage

Data

data refcount trash barriers blocks storage

group id	bytes stored in current generation	bytes stored total	status flag	free space share
2181038088	972.28 MiB	972.28 MiB	{ Valid }	95.0 %
2181038089	966.29 MiB	966.29 MiB	{ Valid }	92.3 %
2181038090	858.26 MiB	858.26 MiB	{ Valid }	94.8 %
2181038092	951.95 MiB	951.95 MiB	{ Valid }	94.7 %
2181038093	793.19 MiB	793.19 MiB	{ Valid }	92.2 %
2181038094	874.82 MiB	874.82 MiB	{ Valid }	95.0 %
2181038095	847.88 MiB	847.88 MiB	{ Valid }	94.5 %
2181038096	986.85 MiB	986.85 MiB	{ Valid }	92.5 %
2181038098	882.24 MiB	882.24 MiB	{ Valid }	94.7 %
2181038099	919.22 MiB	919.22 MiB	{ Valid }	94.6 %
2181038100	915.49 MiB	915.49 MiB	{ Valid }	92.2 %
2181038101	807.56 MiB	807.56 MiB	{ Valid }	94.5 %
2181038102	877.60 MiB	877.60 MiB	{ Valid }	94.8 %
2181038104	869.11 MiB	869.11 MiB	{ Valid }	92.4 %
2181038105	884.52 MiB	884.52 MiB	{ Valid }	94.6 %
2181038106	898.87 MiB	898.87 MiB	{ Valid }	94.6 %
2181038107	845.60 MiB	845.60 MiB	{ Valid }	92.5 %
2181038108	1.00 GiB	1.00 GiB	{ Valid }	94.8 %
2181038110	871.55 MiB	871.55 MiB	{ Valid }	94.7 %
2181038111	919.71 MiB	919.71 MiB	{ Valid }	92.4 %
2181038112	773.51 MiB	773.51 MiB	{ Valid }	95.0 %
2181038113	833.96 MiB	833.96 MiB	{ Valid }	94.5 %
2181038114	780.37 MiB	780.37 MiB	{ Valid }	92.6 %
2181038116	986.28 MiB	986.28 MiB	{ Valid }	94.7 %
2181038117	852.59 MiB	852.59 MiB	{ Valid }	94.6 %

Таблица storage показывает статистику по хранимым данным для каждой группы, в которой блобовница хранит данные. Эта таблица содержит следующие столбцы:

- group id — номер группы, в которой хранятся данные;
- bytes stored in current generation — объём данных, записанных в эту группу в текущем поколении таблетки (считаются только полезные данные, без учёта мусора);
- bytes stored total — объём всех данных, сохранённых этой блобовницей в указанную группу;
- status flag — цветовые флажки состояния группы;
- free space share — показатель заполненности группы (значение 0 соответствует полностью заполненной по месту группе, 1 — полностью свободной).

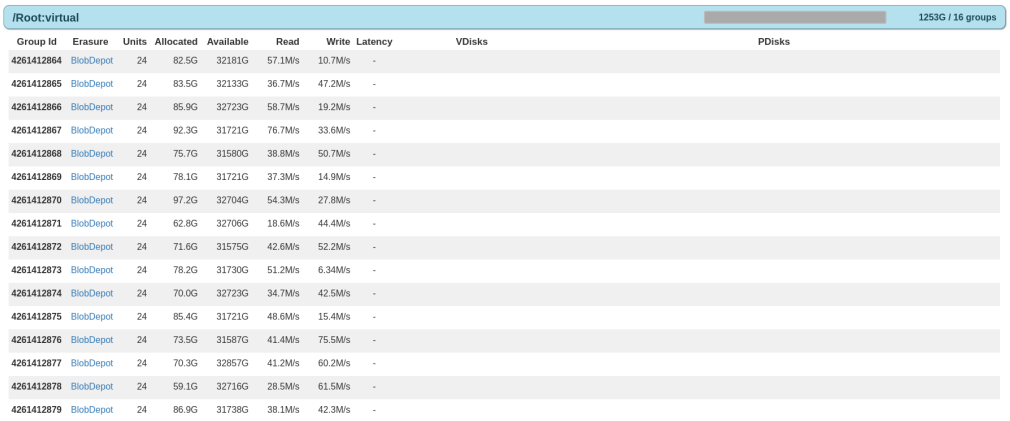
Internal viewer

На странице мониторинга Internal viewer, показанной ниже, блобовницы можно увидеть в разделах Storage и в виде таблеток BD.

В разделе Nodes видны таблетки BD, запущенные на разных узлах системы:



В разделе Storage можно увидеть виртуальные группы, которые работают через блобовницу. Их можно отличить ссылке с текстом BlobDepot в столбце Erasure. Ссылка в этом столбце ведёт на страницу мониторинга таблетки. В остальном виртуальные группы отображаются так же, за исключением того, что у них нет PDisk и VDisk. Однако декомиссуемые группы будут выглядеть так же, как и виртуальные, но иметь PDisk и VDisk до тех пор, пока декомиссия не закончится.



Журнал событий

Таблетка блобовницы пишет события в журнал со следующими названиями компонентов:

- BLOB_DEPOT — компонент таблетки блобовницы.
- BLOB_DEPOT_AGENT — компонент агента блобовницы.
- BLOB_DEPOT_TRACE — специальный компонент для отладочной трассировки всех событий, связанных с данными.

BLOB_DEPOT и BLOB_DEPOT_AGENT выводятся в виде структурированных записей, в которых есть поля, позволяющие идентифицировать блобовницу и группу, которую она обслуживает. Для BLOB_DEPOT это поле Id и оно имеет формат {TabletId:GroupId}:Generation, где TabletId — номер таблетки блобовницы, GroupId — номер группы, которую она обслуживает, Generation — поколение, в котором запущенная блобовница пишет сообщения в журнал. Для BLOB_DEPOT_AGENT это поле названо AgentId и имеет формат {TabletId:GroupId}.

На уровне DEBUG в журнал будут записываться большинство происходящих событий, как на стороне таблетки, так и на стороне агента. Этот режим используется для отладки и не рекомендуется в условиях production-сред из-за большого числа генерируемых событий.

Графики

Каждая таблетка блобовницы выдаёт следующие графики:

График	Тип	Описание
TotalStoredDataSize	простой	Количество сохранённых пользовательских данных нетто (если есть несколько ссылок на один блоб, он учитывается единожды).
TotalStoredTrashSize	простой	Количество байт в мусорных данных, которые уже не нужны, но ещё не переданы в сборку мусора.
InFlightTrashSize	простой	Количество байт мусора, которые ещё ждут подтверждения записи в локальную базу (их ещё нельзя даже начать собирать).

BytesToDecommit	простой	Количество байт данных, которые осталось декомиссовать (в случае, если эта блобовница работает в режиме декомиссии группы).
Puts/Incoming	кумулятивный	Скорость поступления запросов на запись (в штуках в единицу времени).
Puts/Ok	кумулятивный	Количество успешно выполненных запросов на запись.
Puts/Error	кумулятивный	Количество запросов на запись, завершённых с ошибкой.
Decommit/GetBytes	кумулятивный	Скорость чтения данных при декомиссии.
Decommit/PutOkBytes	кумулятивный	Скорость записи данных при декомиссии (учитываются только успешно выполненные записи).

Декомиссия групп

Физические группы являются ценным ресурсом в кластере: группы можно создавать, но их нельзя удалять без удаления использующей их базы, поскольку нет механизма гарантированного вытеснения данных таблеток из группы. При этом количество физических групп определяется размером кластера, а группы нельзя переносить из пула одного тенанта в пул другого из-за использования разных ключей шифрования для разных тенантов.

Это может привести к тому, что для создания новой группы с целью расширения существующей или создания новой БД не хватает ресурсов, а удалить старую группу, чтобы освободить ресурсы, тоже нельзя, поскольку в ней могут быть данные.

Чтобы решить эту проблему, можно завести виртуальную группу с каналами поверх оставшихся групп пула, скопировать в неё данные из физической группы, а затем освободить ресурсы, которые занимает физическая группа. Эту задачу решает процесс декомиссии групп.

Декомиссия группы позволяет удалить избыточные VDisk'и с PDisk'ов, сохраняя при этом данные этой группы. Реализован такой режим через создание блобовницы, которая начинает обслуживать декомиссуемую группу вместо DS proxy. Параллельно с этим блобовница копирует данные из физической декомиссуемой группы. Как только все данные будут скопированы, физические VDisk'и удаляются и ресурсы освобождаются, при этом все данные декомиссованной группы распределяются по другим группам.

Процесс декомиссии полностью прозрачен для таблеток и пользователя и состоит из нескольких этапов:

1. Создание таблетки блобовницы и распространение конфигурации группы с целью блокировки записи на диски физической группы.
2. Копирование метаданных блокировок из физической группы. После этого момента декомиссуемая группа становится доступной для работы. До момента копирования блокировок работа с группой невозможна. Однако этот процесс занимает очень короткий промежуток времени, поэтому практически незаметен для клиента. Запросы, приходящие в этот момент, встают в очередь и ждут завершения этапа.
3. Копирование метаданных барьеров из физической группы.
4. Копирование метаданных блобов из физической группы.
5. Копирование данных блобов из физической группы.
6. Удаление VDisk'ов физической группы.

Стоит ещё раз заметить, что с момента блокировки записи в физическую группу и до момента считывания всех блокировок работа с группой приостанавливается. Время приостановки в условиях штатной работы составляет доли секунды.

Как запускать

Для запуска декомиссии выполняется команда BS_CONTROLLER, в которой нужно указать список декомиссуемых групп, а также номер таблетки Hive, которая будет управлять блобовницами декомиссуемых групп. Также можно указать список пулов, в которых блобовница будет хранить свои данные. Если этот список не указан, то BS_CONTROLLER автоматически выбирает для хранения данных те же пулы, в которых располагаются декомиссуемые группы, а число каналов данных делается равным числу физических групп в этих пулах (но не более 250 штук).

```
dstool -e ... --direct group decommit --group-ids 2181038080 --database=/Root/db1 --wait
```

Параметры командной строки:

- --wait дождаться запуска декомиссии; в случае возникновения ошибки запуска ошибка выводится на экран, декомиссия отменяется автоматически (только при указании этой опции);
- --group-ids GROUP_ID GROUP_ID список групп, для которых можно провести декомиссию;
- --database=DB указать тенант, в котором нужно делать декомиссию (или домен, если декомиссия делается для групп внутри домена);
- --log-channel-sp=POOL_NAME название пула, в котором будет размещён канал 0 таблетки блобовницы;
- --snapshot-channel-sp=POOL_NAME название пула, в котором будет размещён кана 1 таблетки блобовницы; если не указан, то используется значение из --log-channel-sp;
- --data-channel-sp=POOL_NAME[*COUNT] название пула, в котором размещаются каналы данных; если указан параметр COUNT (после знака "звёздочка"), то создаётся COUNT каналов данных в указанном пуле.

Если ни --log-channel-sp, ни --snapshot-channel-sp, ни --data-channel-sp не указаны, то автоматически находится storage pool, которому принадлежит декомиссуемая группа, и в нём заводится нулевой и первый канал блобовницы, а также N каналов данных, где N — число оставшихся физических групп в этом пуле.

Как проверить, что всё запустилось

Посмотреть результат декомиссии можно аналогично созданию виртуальных групп. Для декомиссуемых групп появляется дополнительное поле DecommitStatus, которое может принимать одно из следующих значений:

- NONE — декомиссия для указанной группы не производится;
- PENDING — ожидается декомиссия группы, но ещё не выполняется (создаётся блобовница);
- IN_PROGRESS — производится декомиссия группы (все записи уже попадают в блобовницу, чтения — в блобовницу и в старую группу);
- DONE — декомиссия завершена полностью.

```
$ dstool --cluster=$CLUSTER --direct group list --virtual-groups-only
┌─────────────┬──────────────┬─────────────────┬──────────────┬──────────────────┬──────────────────┬─────────────────┬──
──────────────────┬──────────────────┬──────────────────┬──────────────┐
│ GroupId     │ BoxId:PoolId │ PoolName        │ Generation │ ErasureSpecies │ OperatingStatus │ VDisks_TOTAL │
VirtualGroupState │ VirtualGroupName │ BlobDepotId        │ ErrorReason │ DecommitStatus │
├─────────────┼──────────────┼─────────────────┼──────────────┼──────────────────┼──────────────────┼─────────────────┼──
──────────────────┼──────────────────┼──────────────────┼──────────────┤
│ 2181038080 │ [1:1]        │ /Root:ssd     │ 2          │ block-4-2      │ FULL             │ 8               │
WORKING           │                  │ 72075186224038160 │              │ IN_PROGRESS    │
│ 2181038081 │ [1:1]        │ /Root:ssd     │ 2          │ block-4-2      │ FULL             │ 8               │
WORKING           │                  │ 72075186224038161 │              │ IN_PROGRESS    │
│ 4261412864 │ [1:2]        │ /Root:virtual │ 0          │ none           │ DISINTEGRATED    │ 0               │
WORKING           │ vg1              │ 72075186224037888 │              │ NONE           │
│ 4261412865 │ [1:2]        │ /Root:virtual │ 0          │ none           │ DISINTEGRATED    │ 0               │
WORKING           │ vg2              │ 72075186224037890 │              │ NONE           │
```

```
| 4261412866 | [1:2]        | /Root:virtual | 0          | none           | DISINTEGRATED   | 0           |
WORKING           | vg3              | 72075186224037889 |             | NONE           |
| 4261412867 | [1:2]        | /Root:virtual | 0          | none           | DISINTEGRATED   | 0           |
WORKING           | vg4              | 72075186224037891 |             | NONE           |
└────────────┴──────────────┴───────────────┴────────────┴────────────────┴─────────────────┴─────────────┴─
──────────────────┴──────────────────┴──────────────────┴──────────────┴──────────────────┘
```

Как оценить прогресс

Для оценки времени и прогресса декомиссии предусмотрены графики, по которым можно понять:

- идёт ли декомиссия (Decommit/GetBytes);
- проходит ли запись данных (Decommit/PutOkBytes);
- сколько данных осталось декомиссовать (BytesToDecommit).

Если всё выполняется успешно, то скорость Decommit/GetBytes примерно соответствует Decommit/PutOkBytes. Несущественные расхождения допустимы вследствие того, что декомиссуемые данные могут устаревать и удаляться таблеткой, которая хранит данные в ней.

Для оценки оставшегося времени декомиссии достаточно разделить BytesToDecommit на среднюю скорость Decommit/PutOkBytes.

Развёртывание YDB с функцией Federated Query

Важно

Данная функциональность находится в режиме "Experimental".

Общая схема инсталляции

YDB может выполнять федеративные запросы ко внешним источникам (например, объектным хранилищам или реляционным СУБД) без необходимости перемещения их данных непосредственно в YDB. В данном разделе мы рассмотрим изменения, которые необходимо внести в конфигурацию YDB и окружающую инфраструктуру для включения функциональности федеративных запросов.

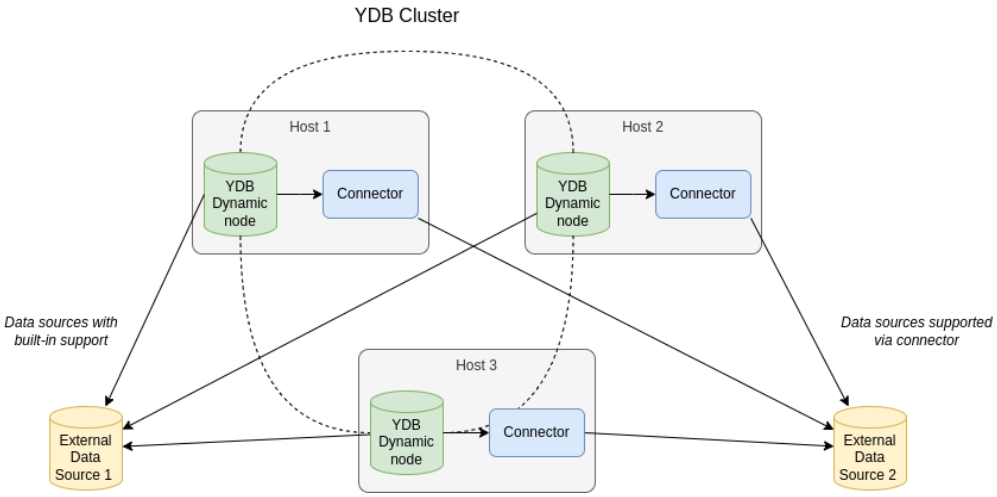
Примечание

Для организации доступа к некоторым из источников данных требуется развёртывание специального микросервиса - коннектора. Ознакомьтесь с перечнем поддерживаемых источников, чтобы понять, требуется ли вам установка коннектора.

Кластер YDB и внешние источники данных в варианте production-инсталляции должны развёртываться на разных физических или виртуальных серверах, в том числе в облаках. Если для доступа к определённому источнику требуется развёртывание коннектора, это необходимо сделать на тех же серверах, на которых развёрнуты динамические узлы YDB. Иными словами, на каждый процесс `ydbd`, работающий в режиме динамического узла, должен приходиться один локальный процесс коннектора.

При этом должны выполняться следующие требования:

- внешний источник данных должен быть доступен по сети для запросов со стороны YDB или со стороны коннектора (при его наличии);
- коннектор должен быть доступен по сети для запросов со стороны YDB (что достигается тривиальным образом благодаря работе этих процессов на одном и том же хосте).



Примечание

В настоящее время мы не поддерживаем развёртывание коннектора в Kubernetes, но планируем добавить её в ближайшем будущем.

Пошаговое руководство

1. Выполните шаги инструкции по развёртыванию динамического узла YDB до подготовки конфигурационных файлов включительно.
2. Если для доступа к нужному вам источнику требуется развернуть коннектор, сделайте это согласно инструкции.
3. В конфигурационном файле YDB в секции `feature_flags` включите флаг `enable_external_data_sources`:

```
feature_flags:
  enable_external_data_sources: true
```

4. В конфигурационный файл YDB добавьте настройки внешних источников данных.

Без использования коннектора

```
query_service_config:
  generic:
    default_settings:
    - name: UsePredicatePushdown
      value: "true"
  all_external_data_sources_are_available: false
  available_external_data_sources:
  - ObjectStorage
```

С использованием коннектора

```
query_service_config:
  generic:
    connector:
      endpoint:
        host: localhost                  # имя хоста, где развернут коннектор
        port: 2130                       # номер порта коннектора
      use_ssl: false                     # флаг, включающий шифрование соединений
      ssl_ca_crt: "/opt/ydb/certs/ca.crt" # путь к сертификату CA
    default_settings:
    - name: UsePredicatePushdown
      value: "true"
  all_external_data_sources_are_available: false
  available_external_data_sources:
  - ClickHouse
  - MySQL
```

5. Продолжайте развёртывание динамического узла YDB по инструкции.

Развёртывание коннекторов ко внешним источникам данных

> **Важно**
>
> Данная функциональность находится в режиме "Experimental".

Коннекторы - специальные микросервисы, предоставляющие YDB универсальную абстракцию доступа ко внешним источникам данных. Коннекторы выступают в качестве точек расширения системы обработки федеративных запросов YDB. В данном руководстве мы рассмотрим особенности развёртывания коннекторов в режиме on-premise.

fq-connector-go

Коннектор `fq-connector-go` реализован на языке Go; его исходный код размещён на GitHub. Он обеспечивает доступ к следующим источникам данных:

- ClickHouse
- Greenplum
- Microsoft SQL Server
- MySQL
- PostgreSQL
- YDB

Коннектор может быть установлен с помощью бинарного дистрибутива или с помощью Docker-образа.

Запуск из бинарного дистрибутива

Для установки коннектора на физический или виртуальный Linux-сервер без средств контейнерной виртуализации используйте бинарные дистрибутивы.

1. На странице с релизами коннектора выберите последний релиз, скачайте архив для подходящей вам платформы и архитектуры. Так выглядит команда для скачивания коннектора версии `v0.2.4` под платформу Linux и архитектуру процессора `amd64`:

   ```
   mkdir /tmp/connector && cd /tmp/connector
   wget https://github.com/ydb-platform/fq-connector-go/releases/download/v0.2.4/fq-connector-go-v0.2.4-lin
   ux-amd64.tar.gz
   tar -xzf fq-connector-go-v0.2.4-linux-amd64.tar.gz
   ```

2. Если на сервере ещё не были развёрнуты узлы YDB, создайте директории для хранения исполняемых и конфигурационных файлов:

   ```
   sudo mkdir -p /opt/ydb/bin /opt/ydb/cfg
   ```

3. Разместите разархивированные исполняемый и конфигурационный файлы коннектора в только что созданные директории:

   ```
   sudo cp fq-connector-go /opt/ydb/bin
   sudo cp fq-connector-go.yaml /opt/ydb/cfg
   ```

4. В рекомендуемом режиме использования коннектор развёртывается на тех же серверах, что и динамические узлы YDB, следовательно, шифрование сетевых соединений между ними *не требуется*. Однако если вам всё же необходимо включить шифрование, подготовьте пару TLS-ключей и пропишите пути до публичного и приватного ключа в поля `connector_server.tls.cert` и `connector_server.tls.key` конфигурационного файла `fq-connector-go.yaml`:

   ```
   connector_server:
     # ...
     tls:
       cert: "/opt/ydb/certs/fq-connector-go.crt"
       key: "/opt/ydb/certs/fq-connector-go.key"
   ```

5. В случае, если внешние источники данных используют TLS, для организации шифрованных соединений с ними коннектору потребуется корневой или промежуточный сертификат удостоверяющего центра (Certificate Authority, CA), которым были подписаны сертификаты источников. На Linux-серверах обычно предустанавливается некоторое количество корневых сертификатов CA. Для ОС Ubuntu список поддерживаемых CA можно вывести следующей командой:

   ```
   awk -v cmd='openssl x509 -noout -subject' '/BEGIN/{close(cmd)};{print | cmd}' < /etc/ssl/certs/ca-certif
   icates.crt
   ```

   Если на сервере отсутствует сертификат нужного CA, скопируйте его в специальную системную директорию и обновите список сертификатов:

   ```
   sudo cp root_ca.crt /usr/local/share/ca-certificates/
   sudo update-ca-certificates
   ```

6. Вы можете запустить сервис вручную или с помощью systemd.

   Вручную

   Запустите сервис из консоли следующей командой:

   ```
   /opt/ydb/bin/fq-connector-go server -c /opt/ydb/cfg/fq-connector-go.yaml
   ```

С использованием systemd

Вместе с бинарным дистрибутивом fq-connector-go распространяется пример конфигурационного файла (юнита) для системы инициализации `systemd`. Скопируйте юнит в директорию `/etc/systemd/system`, активизируйте и запустите сервис:

```
cd /tmp/connector
sudo cp fq-connector-go.service /etc/systemd/system/
sudo systemctl enable fq-connector-go.service
sudo systemctl start fq-connector-go.service
```

В случае успеха сервис должен перейти в состояние `active (running)`. Проверьте его следующей командой:

```
sudo systemctl status fq-connector-go
● fq-connector-go.service - YDB FQ Connector Go
    Loaded: loaded (/etc/systemd/system/fq-connector-go.service; enabled; vendor preset: enabled)
    Active: active (running) since Thu 2024-02-29 17:51:42 MSK; 2s ago
```

Логи сервиса можно прочитать с помощью команды:

```
sudo journalctl -u fq-connector-go.service
```

Запуск в Docker

1. Для запуска коннектора используйте официальный Docker-образ. Он уже содержит конфигурационный файл сервиса. Запустить сервис с настройками по умолчанию можно следующей командой:

```
docker run -d \
    --name=fq-connector-go \
    -p 2130:2130 \
    ghcr.io/ydb-platform/fq-connector-go:latest
```

На порту 2130 публичного сетевого интерфейса вашего хоста запустится слушающий сокет GRPC-сервиса коннектора. В дальнейшем сервер YDB должен будет установить соединение именно с этим сетевым адресом.

2. При необходимости изменения конфигурации подготовьте конфигурационный файл по образцу и примонтируйте его к контейнеру:

```
docker run -d \
    --name=fq-connector-go \
    -p 2130:2130 \
    -v /path/to/config.yaml:/opt/ydb/cfg/fq-connector-go.yaml
    ghcr.io/ydb-platform/fq-connector-go:latest
```

3. В рекомендуемом режиме использования коннектор развёртывается на тех же серверах, что и динамические узлы YDB, следовательно, шифрование сетевых соединений между ними *не требуется*. Но если вам всё же необходимо включить шифрование между YDB и коннектором, подготовьте пару TLS-ключей и пропишите пути до публичного и приватного ключа в секции конфигурационного файла `connector_server.tls.cert` и `connector_server.tls.key` соответственно:

```
connector_server:
  # ...
  tls:
    cert: "/opt/ydb/certs/fq-connector-go.crt"
    key: "/opt/ydb/certs/fq-connector-go.key"
```

При запуске контейнера примонтируйте внутрь него директорию с парой TLS-ключей так, чтобы они оказались доступны для процесса `fq-connector-go` по путям, указанным в конфигурационном файле:

```
docker run -d \
    --name=fq-connector-go \
    -p 2130:2130 \
    -v /path/to/config.yaml:/opt/ydb/cfg/fq-connector-go.yaml
    -v /path/to/keys/:/opt/ydb/certs/
    ghcr.io/ydb-platform/fq-connector-go:latest
```

4. В случае, если внешние источники данных используют TLS, для организации шифрованных соединений с ними коннектору потребуется корневой или промежуточный сертификат удостоверяющего центра (Certificate Authority, CA), которым были подписаны сертификаты источников. Docker-образ для коннектора базируется на образе дистрибутива Alpine Linux, который уже содержит некоторое количество сертификатов от доверенных CA. Проверить наличие нужного CA в списке предустановленных можно следующей командой:

```
docker run -it --rm ghcr.io/ydb-platform/fq-connector-go sh
# далее в консоли внутри контейнера:
apk add openssl
awk -v cmd='openssl x509 -noout -subject' ' /BEGIN/{close(cmd)};{print | cmd}' < /etc/ssl/certs/ca-certi
ficates.crt
```

Если TLS-ключи для источников выпущены CA, не входящим в перечень доверенных, необходимо добавить сертификат этого CA в системные пути контейнера с коннектором. Сделать это можно, например, собрав собственный Docker-образ на основе имеющегося. Для этого подготовьте следующий `Dockerfile`:

```
FROM ghcr.io/ydb-platform/fq-connector-go:latest

USER root

RUN apk --no-cache add ca-certificates openssl
COPY root_ca.crt /usr/local/share/ca-certificates
RUN update-ca-certificates
```

Поместите `Dockerfile` и корневой сертификат CA в одной папке, зайдите в неё и соберите образ следующей командой:

```
docker build -t fq-connector-go_custom_ca .
```

Новый образ `fq-connector-go_custom_ca` можно использовать для развёртывания сервиса с помощью команд, приведённых выше.

Конфигурация

Актуальный пример конфигурационного файла сервиса `fq-connector-go` можно найти в репозитории.

Параметр	Назначение
`connector_server`	Обязательная секция. Содержит настройки основного GPRC-сервера, выполняющего доступ к данным.
`connector_server.endpoint.host`	Хостнейм или IP-адрес, на котором запускается слушающий сокет сервиса.
`connector_server.endpoint.port`	Номер порта, на котором запускается слушающий сокет сервиса.
`connector_server.tls`	Опциональная секция. Заполняется, если требуется включение TLS-соединений для основного GRPC-сервиса `fq-connector-go`. По умолчанию сервис запускается без TLS.
`connector_server.tls.key`	Полный путь до закрытого ключа шифрования.
`connector_server.tls.cert`	Полный путь до открытого ключа шифрования.
`logger`	Опциональная секция. Содержит настройки логирования.
`logger.log_level`	Уровень логгирования. Допустимые значения: `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, `FATAL`. Значение по умолчанию: `INFO`.
`logger.enable_sql_query_logging`	Для источников данных, поддерживающих SQL, включает логирование транслированных запросов. Допустимые значения: `true`, `false`. **ВАЖНО**: включение этой опции может привести к печати конфиденциальных пользовательских данных в логи. Значение по умолчанию: `false`.
`paging`	Опциональная секция. Содержит настройки алгоритма разбиения извлекаемого из источника потока данных на Arrow-блоки. На каждый запрос в коннекторе создаётся очередь из заранее подготовленных к отправке на сторону YDB блоков данных. Аллокации Arrow-блоков формируют наиболее существенный вклад в потребление оперативной памяти процессом `fq-connector-go`. Минимальный объём памяти, необходимый коннектору для работы, можно приблизительно оценить по формуле , где — количество одновременно выполняемых запросов, — параметр `paging.bytes_per_page`, а — параметр `paging.prefetch_queue_capacity`.
`paging.bytes_per_page`	Максимальное количество байт в одном блоке. Рекомендуемые значения - от 4 до 8 МиБ, максимальное значение - 48 МиБ. Значение по умолчанию: 4 МиБ.
`paging.prefetch_queue_capacity`	Количество заранее вычитываемых блоков данных, которые хранятся в адресном пространстве коннектора до обращения YDB за очередным блоком данных. В некоторых сценариях бóльшие значения данной настройки могут увеличить пропускную способность, но одновременно приведут и к большему потреблению оперативной памяти процессом. Рекомендуемые значения - не менее 2. Значение по умолчанию: 2.

YDB Enterprise Manager

YDB Enterprise Manager (далее — YDB EM) — это инструмент для централизованного управления кластерами YDB через веб-интерфейс и API.

> **Примечание**
>
> Развёртывание кластеров YDB возможно без YDB EM — с помощью Ansible, Kubernetes или вручную. YDB EM предоставляет удобный веб-интерфейс поверх уже существующих кластеров.

Назначение

YDB EM подключается к существующим кластерам YDB и предоставляет графический интерфейс и API для решения следующих задач:

- централизованный доступ к базам данных и кластерам YDB из единого интерфейса;
- управление динамическими узлами кластера — запуск, остановка, масштабирование;
- управление базами данных — создание, удаление, изменение параметров;
- мониторинг состояния кластера и узлов;
- управление ресурсами (CPU, RAM), выделенными для динамических узлов;
- адвизор — диагностика и рекомендации по устранению наиболее распространённых проблем;
- расширенный SQL-редактор для выполнения запросов к базам данных;
- AI-ассистент для работы с YDB.

> **Важно**
>
> YDB EM не развёртывает кластер YDB. Перед использованием YDB EM кластер должен быть развёрнут одним из способов развёртывания.

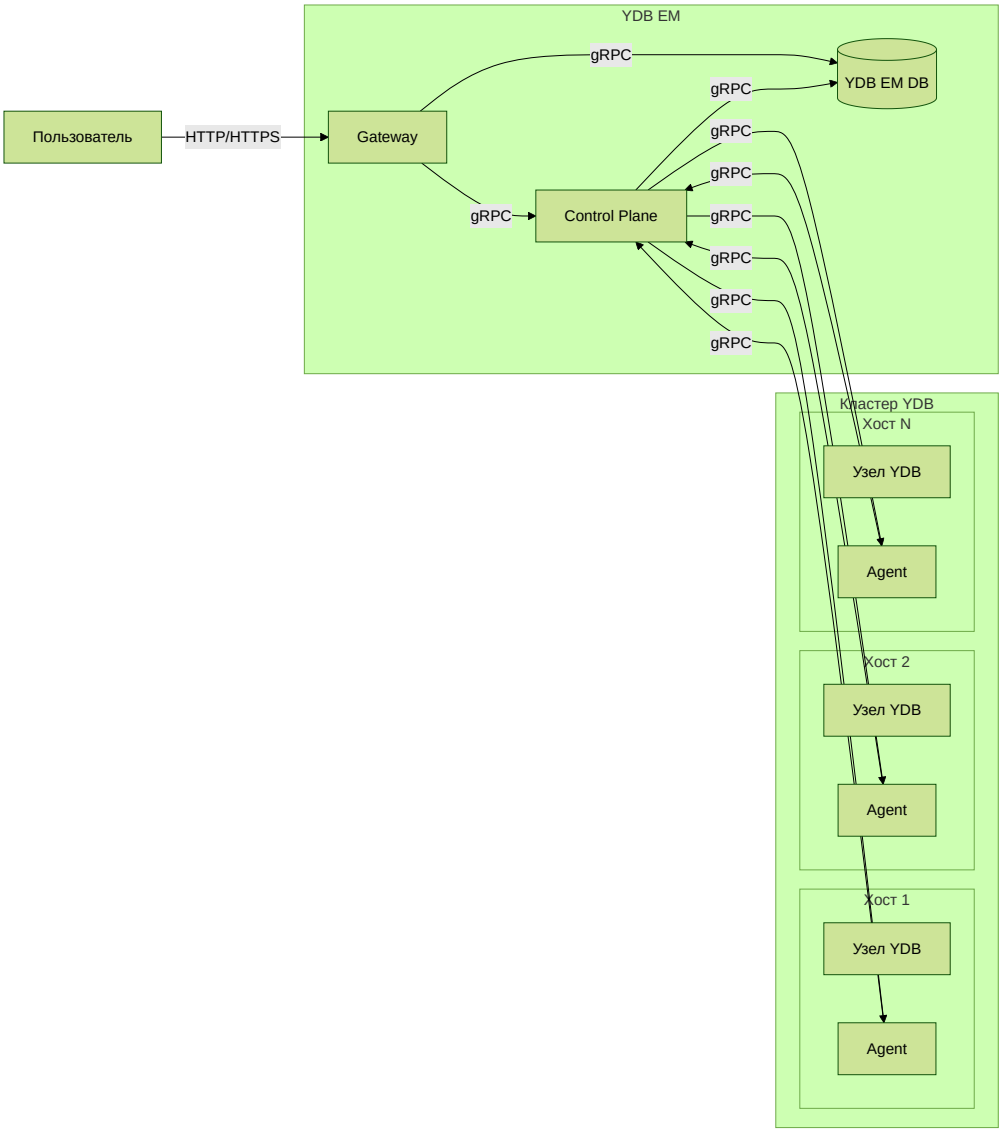
Архитектура

YDB EM состоит из трёх компонентов:

- **Gateway** — веб-интерфейс и API-бэкенд. Принимает запросы от пользователей (через браузер или API) и взаимодействует с Control Plane и базой данных YDB EM.
- **Control Plane (CP)** — координирует управление кластером. Получает команды от Gateway, хранит конфигурацию в базе данных YDB EM и отправляет задания агентам.
- **Agent** — запускается на каждом хосте кластера YDB, на котором работают динамические узлы. Агент выполняет команды Control Plane: запускает и останавливает процессы узлов YDB, следит за их состоянием и передаёт информацию о доступных ресурсах хоста.

Для хранения собственных метаданных (конфигурация кластеров, состояние узлов) YDB EM использует базу данных YDB — она может располагаться в том же кластере, которым управляет EM.

Схема взаимодействия



Пользователь взаимодействует с Gateway через браузер или API. Gateway передаёт запросы в Control Plane, который координирует работу агентов на хостах кластера. Агенты управляют процессами узлов YDB и сообщают о состоянии хостов.

Основные материалы

- Первоначальное развёртывание YDB EM
- Настройка SSO в YDB EM
- Настройка резервного копирования в S3 для YDB EM
- Настройка AI-ассистента в YDB EM

Первоначальное развёртывание YDB EM

В этом руководстве описан процесс первоначального развёртывания YDB Enterprise Manager (далее — YDB EM) с помощью Ansible. По завершении вы получите работающий экземпляр YDB EM, подключённый к существующему кластеру YDB.

Перед началом работы

Требования

Для развёртывания YDB EM необходимо:

1. **Работающий кластер YDB**. YDB EM подключается к существующему кластеру для управления его ресурсами; без кластера работа EM невозможна. В этом кластере будет автоматически создана служебная база данных для хранения метаинформации YDB EM. Подробнее о развёртывании кластера — в разделе Способы развёртывания YDB.
2. **Параметры подключения к базе данных YDB**, которая будет использоваться для хранения метаданных YDB EM:
   - Эндпоинт (например, `grpcs://ydb-node01.ru-central1.internal:2135`).
   - Имя пользователя и пароль для получения токена авторизации. Пользователь должен обладать правами на модификацию схемы, запись и чтение данных. Если кластер YDB настроен на аутентификацию через LDAP, используйте учётную запись из соответствующего каталога.
3. **Хост для компонентов Gateway и Control Plane** — один сервер (физический или виртуальный), на который будут установлены серверные компоненты YDB EM. Это может быть любой из хостов кластера YDB или отдельная машина.
4. **Ansible** на управляющей машине (с которой выполняется установка). Рекомендуется ansible-core 2.14–2.18, подробнее см. Развёртывание YDB кластера с помощью Ansible.
5. **SSH-доступ** с управляющей машины ко всем серверам, на которые будут установлены компоненты YDB EM (хост Gateway/CP и хосты кластера для агентов).

Сетевые требования

Сетевая конфигурация должна разрешать TCP-соединения по следующим портам (значения по умолчанию):

Источник	Приёмник	Порт	Протокол	Назначение
Пользователь	Gateway	8789	HTTP/HTTPS	Веб-интерфейс и API
Gateway, Agent	Control Plane	8787	gRPC	Управляющие команды
Gateway, Control Plane	YDB	2135	gRPC	Подключение к базе данных YDB EM DB

Загрузка пакета

Скачайте пакет YDB EM на управляющую машину (с которой будет выполняться установка через Ansible). Актуальный номер версии и ссылку на архив уточняйте в информации о последнем релизе:

Версия	Ссылка
`<VERSION>`	Скачать архив

Распакуйте скачанный архив в рабочую директорию (замените `<VERSION>` на актуальную версию):

```
tar -xf ydb-em-<VERSION>-stable-linux-amd64.tar.xz
```

Содержимое пакета:

Файл	Описание
`bin/ydb-em-gateway`	Бинарный файл Gateway
`bin/ydb-em-cp`	Бинарный файл Control Plane
`bin/ydb-em-agent`	Бинарный файл Agent
`collections/ydb_platform-ydb-*.tar.gz`	Ansible-коллекция для YDB
`collections/ydb_platform-ydb_em-*.tar.gz`	Ansible-коллекция для YDB EM
`examples.tar.gz`	Шаблоны конфигурации Ansible (инвентарь и плейбуки)
`install.sh`	Скрипт автоматической установки

Установка Ansible-коллекций

Установите Ansible-коллекцию для YDB, если она ещё не установлена:

```
ansible-galaxy collection install collections/ydb_platform-ydb-*.tar.gz
```

Установите Ansible-коллекцию для YDB EM:

```
ansible-galaxy collection install collections/ydb_platform-ydb_em-*.tar.gz
```

Примечание

Версии Ansible-коллекций входят в состав пакета YDB EM. Используйте файлы из директории `collections/` вашего пакета.

Подготовка конфигурации

Распаковка шаблонов конфигурации

Пакет YDB EM содержит готовые шаблоны конфигурации Ansible — файлы инвентаря, плейбуки и директории для размещения бинарных файлов и сертификатов. Распакуйте этот архив:

```
tar -xf examples.tar.gz
```

В результате появится директория `examples/`, которая будет использоваться как рабочая директория для установки.

Размещение файлов

1. Скопируйте бинарные файлы из директории `bin/` пакета в директорию `examples/files/`:

   ```
   cp bin/ydb-em-gateway bin/ydb-em-cp bin/ydb-em-agent examples/files/
   ```

2. Разместите TLS-сертификаты в директории `examples/files/certs/`. Используйте те же сертификаты, что и для узлов кластера YDB (сертификат CA, сертификаты и ключи узлов):

   ```
   cp /path/to/your/certs/* examples/files/certs/
   ```

Настройка инвентаря Ansible

Ansible использует файлы инвентаря для описания целевых серверов и их параметров. Конфигурация YDB EM разделена на несколько файлов:

- `examples/inventory/50-inventory.yaml` — общие параметры: SSH-подключение, подключение к YDB, настройки Control Plane.
- `examples/inventory/90-inventory.yaml` — описание серверов (какие хосты используются для каких компонентов).
- `examples/inventory/99-inventory-vault.yaml` — конфиденциальные данные (пароли).

Настройка списка хостов

Откройте файл `examples/inventory/90-inventory.yaml` и настройте две группы хостов.

Группа `ydb_em` — хосты для серверных компонентов YDB EM (Gateway и Control Plane). Достаточно одного хоста — это может быть любой из хостов кластера YDB или отдельный сервер:

```
ydb_em:
  hosts:
    ydb-node01.ru-central1.internal:
```

Группа `ydbd_dynamic` — хосты кластера YDB, на которые будет установлен Agent. Agent нужен на каждом хосте, где работают динамические узлы YDB, — именно через агентов YDB EM управляет процессами узлов. Как правило, сюда включаются все хосты кластера:

```
ydb:
  children:
    ydbd_dynamic:
      hosts:
        ydb-node01.ru-central1.internal:
            ydb_em_agent_cpu: 4
            ydb_em_agent_memory: 8
            location: db-dc-1
        ydb-node02.ru-central1.internal:
            ydb_em_agent_cpu: 4
            ydb_em_agent_memory: 16
            location: db-dc-2
        ydb-node03.ru-central1.internal:
            location: db-dc-3
    ydb_em:
      hosts:
        ydb-node01.ru-central1.internal:
```

Для каждого хоста в группе `ydbd_dynamic` можно указать дополнительные параметры. Если параметры не указаны, будут использованы значения по умолчанию. Уточнить настройки ресурсов можно позже через веб-интерфейс YDB EM.

Параметр	Описание
`ydb_em_agent_cpu`	Количество CPU, доступных для узлов на хосте
`ydb_em_agent_memory`	Объём RAM (в гигабайтах), доступный для узлов на хосте
`ydb_em_agent_name`	Имя хоста, используемое агентом
`location`	Расположение хоста (зона доступности)

> **Важно**
>
> Бинарные файлы YDB должны быть установлены на всех хостах группы `ydbd_dynamic` до начала развёртывания YDB EM. Подробнее — в разделе Развёртывание YDB кластера с помощью Ansible.

Настройка SSH-подключения

В файле `examples/inventory/50-inventory.yaml` проверьте параметры SSH-подключения с управляющей машины к целевым серверам:

```
# Удалённый пользователь с правами sudo
ansible_user: ansible

# Настройки для подключения через Bastion/Jump host (JUMP_IP)
# ansible_ssh_common_args: "-o ProxyJump=ansible@{{ lookup('env','JUMP_IP') }} -o UserKnownHostsFile=/dev/nul
l -o StrictHostKeyChecking=no"

# Общие настройки SSH
ansible_ssh_common_args: "-o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no"

# Приватный ключ для подключения
ansible_ssh_private_key_file: "~/.ssh/id_ed25519"
```

Важно

Параметры `UserKnownHostsFile=/dev/null` и `StrictHostKeyChecking=no` отключают проверку ключей хостов SSH, что делает соединение уязвимым для атак типа «человек посередине» (MITM). Такая конфигурация допустима только для тестовых сред. В production-окружении рекомендуется использовать проверку ключей хостов, указав путь к реальному файлу `known_hosts`:

```
ansible_ssh_common_args: "-o StrictHostKeyChecking=yes"
```

Примечание

Если для доступа к серверам используется Bastion/Jump host, раскомментируйте и настройте соответствующую строку `ansible_ssh_common_args`.

Настройка подключения к базе данных YDB

В файле `examples/inventory/50-inventory.yaml` укажите параметры подключения к базе данных YDB, которая будет использоваться для хранения метаданных YDB EM:

```
# Эндпоинт подключения к YDB
ydb_em_db_connection_endpoint: "grpcs://ydb-node01.ru-central1.internal:2135"
# Корневой домен
ydb_em_db_connection_db_root: "/Root"
# Имя базы данных
ydb_em_db_connection_db: "db"
# Пользователь YDB
ydb_user: root
```

Пароль пользователя YDB задаётся в файле `examples/inventory/99-inventory-vault.yaml`:

```
all:
  children:
    ydb:
      vars:
        ydb_password: <пароль>
```

Совет

Для защиты конфиденциальных данных рекомендуется шифровать файл `99-inventory-vault.yaml` с помощью Ansible Vault:

```
ansible-vault encrypt examples/inventory/99-inventory-vault.yaml
```

Настройка подключения к Control Plane

В файле `examples/inventory/50-inventory.yaml` укажите эндпоинт подключения к YDB EM Control Plane:

```
# Эндпоинт должен указывать на хост из группы ydb_em
ydb_em_cp_connection_endpoint: "grpcs://ydb-node01.ru-central1.internal:8787"
```

Этот параметр используется в конфигурации **Gateway** и **Agent** для подключения к **Control Plane**.

Примечание

При необходимости обеспечения высокой доступности вместо прямого адреса хоста можно указать адрес балансировщика нагрузки (например, L3/L4 балансировщик или DNS-балансировка).

Запуск установки

После завершения настройки конфигурации перейдите в директорию `examples` и выполните установку одним из способов:

Автоматическая установка

Запустите скрипт автоматической установки:

```
cd examples
./install.sh
```

Скрипт автоматически выполнит все необходимые шаги по развёртыванию компонентов YDB EM.

Ручной запуск Ansible

Также можно выполнить установку вручную, запустив плейбук Ansible:

```
ansible-playbook ydb_platform.ydb_em.initial_setup
```

Примечание

Если файл `99-inventory-vault.yaml` зашифрован с помощью Ansible Vault, добавьте флаг `--ask-vault-pass`:

```
ansible-playbook ydb_platform.ydb_em.initial_setup --ask-vault-pass
```

Проверка установки

После успешного завершения установки откройте веб-интерфейс YDB EM в браузере:

```
https://<FQDN хоста из группы ydb_em>:8789/ui/clusters
```

Например:

```
https://ydb-node01.ru-central1.internal:8789/ui/clusters
```

В веб-интерфейсе должна отобразиться страница со списком кластеров YDB.

Устранение неполадок

Не удаётся подключиться к веб-интерфейсу

1. Убедитесь, что порт `8789` доступен с вашей машины:

   ```
   curl -k https://<FQDN>:8789/ui/clusters
   ```

2. Проверьте статус сервиса YDB EM Gateway на целевом хосте:

   ```
   sudo systemctl status ydb-em-gateway
   ```

3. Проверьте логи сервиса:

   ```
   sudo journalctl -u ydb-em-gateway -n 100
   ```

Агент не подключается к Control Plane

1. Убедитесь, что порт `8787` доступен между узлами кластера и хостом Control Plane.
2. Проверьте статус сервиса YDB EM Agent на узле:

   ```
   sudo systemctl status ydb-em-agent
   ```

3. Проверьте корректность TLS-сертификатов и эндпоинта Control Plane в конфигурации агента.

Ошибки подключения к базе данных YDB

1. Убедитесь, что эндпоинт, имя базы данных, пользователь и пароль указаны корректно.
2. Проверьте сетевую доступность порта `2135` на узле YDB с хоста YDB EM.
3. Проверьте логи YDB EM Control Plane:

   ```
   sudo journalctl -u ydb-em-cp -n 100
   ```

Настройка SSO в YDB EM

YDB Enterprise Manager (далее — YDB EM) поддерживает единый вход (Single Sign-On, SSO) через внешний провайдер идентификации (Identity Provider, IdP) по протоколу OpenID Connect (OIDC). Для работы в веб-интерфейсе YDB EM пользователь может войти с корпоративной учётной записью на странице IdP. Если у пользователя уже есть активная сессия у IdP, повторный ввод учётных данных зависит от политики провайдера.

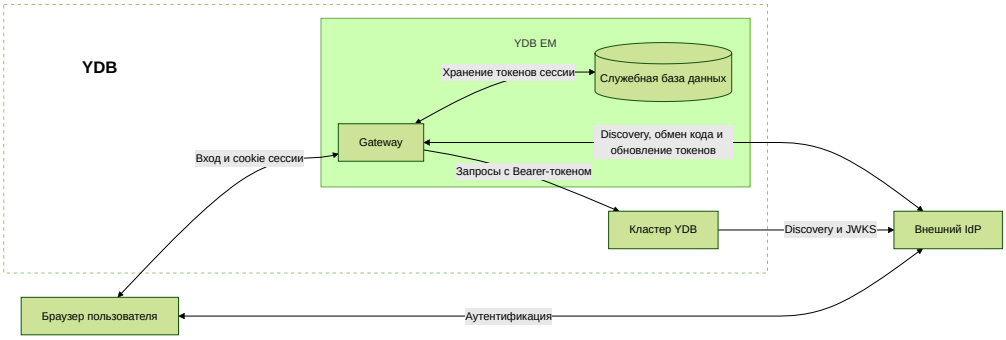
SSO в веб-интерфейсе поддерживается с помощью YDB EM. При этом сервер YDB поддерживает аутентификацию через внешний IdP по JWT-токенам, переданным Gateway, CLI или другим клиентом.

Компоненты SSO

В едином входе участвуют браузер пользователя, IdP, Gateway, служебная база данных YDB EM и кластер YDB, к которому обращается пользователь.

На схеме используются следующие термины:

- OIDC Discovery — получение конфигурации IdP из JSON-документа по известному адресу. Документ содержит адреса сервисов аутентификации, выдачи токенов и набора ключей.
- JWKS (JSON Web Key Set) — набор ключей в формате JSON. IdP публикует в нём открытые ключи, с помощью которых YDB проверяет подпись JWT-токенов.

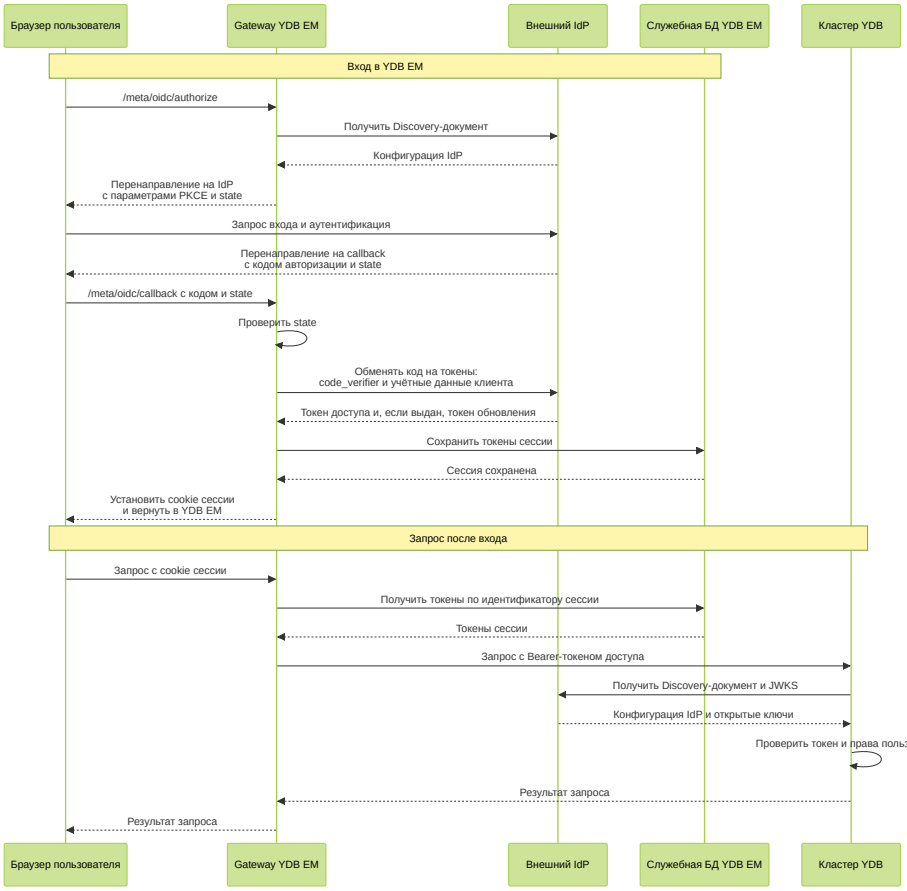


Как работает SSO

Gateway — компонент YDB EM, который обслуживает веб-интерфейс и API. Он выполняет вход по схеме Authorization Code с PKCE (Proof Key for Code Exchange), используя метод `S256`:

1. Браузер обращается к `/meta/oidc/authorize` на Gateway. Gateway получает адреса сервисов IdP из Discovery-документа `<issuer>/.well-known/openid-configuration` и перенаправляет браузер на страницу входа провайдера.
2. Пользователь проходит аутентификацию у IdP. Провайдер возвращает браузер на `/meta/oidc/callback` с кодом авторизации и параметром состояния `state`.
3. Gateway сопоставляет `state` с начатым входом и обменивает код на токены, передавая IdP учётные данные OIDC-клиента и проверочный код PKCE `code_verifier`. Это случайная секретная строка, созданная Gateway при начале входа; она подтверждает, что код авторизации обменивает тот же клиент, который начал вход.
4. Gateway сохраняет токены в служебной базе данных YDB EM и устанавливает в браузере cookie с непрозрачным идентификатором сессии. Сами токены в cookie не передаются. Cookie имеет префикс `__Host-` и атрибуты `Secure`, `HttpOnly`, `SameSite=Strict`.
5. При последующих запросах браузер передаёт cookie, а Gateway получает из сессии токен доступа (access token) и использует его как `Bearer`-токен для запросов к YDB. Сервер YDB проверяет JWT-токен и определяет пользователя и его группы для авторизации.

Диаграмма показывает успешный вход и последующий запрос к кластеру. Конфигурация IdP и ключи могут использоваться из кеша; их получение показано для случая, когда они ещё не загружены.



Префикс и атрибуты cookie ограничивают доступ к идентификатору сессии:

- `__Host-` требует HTTPS, атрибута `Secure`, пути `Path=/` и отсутствия атрибута `Domain`. Cookie привязана к конкретному хосту Gateway; поддомены не могут установить такую cookie для него.
- `Secure` разрешает передачу cookie только по HTTPS.
- `HttpOnly` запрещает JavaScript читать или изменять cookie через API браузера, снижая риск кражи идентификатора сессии скриптом.
- `SameSite=Strict` запрещает отправлять cookie в межсайтовых запросах, снижая риск подделки запросов от имени пользователя (CSRF).

Подробнее см. в разделе Cookie Security документа OAuth 2.0 for Browser-Based Applications (RFC 10017) и в описании заголовка `Set-Cookie`.

Если IdP выдал токен обновления (refresh token), Gateway автоматически обновляет истекающий токен доступа. Если сессию нельзя продолжить, пользователю необходимо войти снова.

При выходе через YDB EM Gateway удаляет серверную сессию и cookie и запрашивает отзыв токенов, если IdP предоставляет адрес сервиса отзыва токенов `revocation_endpoint`.

Перед началом работы

Для настройки необходимы:

- Развёрнутый YDB EM.
- IdP с поддержкой OIDC Discovery, Authorization Code, PKCE с методом `S256` и аутентификации клиента методом `client_secret_basic`.
- HTTPS-доступ к Gateway и IdP. Браузер должен доверять сертификату Gateway, а Gateway и узлы YDB — сертификатам IdP.
- Сетевой доступ от Gateway к Discovery-документу и сервису выдачи токенов IdP, а от узлов YDB — к Discovery-документу и JWKS. Браузер должен иметь доступ к странице входа IdP и адресу возврата на Gateway.
- Поддержка аутентификации через внешний IdP на кластерах YDB, к которым YDB EM обращается с пользовательскими токенами, включая кластер, через который выполняется вход в YDB EM.

Настройка IdP

Области доступа (scopes) определяют, какие сведения о пользователе и возможности клиент запрашивает у IdP. Например, `profile` и `email` используются для запроса данных профиля и адреса электронной почты. Необходимые области доступа зависят от настроек IdP.

Зарегистрируйте в IdP конфиденциальный OIDC-клиент для YDB EM:

1. Включите Authorization Code и PKCE с методом `S256`.
2. Разрешите аутентификацию клиента на сервисе выдачи токенов методом `client_secret_basic`: Gateway передаёт `client_id` и `client_secret` в заголовке HTTP Basic.
3. Укажите разрешённый адрес возврата (redirect URI). Например, для Gateway по адресу `https://em.example.com:8789`:

   ```
   https://em.example.com:8789/meta/oidc/callback
   ```

4. Сохраните `client_id` и `client_secret` для конфигурации Gateway.

5. Настройте выдачу JWT-токенов доступа с нужными `iss` (issuer, издатель токена), `aud` (audience, получатели токена), идентификатором пользователя и группами. Значение `iss` должно совпадать с `issuer` в конфигурации проверки токенов YDB. Поле `aud` задаёт, для каких получателей выпущен токен: если на кластере задан параметр `audience`, его значение должно присутствовать в `aud`. Пример такой проверки приведён ниже. Именно токен доступа передаётся в YDB; настройка этих полей только в токене идентификации (ID token) недостаточна. Требования к подписи, ключам и полям JWT-токена описаны в разделе аутентификации через внешний IdP.
6. Если требуется автоматическое продление сессии, разрешите выдачу токена обновления. Необходимые разрешения и области доступа зависят от IdP.

Используйте один и тот же `issuer` в настройках Gateway и YDB, например `https://idp.example.com/realms/company`. Укажите HTTPS-адрес без завершающего `/`, совпадающий с `issuer` в Discovery-документе и `iss` в JWT-токене.

Gateway получает `authorization_endpoint` и `token_endpoint` из Discovery-документа. Их URL должны начинаться со значения `issuer`. Необязательный `revocation_endpoint` используется только при соблюдении того же условия.

Настройка Gateway

Добавьте секцию `security.oidc` в YAML-конфигурацию Gateway. Если секция `security` уже существует, дополните её:

```
security:
  oidc:
    issuer: "https://idp.example.com/realms/company"
    client_id: "ydb-em"
    client_secret: "<client-secret>"
    scopes:
      - openid
      - profile
      - email
    redirect_to_idp_on_unauthorized: true
```

Замените `<client-secret>` секретом, выданным IdP при регистрации OIDC-клиента YDB EM. Секрет клиента (`client_secret`) — конфиденциальная строка, которой Gateway вместе с `client_id` подтверждает свою подлинность перед IdP при обмене кода на токены и их обновлении. Это учётные данные приложения YDB EM, а не пароль пользователя. Ограничьте доступ к конфигурационному файлу, поскольку он содержит секрет.

Параметр	Описание
`security.oidc.issuer`	HTTPS-адрес издателя токенов, используемый для OIDC Discovery. Обязательный параметр.
`security.oidc.client_id`	Идентификатор клиента YDB EM в IdP. Обязательный параметр.
`security.oidc.client_secret`	Секрет клиента YDB EM в IdP. Обязательный параметр.
`security.oidc.scopes`	Список запрашиваемых областей доступа. По умолчанию пустой; `openid` Gateway добавляет автоматически. Дополнительные области доступа, например `profile`, `email` или необходимые для получения групп и токена обновления, согласуйте с настройками IdP.
`security.oidc.redirect_to_idp_on_unauthorized`	По умолчанию `true`: при получении ответа `401 Unauthorized`, UI выполняет редирект пользователя на страницу аутентификации IdP. При `false` редирект не происходит, но `/meta/oidc/authorize` остаётся доступен для явного начала входа.

Если секция `security.oidc` присутствует, все три поля `issuer`, `client_id` и `client_secret` должны быть непустыми, иначе Gateway не загрузит конфигурацию. Чтобы отключить OIDC, удалите секцию целиком.

Изменение `issuer` или `client_id` в конфигурации Gateway требует повторного входа пользователей, поскольку от этих параметров зависит имя cookie сессии.

HTTPS и балансировщик

Gateway формирует redirect URI из заголовка `Host`, признака TLS входящего соединения и пути `/meta/oidc/callback`. Поэтому адрес должен совпадать с зарегистрированным в IdP, включая схему, имя хоста и порт.

Если перед Gateway расположен обратный прокси или балансировщик, сохраните внешний `Host` и используйте HTTPS-соединение до Gateway. Одного заголовка `X-Forwarded-Proto` недостаточно: при формировании redirect URI Gateway определяет схему по собственному входящему соединению. Доступ браузера по HTTPS также необходим для cookie с атрибутом `Secure`.

Если работают несколько экземпляров Gateway, обеспечьте попадание запросов `/meta/oidc/authorize` и `/meta/oidc/callback` одного входа на один экземпляр. Незавершённый обмен кодом хранится в памяти этого экземпляра; общей служебной базы данных недостаточно для переноса начатого входа между экземплярами.

В конфигурации Gateway необходимо включить маршрутизатор и добавить маршрут для OIDC-запросов:

```
router:
  enabled: true
  routes_to_handle:
    - path: /meta/oidc
```

Если секция `router` уже существует, дополните список `routes_to_handle`, сохранив остальные маршруты.

Настройка YDB

В конфигурации кластеров, принимающих пользовательский токен доступа от YDB EM, настройте проверку JWT-токенов. Например, для кластера `prod`, который доверяет IdP из примера настройки Gateway, добавьте следующие параметры в секцию `auth_config`:

```
auth_config:
  external_idp_config:
    issuer: "https://idp.example.com/realms/company"
    audience: "prod"
    subject_claim_name: "preferred_username"
    groups_claim_name: "groups"
  external_idp_authentication_domain: "sso"
  use_access_service: false
```

Настройте IdP так, чтобы токен доступа, выдаваемый клиенту `ydb-em`, содержал соответствующие поля. Пример фрагмента полезной нагрузки токена:

```
{
  "iss": "https://idp.example.com/realms/company",
  "aud": ["prod"],
  "preferred_username": "alice",
  "groups": ["developers"]
}
```

В этом примере `issuer` одинаков в настройках Gateway и кластера и совпадает с `iss` в токене. Значение `audience: "prod"` входит в список `aud`. Параметры `subject_claim_name` и `groups_claim_name` задают поля, из которых YDB получает имя пользователя и его группы. С учётом домена `sso` будут сформированы SID `alice@sso` и `developers@sso`.

Описание всех параметров и ограничений совместимости приведено в разделе «Конфигурация аутентификации с использованием внешнего IdP».

Предоставьте пользователям или группам права на нужные объекты. Успешный вход через IdP сам по себе не выдаёт права в YDB.

Настройки подключения самого Gateway к служебной базе данных сохраняются отдельно: пользовательская OIDC-сессия не заменяет сервисные учётные данные YDB EM.

Пример проверки получателя токена

Предположим, YDB EM подключён к трём кластерам, которые доверяют одному IdP с одинаковым `issuer`. На первом кластере параметр `auth_config.external_idp_config.audience` не задан, на втором он равен `prod`, а на третьем — `preprod`. IdP включает в поле `aud` токена список получателей, для которых разрешено использовать этот токен.

В таблице показано, на каких кластерах токен будет принят при условии, что его подпись, издатель, срок действия и остальные проверяемые поля корректны:

Поле `aud` в токене	Первый кластер: `audience` не задан	Второй кластер: `audience: prod`	Третий кластер: `audience: preprod`
Поле отсутствует	Да	Нет	Нет
`["other"]`	Да	Нет	Нет
`["prod"]`	Да	Да	Нет
`["preprod"]`	Да	Нет	Да
`["prod", "preprod"]`	Да	Да	Да

Первый кластер принимает корректные токены всех пользователей этого IdP независимо от `aud`. Второй и третий принимают токен только при наличии соответственно `prod` или `preprod` в списке `aud`. Поэтому рекомендуется задавать `audience`, чтобы кластер не принимал токены, выпущенные для других получателей.

Успешная проверка токена позволяет аутентифицировать пользователя, но доступ к данным по-прежнему определяется его правами и правами его групп в YDB.

Применение настройки и проверка входа

1. Примените конфигурацию YDB штатным способом для вашего развёртывания.
2. Разверните обновлённую конфигурацию Gateway и перезапустите его.
3. Проверьте регистрацию OIDC-обработчиков:

   ```
   curl --fail https://em.example.com:8789/capabilities
   ```

   В объекте `Capabilities` должны присутствовать `/meta/oidc/authorize` и `/meta/oidc/callback` со значением `1`.
4. Откройте в браузере адрес начала входа:

   ```
   https://em.example.com:8789/meta/oidc/authorize?return_to=%2Fui%2Fclusters
   ```

   Параметр `return_to` задаёт локальный путь для возврата после входа. Если параметр отсутствует или содержит недопустимый путь, используется `/`.
5. Войдите на странице IdP и убедитесь, что браузер вернулся в YDB EM на `/ui/clusters`.
6. Откройте доступную пользователю базу данных и выполните запрос на чтение данных из таблицы, на которую ему выданы права чтения. Это проверяет не только вход в YDB EM, но и передачу токена и авторизацию в YDB.

Настройка резервного копирования в S3 для YDB EM

В этом руководстве описана настройка резервного копирования баз данных в S3-совместимое хранилище через YDB Enterprise Manager (YDB EM). Резервное копирование настраивается в конфигурационном файле Control Plane. В примерах ниже путь к этому файлу обозначен как `<ydb-em-cp-config.yaml>`.

Для настройки S3-бэкапов нужны три блока конфигурации:

- `backup_targets` — куда сохранять резервные копии;
- `secret_key` — каким мастер-ключом зашифрованы ключи доступа к S3;
- `locations[].default_backup_config` — когда запускать резервное копирование и сколько хранить копии.

> **Примечание**
>
> Эта статья описывает настройку автоматического резервного копирования через YDB EM. Общую информацию о способах резервного копирования в YDB см. в разделах Концепции резервного копирования и Резервное копирование и восстановление.

Настройка S3-хранилища

Добавьте в конфигурацию Control Plane блок `backup_targets`:

```
backup_targets:
  - target_id: "target-em"
    tags:
      locations:
        - "em"
    settings:
      s3:
        endpoint: s3.example.net
        bucket: ydb-em-backups
        access_key: "<encrypted access key>"
        secret_key: "<encrypted secret key>"
        compression: "zstd"
```

Параметр	Описание
`target_id`	Уникальный идентификатор целевого хранилища резервных копий.
`tags.locations`	Список `location_id`, для которых применяется этот target. База данных будет копироваться в этот target, если ее `location_id` есть в списке. Для YDB EM обычно используется значение `em`, соответствующее `locations[].database_location_id` и `meta_location_id`.
`settings.s3.endpoint`	Адрес S3-совместимого хранилища.
`settings.s3.bucket`	Имя бакета для резервных копий.
`settings.s3.scheme`	Протокол подключения: `1` — HTTP, `2` — HTTPS. Если параметр не задан, используется HTTPS.
`settings.s3.access_key`	Зашифрованный ключ доступа к S3.
`settings.s3.secret_key`	Зашифрованный секретный ключ доступа к S3.
`settings.s3.compression`	Алгоритм сжатия экспортируемых данных. По умолчанию используется `zstd`. Удалите параметр, если сжатие не требуется.

Location — это логическая группа баз данных в YDB EM, к которой каждая база привязана через идентификатор `location_id`. Как правило, одна location соответствует одной зоне размещения баз данных — например, дата-центру или зоне доступности, в которой они работают, — но location можно использовать и для логического разделения, например по окружениям (`prod`, `test`). Например, если базы данных развёрнуты в двух дата-центрах, им можно назначить разные location и направлять резервные копии из каждого дата-центра в своё S3-хранилище. Для типовой установки YDB EM используется единственная location со значением `em`, которому соответствуют поля `locations[].database_location_id` и `meta_location_id` в конфигурации Control Plane.

`tags.locations` нужен, когда один Control Plane управляет базами из нескольких location и для разных location нужно использовать разные хранилища резервных копий. Worker выбирает target по `location_id` базы: если `location_id` базы входит в `tags.locations`, для резервного копирования используется этот target. Так можно, например, направлять бэкапы из разных зон или окружений в разные S3-бакеты.

> **Примечание**
>
> Вместо привязки target к `location_id` можно использовать `explicit_backup_targets` с правилами `by_database_path` или `by_cloud_id`, если конкретную базу данных или облако нужно закрепить за определенным target.

Настройка мастер-ключа

На верхнем уровне конфигурационного файла укажите путь к файлу с мастер-ключом:

```
secret_key: configs/em/secret_key
```

Этот мастер-ключ используется для шифрования и расшифровки значений `settings.s3.access_key` и `settings.s3.secret_key`. Файл мастер-ключа должен быть доступен процессам Control Plane по пути, указанному в параметре `secret_key`.

Важно

Не сохраняйте `access_key` и `secret_key` в конфигурации в открытом виде. YDB EM ожидает зашифрованные значения и расшифровывает их во время выполнения.

Шифрование ключей доступа к S3

Зашифруйте ключи доступа к S3 тем же мастер-ключом, который указан в параметре `secret_key`. Для шифрования используйте бинарный файл Control Plane `ydb-em-cp`, который входит в пакет YDB EM как `bin/ydb-em-cp` и размещается на хосте Control Plane при первоначальном развёртывании:

```
ydb-em-cp admin crypto encrypt --body '<plaintext access key>' --cfg-file <ydb-em-cp-config.yaml>
ydb-em-cp admin crypto encrypt --body '<plaintext secret key>' --cfg-file <ydb-em-cp-config.yaml>
```

Скопируйте полученные зашифрованные строки в параметры `settings.s3.access_key` и `settings.s3.secret_key`.

Чтобы проверить, что значение можно расшифровать тем же мастер-ключом, выполните:

```
ydb-em-cp admin crypto decrypt --body '<encrypted value>' --cfg-file <ydb-em-cp-config.yaml>
```

Настройка расписания и срока хранения

Target определяет, куда сохранять резервные копии. Расписание и срок хранения задаются отдельно, в блоке `locations[].default_backup_config`. Это конфигурация резервного копирования по умолчанию для баз данных в указанной location. Если для конкретной базы данных в YDB EM задана индивидуальная конфигурация резервного копирования, она может отличаться от этих значений по умолчанию.

```
locations:
  - database_location_id: em
    default_backup_config:
      backup_settings:
        - name: daily
          type: SYSTEM
          backup_schedule:
            daily_backup_schedule:
              execute_time:
                hours: 20
          backup_time_to_live: "604800s"
```

Параметр	Описание
`backup_settings[].name`	Имя настройки резервного копирования.
`backup_settings[].type`	Тип настройки. Для системного расписания используйте `SYSTEM`.
`backup_schedule.daily_backup_schedule.execute_time.hours`	Час запуска ежедневного резервного копирования в UTC.
`backup_time_to_live`	Срок хранения резервной копии в секундах. Например, `604800s` — 7 суток.

Применение конфигурации

Чтобы применить настройки:

1. Разместите обновленный конфигурационный файл в рабочей директории Control Plane. Например, для установки в `/opt/ydb-em` это может быть файл `/opt/ydb-em/ydb-em-cp/cfg/config.yaml`.
2. Убедитесь, что файл мастер-ключа, указанный в `secret_key`, доступен процессам Control Plane.
3. Проверьте, что `settings.s3.access_key` и `settings.s3.secret_key` зашифрованы тем же мастер-ключом.
4. Перезапустите сервис Control Plane:

   ```
   sudo systemctl restart ydb-em-cp
   ```

После перезапуска Control Plane по расписанию определяет target по `location_id`, настраивает S3-хранилище и запускает экспорт данных в указанный бакет.

Проверочный список

Перед запуском проверьте:

- `backup_targets[].tags.locations` содержит `location_id` баз данных, для которых нужно включить резервное копирование;
- `endpoint`, `bucket` и `scheme` указывают на нужное S3-хранилище;
- `access_key` и `secret_key` зашифрованы командой `ydb-em-cp admin crypto encrypt`;
- верхнеуровневый параметр `secret_key` указывает на корректный файл мастер-ключа;
- в `locations[].default_backup_config` задано расписание и срок хранения;
- обновленный конфигурационный файл размещен в рабочей директории Control Plane;
- сервис Control Plane перезапущен после изменения конфигурации.

Настройка AI-ассистента в YDB EM

В этой инструкции описано, как включить AI-ассистента в YDB Enterprise Manager (YDB EM). После настройки пользователи увидят ассистента в веб-интерфейсе YDB EM. Ассистент отправляет запросы к модели через Gateway и может использовать инструменты Model Context Protocol (MCP), предоставляемые Gateway.

Перед началом

Эту инструкцию можно использовать до первого развёртывания YDB EM или для уже работающей установки. При первом развёртывании добавьте переменные в inventory до запуска initial setup playbook. Инструкцию по развёртыванию см. в разделе Первоначальное развёртывание YDB EM.

Убедитесь, что у вас есть:

1. Доступ к Ansible inventory, который используется для развёртывания YDB EM.
2. Endpoint OpenAI-compatible модели, который будет доступен с хоста Gateway.
3. Права на изменение token-файла Gateway. В рамках настройки будет необходимо прописать секрет для доступа к модели в token-файл. Gateway читает этот файл и использует значение поля `Token` записи, у которой поле `Name` равно `ydb_em_ai_model_token_name`, например `model-token`, как upstream-заголовок `Authorization`.
4. Если вы обновляете существующую установку, доступ к развёрнутому хосту Gateway.

Для уже работающей установки сначала убедитесь, что хост Gateway управляется тем же Ansible inventory. Перед запуском playbook проверьте активный service manager unit, владельца процесса, путь к конфигурации, token-файл и порт, который слушает Gateway. Если на хосте используется custom или ручной layout Gateway, применяйте те же настройки через операционную процедуру этой установки, а не запускайте playbook вслепую.

> **Важно**
>
> Не помещайте API-ключи модели, OAuth-токены и другие секреты в `ydb_em_ai_assistant_client_runtime_config`. Gateway возвращает это значение в браузер через `GET /meta/ai_assistant_client_config`. Секреты держите в token-файле.

Как это работает

Браузер работает с ассистентом через Gateway:

1. UI проверяет `GET /capabilities`. Кнопка ассистента отображается, если `Settings.Proxy.Model` равно `true` и включена пользовательская настройка AI-ассистента. Когда backend capability появляется впервые, YDB EM инициализирует эту пользовательскую настройку включённой.
2. UI получает runtime-настройки из `GET /meta/ai_assistant_client_config`.
3. UI отправляет запросы к модели в `/proxy/model/...`.
4. Gateway перенаправляет запросы к модели в `ydb_em_ai_model_endpoint` и добавляет заголовок Authorization из tokenator.
5. UI отправляет MCP-запросы в `/meta/mcp`. Gateway публикует MCP-инструменты через этот endpoint и выполняет их через API YDB EM и proxy endpoints кластеров. Если поиск по документации настроен, `search_docs` публикуется через тот же MCP endpoint и доступен ассистенту.

Настройте доступ к модели

`ydb_em_ai_model_token_name` — это имя записи tokenator, которую Gateway использует для запросов к модели.

Пример записи tokenator:

```
StaticTokenInfo {
  Name: "model-token"
  Token: "Bearer <model-api-token>"
}
```

Способ доставки токена зависит от вашего процесса развёртывания. Главное требование: tokenator-файл Gateway содержит запись, у которой `Name` совпадает с `ydb_em_ai_model_token_name`.

> **Примечание**
>
> Стандартный Ansible-шаблон токенов YDB EM генерирует только запись `meta` для доступа к metabase. Если в вашей установке используется этот шаблон без изменений, расширьте или переопределите tokenator-файл через ваш процесс развёртывания, чтобы в нём также были `model-token` и, если он отличается, токен для embeddings.

Настройте Ansible-переменные

Добавьте переменные AI-ассистента в inventory YDB EM, например в `examples/inventory/50-inventory.yaml` или другой inventory-файл вашего развёртывания.

```yaml
ydb_em_ai_assistant_enabled: true
ydb_em_ai_model_endpoint: "https://llm.example.com"
ydb_em_ai_model_token_name: "model-token"

ydb_em_ai_assistant_client_runtime_config:
  llm:
    baseURL: "/proxy/model/v1"
    model: "<model-name>"
    temperature: 0.2
  mcp:
    - id: "ydb-meta"
      url: "/meta/mcp"
```

```
    transport: "streamable"
    toolCallTimeoutMs: 600000
```

Параметр	Описание
`ydb_em_ai_assistant_enabled`	Включает настройки AI-ассистента в Gateway.
`ydb_em_ai_model_endpoint`	Upstream model API endpoint, который использует Gateway.
`ydb_em_ai_model_token_name`	Имя записи tokenator для запросов к модели.
`ydb_em_ai_assistant_client_runtime_config.llm.baseURL`	URL модели, который видит браузер. Используйте Gateway proxy, обычно `/proxy/model/v1`.
`ydb_em_ai_assistant_client_runtime_config.llm.model`	Имя модели, которое отправляется в OpenAI-compatible API.
`ydb_em_ai_assistant_client_runtime_config.mcp`	MCP-серверы, доступные ассистенту. Для YDB EM используйте `/meta/mcp`.

Gateway добавляет путь из `/proxy/model/...` к `ydb_em_ai_model_endpoint`. При `baseURL: "/proxy/model/v1"` запрос chat completions будет перенаправлен в `/v1/chat/completions` на upstream endpoint. Настройте endpoint так, чтобы этот путь был корректен для вашего provider.

Настройте поиск по документации

Рекомендуется включить поиск по документации, чтобы ассистент получил MCP-инструмент `search_docs`. Для этого инструмента Gateway вызывает OpenAI-compatible embeddings endpoint и добавляет `/embeddings` к настроенному base URL, если suffix отсутствует. Когда поиск по документации включён, ассистент получает `search_docs` через настроенный MCP-сервер `/meta/mcp`.

```
ydb_em_docs_search_enabled: true
ydb_em_docs_search_embeddings_upstream_base_url: "https://llm.example.com/v1"
ydb_em_docs_search_embeddings_token_name: "model-token"
ydb_em_docs_search_embeddings_model: "<embeddings-model-name>"
ydb_em_docs_search_vector_size: 0
ydb_em_docs_search_limit: 6
ydb_em_docs_search_score: 0.6
```

Параметр	Описание
`ydb_em_docs_search_enabled`	Настраивает backend семантического поиска по документации и публикует `search_docs` через MCP, если все обязательные настройки поиска валидны.
`ydb_em_docs_search_embeddings_upstream_base_url`	Базовый URL OpenAI-совместимого endpoint для embeddings.
`ydb_em_docs_search_embeddings_token_name`	Имя записи tokenator для embeddings-запросов.
`ydb_em_docs_search_embeddings_model`	Имя embeddings-модели.
`ydb_em_docs_search_vector_size`	Необязательное переопределение размерности вектора эмбеддинга. Не задавайте переменную, чтобы использовать Ansible default `0`. При `0` Gateway не отправляет OpenAI-compatible поле `dimensions` и отключает проверку размера ответа. Положительное значение задавайте только если embeddings provider и модель поддерживают explicit dimensions и ожидаемый размер известен.
`ydb_em_docs_search_limit`	Максимальное количество возвращаемых документов.
`ydb_em_docs_search_score`	Минимальная оценка релевантности от `0` до `1`. Документы с меньшей оценкой не возвращаются.

Примените конфигурацию

После обновления inventory запустите playbook YDB EM из каталога с вашим inventory. Один и тот же playbook используется для первого развёртывания и для применения этого изменения к существующей установке:

```
ansible-playbook ydb_platform.ydb_em.initial_setup
```

Если vault-файл зашифрован, добавьте vault-параметр, который используется в вашем развёртывании:

```
ansible-playbook ydb_platform.ydb_em.initial_setup --ask-vault-pass
```

Роль сгенерирует конфигурацию и стандартный token-файл Gateway и убедится, что сервис Gateway запущен. Если дополнительные учётные данные модели доставляются через переопределённый token-файл или отдельный шаг развёртывания, убедитесь, что развёрнутый token-файл содержит настроенное имя записи. Если вы применяете эти настройки к уже работающему развёртыванию, перезапустите Gateway по вашей операционной процедуре, если ваш Ansible-запуск не перезапускает сервис после изменения конфигурации или token-файла.

Проверьте настройку

Откройте веб-интерфейс YDB EM:

```
https://<gateway-host>:8789/ui/clusters
```

Проверьте, что model proxy включён:

```
curl -k https://<gateway-host>:8789/capabilities
```

Ответ должен содержать:

```
{
  "Settings": {
    "Proxy": {
      "Model": true
    }
  }
}
```

Проверьте runtime-конфигурацию, которую получает UI:

```
curl -k https://<gateway-host>:8789/meta/ai_assistant_client_config
```

Ответ должен содержать блок `llm` и не должен содержать секреты модели.

Проверьте model proxy:

```
curl -k https://<gateway-host>:8789/proxy/model/v1/chat/completions \
  -H 'Content-Type: application/json' \
  -d '{"model":"<model-name>","messages":[{"role":"user","content":"ping"}],"max_tokens":1}'
```

Если в вашем развёртывании Gateway требует аутентификацию пользователя, добавьте те же authentication headers, которые используются для UI YDB EM.

Если включён поиск по документации, также проверьте, что `/meta/mcp` публикует `search_docs`, и выполните безопасный поисковый запрос через ваш MCP-клиент или tooling ассистента. Это проверяет и регистрацию MCP-инструмента, и путь embeddings/metabase.

Устранение неполадок

Кнопка ассистента не отображается

Проверьте, что `ydb_em_ai_assistant_enabled` равно `true`, а `/capabilities` содержит `Settings.Proxy.Model: true`. Также проверьте пользовательскую настройку AI-ассистента в UI YDB EM: YDB EM инициализирует её включённой, когда backend capability присутствует, но пользователь может её выключить.

Runtime-конфигурация невалидна

Проверьте `GET /meta/ai_assistant_client_config`. Для включённого ассистента ответ должен быть JSON-объектом со строковыми полями `llm.baseURL` и `llm.model`. Ответ `null` означает, что Gateway не загрузил включённую клиентскую конфигурацию AI-ассистента.

Model proxy возвращает ошибку авторизации

Проверьте, что `ydb_em_ai_model_token_name` совпадает с именем записи tokenator, а запись возвращает заголовок Authorization, который ожидает endpoint модели. Если tokenator не находит запись, Gateway отправляет upstream-запрос без ожидаемого заголовка Authorization.

Model proxy вызывает неправильный upstream path

Проверьте `ydb_em_ai_model_endpoint` вместе с `llm.baseURL`. Gateway добавляет путь из `/proxy/model/...` к настроенному endpoint. Дублирование `/v1` часто приводит к upstream `404`.

MCP-инструменты недоступны

Проверьте, что runtime-конфигурация содержит `url: "/meta/mcp"` и что пользовательские запросы могут обращаться к `/meta/mcp` через Gateway. Gateway регистрирует `/meta/mcp` только если MCP включён; по умолчанию он включён, но custom Gateway config может его отключить.

Поиск по документации недоступен

Проверьте настройки `ydb_em_docs_search_*`, путь embeddings endpoint и запись tokenator для embeddings-запросов. Также убедитесь, что documentation vectors доступны в metabase YDB EM. Если поиск по документации не настроен, инструмент `search_docs` не публикуется.

Embeddings endpoint отклоняет размер вектора

Если `search_docs` падает с `variable embedding size not supported`, установите `ydb_em_docs_search_vector_size: 0` и проверьте embeddings endpoint и модель.

Поиск по документации возвращает 504

Если `search_docs` возвращает `504 Gateway Timeout`, а в логах Gateway для `/meta/docs` есть `Member not found: doc_id`, проверьте таблицу metabase `ydb/Docs.db`. Текущий запрос Gateway docs search использует колонки `doc_id`, `embedding` и `payload`. Для полезных результатов в таблице также должны быть строки, а сама таблица должна быть построена для embeddings model, совместимой с `ydb_em_docs_search_embeddings_model`.

Обзор наблюдаемости

Данный раздел содержит описания по работе с инструментами наблюдения за кластером YDB.

Основные подразделы:

- Настройка мониторинга кластера YDB
- Логирование в YDB
- Минидампы (встроенный Google Breakpad)
- Трассировка в YDB
- Системные представления кластера
- Алертинг
- Диагностика изолированного кластера

Настройка мониторинга кластера YDB

Страница входит в раздел Обзор наблюдаемости и описывает настройку сбора метрик кластера YDB в Prometheus и визуализацию в Grafana. На каждом узле метрики также доступны во встроенном HTTP-интерфейсе мониторинга — раздел Доступ к метрикам через веб-интерфейс.

Совет

Перед началом работы ознакомьтесь с описанием метрик и справочником по дашбордам Grafana.

Определите, как развернут кластер YDB:

- по одному из способов развертывания кластера (Ansible, Kubernetes или вручную);
- либо в конфигурации быстрого старта (одноузловой локальный кластер).

Настройка мониторинга с помощью Prometheus и Grafana

Подготовка к установке

- Установите Prometheus.
- Установите Grafana.

Полный цикл развертывания Prometheus и Grafana в этом разделе не рассматривается — обратитесь к документации выбранных продуктов.

Варианты подготовки конфигурации сбора метрик

Конфигурацию сбора метрик с узлов YDB можно подготовить одним из способов ниже. Во всех случаях потребуется файл `prometheus_ydb.yml` — в нем описан сбор метрик YDB.

Список узлов хранения и динамических узлов базы данных указывается в файлах `ydbd-storage.yml` и `ydbd-database.yml` соответственно. Пути к этим файлам задаются в `prometheus_ydb.yml` (подробнее — в разделе Запуск Prometheus с подготовленной конфигурацией).

Ansible (с TLS)

Рекомендуемый способ получить согласованный набор файлов — сгенерировать конфигурацию сбора метрик с использованием плейбука `generate_promconf`.

Перейдите в рабочий каталог сценария Ansible для вашего кластера:

```
cd <path_to_ansible_project>
```

Запустите плейбук генерации конфигурации для Prometheus:

```
ansible-playbook ydb_platform.ydb.generate_promconf
```

Плейбук создаст каталог `promconf` со следующим содержимым:

- `prometheus_ydb.yml` — файл конфигурации Prometheus.
- `ydbd-storage.yml` — список узлов хранения кластера.
- `ydbd-database.yml` — список динамических узлов базы данных.
- `ca.crt` — сертификат, использованный при развертывании кластера.
- `grafana-dashboards` — каталог с шаблонами дашбордов для Grafana. Шаблоны загружаются из репозитория GitHub.

Проверьте содержимое сгенерированных файлов `ydbd-storage.yml` и `ydbd-database.yml`. Список узлов и портов должен совпадать с фактической топологией кластера, в том числе с узлами, отображаемыми во YDB UI кластера.

Пример проверки содержимого каталога с конфигурацией:

```
cd promconf
ls -la
```

Пример вывода:

```
-rw-rw-r-- 1 1818 ca.crt
drwxrwxr-x 2 4096 grafana-dashboards
-rw-rw-r-- 1 17532 prometheus_ydb.yml
-rw-rw-r-- 1 165 ydbd-database.yml
-rw-rw-r-- 1 164 ydbd-storage.yml
```

Проверка доступности узлов — по HTTPS; см. Метрики в формате Prometheus.

Вручную без TLS

Скопируйте файлы из каталога ydb/deploy/prometheus репозитория YDB.

Заполните секции `targets` в `ydbd-storage.yml` и `ydbd-database.yml`: укажите хосты и порты мониторинга (`--mon-port`) всех узлов хранения и динамических узлов баз данных, с которых нужно собирать метрики (как определить порт, см. Как определить порт мониторинга).

В `prometheus_ydb.yml` для задач сбора метрик задайте `scheme: http` и отключите или удалите параметры `tls_config`.

Проверка доступности узлов — по HTTP; см. Метрики в формате Prometheus.

Вручную с TLS

Скопируйте файлы из каталога ydb/deploy/prometheus репозитория YDB.

Заполните секции `targets` в `ydbd-storage.yml` и `ydbd-database.yml`: укажите хосты и порты мониторинга (`--mon-port`) всех узлов хранения и динамических узлов баз данных, с которых нужно собирать метрики (как определить порт, см. Как определить порт мониторинга).

В `prometheus_ydb.yml` для задач сбора метрик задайте `scheme: https` и настройте `tls_config`. Укажите путь к сертификату центра сертификации (CA), которым подписаны сертификаты TLS кластера:

```
scheme: https
tls_config:
    ca_file: '<ydb-ca-file>'
```

Убедитесь, что все пути в `tls_config` указывают на существующие файлы и что пользователь, под которым запущен Prometheus, имеет права на их чтение.

Если в вашей конфигурации используются клиентский сертификат и ключ, добавьте их в `tls_config`:

```
tls_config:
    ca_file: '<ydb-ca-file>'
    cert_file: '<ydb-client-cert-file>'
    key_file: '<ydb-client-key-file>'
```

Проверка доступности узлов — по HTTPS (`curl` с `https://`, см. Метрики в формате Prometheus). Укажите в `curl` флаг `--cacert` с тем же путём, что в `ca_file`.

Локальный однонодовый кластер (быстрый старт)

Если YDB запущен локально в конфигурации «один узел» и мониторинг слушает на одном порту (часто `8765`), в обоих файлах — `ydbd-storage.yml` и `ydbd-database.yml` — в секции `targets` укажите один и тот же адрес, например `localhost:8765` или `<hostname>:8765`.

Независимо от выбранного способа подготовки конфигурации шаги запуска и проверки Prometheus ниже одинаковы.

Запуск Prometheus с подготовленной конфигурацией

Отредактированные файлы положите в любую удобную директорию на машине, где запускается Prometheus (рядом с бинарником или отдельно, например `/etc/prometheus`). Файлы `prometheus_ydb.yml`, `ydbd-storage.yml` и `ydbd-database.yml` расположите в одной папке.

В `prometheus_ydb.yml` в каждой задаче секции `scrape_configs` параметр `file_sd_configs` указывает, из каких файлов брать список целей — `ydbd-storage.yml` и `ydbd-database.yml`. По умолчанию пути в `file_sd_configs` относительные: Prometheus ищет эти файлы относительно рабочей директории процесса при старте. Если задать абсолютные пути, запускать Prometheus можно из любой рабочей директории.

Если Prometheus поднимается только для YDB, в параметре `--config.file` укажите полный или относительный путь к `prometheus_ydb.yml`. Перед запуском перейдите в каталог с конфигурацией и запустите процесс из него:

```
cd <path_to_config_dir>
prometheus --config.file=prometheus_ydb.yml
```

Если Prometheus уже используется для других систем, не подменяйте основной конфиг файлом `prometheus_ydb.yml` в `--config.file`. Добавьте в существующий файл конфигурации (обычно `prometheus.yml`) задачи из секции `scrape_configs` в `prometheus_ydb.yml` — все записи с `job_name`, начинающимся с `ydb/`.

Убедитесь, что `ydbd-storage.yml` и `ydbd-database.yml` доступны по путям в `file_sd_configs` перенесенных задач (см. абзац про `file_sd_configs` выше).

Секцию `global:` из `prometheus_ydb.yml` переносите только при необходимости и сверьте значения с уже заданными в вашем конфиге. Запускайте Prometheus с вашим конфигом, например:

```
prometheus --config.file=<your_prometheus.yml>
```

После изменений проверьте конфигурацию:

```
promtool check config <your_prometheus.yml>
```

Проверьте, что Prometheus запущен и отвечает:

```
curl "http://localhost:9090/-/healthy"
```

В веб-интерфейсе Prometheus (как правило, порт `9090`) откройте **Status** → **Targets** и убедитесь, что группы опроса метрик находятся в состоянии **UP** (успешный сбор). Исключение — группа, связанная с топиками: при отсутствии топиков в базе данных для нее может отображаться ответ `204 No content`. Это не признак ошибки конфигурации.

Настройка Grafana

Подключение Prometheus как источника данных

1. Откройте веб-интерфейс Grafana.
2. Перейдите в **Connections** → **Data sources** → **Add data source**.
3. Выберите тип **Prometheus**.
4. В поле **Name** укажите произвольное имя источника данных, например `ydb`.

5. В поле **Prometheus server URL** укажите URL экземпляра Prometheus, в котором уже настроен сбор метрик с кластера YDB (например, `http://localhost:9090`, если Grafana и Prometheus на одной машине и Prometheus слушает стандартный порт).
6. При необходимости заполните поля аутентификации, TLS и таймаутов в соответствии с политикой вашей инсталляции.
7. Нажмите **Save & test** в нижней части экрана. При корректной настройке отобразится сообщение об успешном запросе к API Prometheus (например, `Successfully queried the Prometheus API`).

Дополнительно см. инструкцию Prometheus по созданию источника данных в Grafana.

Импорт дашбордов

Готовые дашборды YDB находятся в репозитории. Если вы использовали плейбук Ansible для генерации конфигурации Prometheus, шаблоны дашбордов будут размещены в подкаталоге `grafana-dashboards`. Импортируйте JSON-файлы в Grafana через веб-интерфейс или provisioning.

Состав панелей и рекомендации по использованию дашбордов приведены в справочнике по дашбордам Grafana.

Результат

Пример дашборда в Grafana

После импорта откройте дашборд **YDB Essential Metrics**. В верхней части выберите источник данных Prometheus (как в подключении) и имя базы данных.



После настройки убедитесь, что цели YDB в Prometheus в состоянии **UP** (см. Запуск Prometheus с подготовленной конфигурацией), а на дашборде **YDB Essential Metrics** отображаются метрики.

Доступ к метрикам через веб-интерфейс

Помимо сбора в Prometheus, каждый узел кластера предоставляет встроенный HTTP-интерфейс для просмотра метрик в браузере. Интерфейс показывает текущие значения метрик на момент запроса, без истории; непрерывный сбор и хранение настраивают в Prometheus (см. выше). Интерфейс слушает порт `--mon-port` (по умолчанию `8765`) на хосте узла.

Главная страница — `http://<ydb-server-address>:<ydb-port>/counters/`: на ней отображается список групп метрик (подсистем). Имена групп и метрик — в описании метрик.

где:

- `<ydb-server-address>` — адрес сервера YDB;
- `<ydb-port>` — порт мониторинга узла, параметр `--mon-port` при запуске. Значение по умолчанию: `8765`. Как определить порт на конкретном хосте, см. ниже.

При включенном TLS на узле используйте схему `https://` в URL.

Как определить порт мониторинга (`--mon-port`)

Если порт неизвестен, определите его на хосте, где запущен узел YDB (по SSH или в локальной консоли). Если в кластере несколько серверов, при необходимости выполните команду на каждом из них.

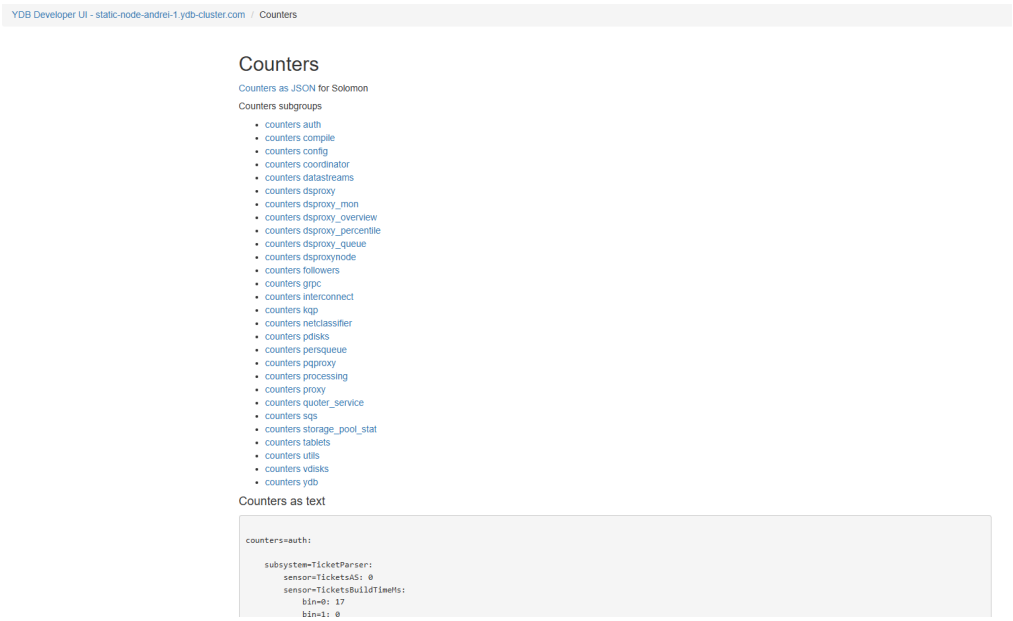
```
ps aux | grep ydbd
```

```
ubuntu@                    :~$ ps aux | grep ydbd
ydb      1128740 25.1 10.9 6398500 1796836 ?     SLsl Mar03 470:34 /opt/ydb/bin/ydbd server --yaml-config /opt/ydb/cfg/ydbd-config-static.yaml --node static --grpcs-port 2135 --grpc-ca /opt/ydb/certs/ca.crt --ic-port 19001 --ca /opt/ydb/certs/ca.crt --mon-port 8765 --mon-cert /opt/ydb/certs/web.pem
ydb      1129692 15.9  5.7 3093708 933840 ?      SLsl Mar03 287:11 /opt/ydb/bin/ydbd server --yaml-config /opt/ydb/cfg/ydbd-config-dynamic.yaml --grpcs-port 2137 --ic-port 19003 --mon-port 8767 --tenant /Root/database --node-broker-use-tls true --grpc-ca /opt/ydb/certs/ca.crt --ca /opt/ydb/certs/ca.crt --mon-cert /opt/ydb/certs/web.pem --grpc-cert /opt/ydb/certs/node.crt --grpc-key /opt/ydb/certs/node.key --node-broker grpcs://[redacted].ydb-cluster.com:2135 --node-broker grpcs://[redacted].ydb-cluster.com:2135
ydb      1129780 16.1  5.8 3095756 956340 ?      SLsl Mar03 300:53 /opt/ydb/bin/ydbd server --yaml-config /opt/ydb/cfg/ydbd-config-dynamic.yaml --grpcs-port 2138 --ic-port 19004 --mon-port 8768 --tenant /Root/database --node-broker-use-tls true --grpc-ca /opt/ydb/certs/ca.crt --ca /opt/ydb/certs/ca.crt --mon-cert /opt/ydb/certs/web.pem --grpc-cert /opt/ydb/certs/node.crt --grpc-key /opt/ydb/certs/node.key --node-broker grpcs://[redacted].ydb-cluster.com:2135 --node-broker grpcs://static-node-andrei-2.ydb-cluster.com:2135 --node-broker grpcs://[redacted].ydb-cluster.com:2135
ubuntu   1146510  0.0  0.0   6612  2320 pts/0    S+   13:35   0:00 grep --color=auto ydbd
```

В выводе может быть несколько процессов `ydbd` с разными значениями `--mon-port` (например, статический и динамический узел на одном сервере). Добавьте в мониторинг все порты тех узлов, метрики которых нужно собирать. На скриншоте выделены отдельные значения только для примера — ориентируйтесь на фактический вывод команды на ваших хостах.

Группы метрик на главной странице

На главной странице перечислены группы метрик по подсистемам — `auth`, `compile`, `grpc`, `kqp`, `pdisks`, `vdisks` и другие. Каждая группа — ссылка на страницу с метриками этой подсистемы.



Просмотр метрик группы (подсистемы)

Чтобы открыть метрики одной группы (подсистемы), перейдите по URL:

```
http://<ydb-server-address>:<ydb-port>/counters/counters=<servicename>/
```

- `<servicename>` — имя группы (подсистемы). Доступные группы отображаются на главной странице веб-интерфейса мониторинга (см. Группы метрик на главной странице).

Используйте `http` или `https` в соответствии с настройкой TLS кластера (как в Метриках в формате Prometheus).

Например, метрики утилизации ресурсов сервера — в группе `utils`:

```
http://<ydb-server-address>:<ydb-port>/counters/counters=utils
```

Метрики в формате Prometheus

Тот же узел отдает метрики в формате Prometheus — по URL с суффиксом `/prometheus`. Именно эти адреса опрашивает Prometheus (параметр `metrics_path` в `scrape_configs`).

Без TLS

```
http://<ydb-server-address>:<ydb-port>/counters/counters=<servicename>/prometheus
```

Проверить доступность эндпоинта:

```
curl "http://<ydb-server-address>:<ydb-port>/counters/counters=<servicename>/prometheus"
```

С TLS

```
https://<ydb-server-address>:<ydb-port>/counters/counters=<servicename>/prometheus
```

Проверить доступность эндпоинта:

```
curl --cacert <path-to-ca.crt> "https://<ydb-server-address>:<ydb-port>/counters/counters=<servicename>/prometheus"
```

Укажите в `--cacert` тот же путь, что в `ca_file` в `tls_config` (см. подготовку конфигурации). При самоподписанном сертификате для диагностики можно добавить флаг `-k`.

Связь с конфигурацией Prometheus

В шаблонном `prometheus_ydb.yml` хосты и порты совпадают с `ydbd-storage.yml` и `ydbd-database.yml`. Для каждой группы метрик в `scrape_configs` задан `metrics_path` вида `/counters/counters=<servicename>/prometheus` — те же подсистемы, что в списке на главной странице веб-интерфейса.

Другие системы с поддержкой формата Prometheus (Zabbix, Amazon CloudWatch и др.) подключают к тем же URL.

См. также

- Справка по метрикам
- Дашборды Grafana для YDB

- Быстрый старт с YDB
- Способы развёртывания YDB

Логирование в YDB

Каждый компонент YDB пишет сообщения разного уровня в логи (журналы). По ним можно детектировать критические проблемы или разобраться в причинах неполадок.

Настройка логирования

Настройку логирования у отдельных компонентов можно произвести в YDB UI.

На данный момент есть два варианта для запуска логирования YDB: вручную и с использованием systemd.

Вручную

Для удобства YDB предоставляет стандартные механизмы сбора логов и метрик.
Логирование осуществляется в стандартные каналы `stdout` и `stderr` и может быть перенаправлено при помощи популярных решений.

С использованием systemd

По умолчению пишутся в `journald` и достать их можно через `journalctl -u ydbd-storage`. Для доступа к логам узлов баз данных, измените имя systemd юнита соответствующим образом.

Минидампы (встроенный Google Breakpad)

Узлы `ydbd` в YDB могут собирать **минидампы** при аварийном завершении процесса с помощью встроенного Google Breakpad. Минидамп — это компактный снимок состояния процесса (стек, регистры, список загруженных модулей) в момент падения по необработанному сигналу (`SIGSEGV` , `SIGABRT` , `SIGFPE` и другим).

В отличие от системного core dump, минидамп:

- занимает существенно меньше места;
- собирается собственным обработчиком сигналов и не зависит от настроек `ulimit` и `core_pattern` на хосте;
- может автоматически передаваться внешнему скрипту для пост-обработки (загрузки, символизации, оповещения).

Примечание

Сбор минидампов работает только на Linux. На других платформах (например, macOS) `ydbd` запускается, но встроенный Breakpad в сборку не входит и минидампы не собираются.

Включение сбора минидампов

По умолчанию сбор минидампов выключен. Чтобы включить его, нужно указать директорию, в которую будут записываться дампы. Это можно сделать двумя способами: переменными окружения и опциями командной строки `ydbd server` . Опции командной строки имеют приоритет над переменными окружения.

Если директория не задана ни одним из способов, встроенный Breakpad не активируется и не устанавливает обработчики сигналов.

Примечание

Указанная директория должна существовать до запуска процесса; YDB не создаёт её автоматически. Если директория отсутствует в момент падения, минидамп не будет записан.

Переменные окружения

Переменная	Назначение
`BREAKPAD_MINIDUMPS_PATH`	директория для записи минидампов; задание этой переменной включает сбор минидампов при старте процесса
`BREAKPAD_MINIDUMPS_SCRIPT`	путь к скрипту пост-обработки минидампа

Опции командной строки

Опции добавляются к команде `ydbd server` :

Опция	Назначение
`--breakpad-minidumps-path PATH`	директория для записи минидампов
`--breakpad-minidumps-script SCRIPT`	путь к скрипту пост-обработки минидампа

Пример запуска:

```
ydbd server \
    --breakpad-minidumps-path /var/log/ydb/minidumps \
    --breakpad-minidumps-script /usr/local/bin/process-minidump.sh \
    ...
```

Скрипт пост-обработки

Если задан скрипт пост-обработки, он запускается сразу после записи минидампа. Скрипт вызывается со следующими аргументами:

Аргумент	Значение
`$1`	`true` или `false` — был ли минидамп успешно записан
`$2`	путь к файлу минидампа (пустая строка, если запись не удалась)

Скрипту не передаётся окружение процесса `ydbd` , поэтому все необходимые ему параметры должны быть зашиты в самом скрипте или вычисляться из аргументов.

Важно

Скрипт запускается из обработчика сигнала упавшего процесса через `fork` / `exec` . Обработчик сигнала блокируется до завершения скрипта, поэтому скрипт должен завершаться быстро. Тяжёлую обработку (загрузку дампа, символизацию, оповещения) следует выносить в фоновые процессы, которые скрипт запускает перед выходом.

Пример скрипта, который только перемещает дамп в архивную директорию:

```
#!/bin/sh
# $1 — succeeded (true|false), $2 — путь к минидампу
if [ "$1" = "true" ]; then
```

```
    mv "$2" /var/log/ydb/minidumps/archive/
fi
```

Анализ минидампов

Минидамп не содержит символов: для получения читаемого стека нужны символы того же сборочного артефакта `ydbd`. Для разбора используются штатные инструменты Google Breakpad (`minidump_stackwalk` и `.sym`-файлы), полученные из символов соответствующей сборки.

См. также

- Логирование в YDB
- Настройка мониторинга кластера YDB
- Google Breakpad

Системные представления кластера

Для возможности внутренней интроспекции состояния кластера пользователю предоставляется возможность осуществлять запросы в специальные служебные представления (system views). Эти представления доступны из корневой директории кластера и используют системный префикс пути `.sys`.

Пользователи облачных баз данных обычно не имеют доступа к системным представлениям кластера, так как за его поддержку и своевременную диагностику отвечает команда облака.

В описаниях доступных полей далее по тексту колонка **Ключ** содержит индекс поля первичного ключа соответствующего представления.

> **Примечание**
>
> Аналогичные системные представления существуют и для происходящего внутри конкретной базы данных, они описаны в отдельной статье для разработчиков.

Разграничение прав доступа

Возможность гибко настраивать права доступа к системным представлениям позволяет точно разграничить доступ к служебной информации и пользовательским данным.

Типовые сценарии разграничения прав

- **Доступ только к системным представлениям:**
  Если в вашей инфраструктуре есть пользователи или сервисы, которым необходимо мониторить состояние кластера или базы — например, администраторы баз данных — но не требуется доступ к пользовательским данным, рекомендуется выдать этим пользователям права на чтение только для каталога `.sys`.
- **Доступ только к данным:**
  В ситуациях, когда отдельным пользователям или группам нужно скрыть информацию о служебных и системных объектах (например, аналитикам, которым не нужен доступ к внутренней диагностике), достаточно предоставить права на чтение только соответствующего каталога с пользовательскими данными. Права на каталог `.sys` или корень базы данных при этом выдавать не нужно — это гарантирует, что пользователи не увидят и не смогут запросить системные представления.

Дополнительно, поддерживается возможность назначения различных прав на отдельные системные представления — как для конкретных пользователей, так и для групп. Такой подход позволяет гибко реализовать принцип наименьших привилегий и контролировать разный уровень доступа к служебной информации для разных ролей и задач.

Distributed Storage

Информация о работе распределённого хранилища содержится в нескольких взаимосвязанных представлениях, каждое из которых отвечает за описание своей сущности, а именно:

- PDisk
- VSlot
- Group
- Storage Pool

Кроме этого, есть отдельное представление, которое показывает статистику использования количества групп в разных пулах хранилища и возможности роста этих пулов.

ds_pdisks

Поле	Тип	Ключ	Значение
NodeId	Uint32	0	Идентификатор узла, на котором запущен PDisk
PDiskId	Uint32	1	Идентификатор PDisk (уникален внутри узла)
Type	String		Тип носителя (`ROT` , `SSD` , `NVME`)
Kind	Uint64		Числовой идентификатор, задаваемый пользователем, который нужен для объединения дисков с одинаковым типом носителя в разные подгруппы
Path	String		Путь к блочному устройству внутри машины
Guid	Uint64		Уникальный идентификатор, генерируемый случайно при добавлении диска в систему, предназначенный для предотвращения потери данных в случае перемены дисков местами
BoxId	Uint64		Идентификатор Box, в который входит данный PDisk
SharedWithOs	Bool		Наличие метки "SharedWithOs", устанавливаемой вручную при создании PDisk. Может использоваться для фильтрации дисков при создании новых групп.
ReadCentric	Bool		Наличие метки "ReadCentric", устанавливаемой вручную при создании PDisk. Может использоваться для фильтрации дисков при создании новых групп.
AvailableSize	Uint64		Число доступных для выделения байт на PDisk
TotalSize	Uint64		Общее число байт на PDisk
Status	String		Режим работы PDisk, который влияет на его участие в выделении групп (`ACTIVE` , `INACTIVE` , `BROKEN` , `FAULTY` , `TO_BE_REMOVED`)
StatusChangeTimestamp	Timestamp		Время, когда последний раз поменялся `Status` ; если `NULL` , то `Status` не менялся с момента создания PDisk

ExpectedSlotCount	Uint32		Максимальное число VSlot, которое может быть создано на этом PDisk
NumActiveSlots	Uint32		Количество занятых VSlot
DecommitStatus	String		Статус вывода из эксплуатации (декомиссии) PDisk (`DECOMMIT_NONE`, `DECOMMIT_PENDING`, `DECOMMIT_IMMINENT`, `DECOMMIT_REJECTED`)

ds_vslots

Поле	Тип	Ключ	Значение
NodeId	Uint32	0	Идентификатор узла, на котором запущен VSlot
PDiskId	Uint32	1	Идентификатор PDisk внутри узла, на котором запущен VSlot
VSlotId	Uint32	2	Идентификатор VSlot внутри PDisk
GroupId	Uint32		Номер группы хранения, в которую входит данный VSlot
GroupGeneration	Uint32		Поколение конфигурации группы хранения, в которую входит данный VSlot
FailRealm	Uint32		Относительный номер группы отказа (fail realm) VSlot внутри группы хранения
FailDomain	Uint32		Относительный номер домена отказа (fail domain) VSlot внутри группы отказа (fail realm)
VDisk	Uint32		Относительный номер VSlot внутри домена отказа (fail domain)
AllocatedSize	Uint64		Число байт, которые VSlot занимает на PDisk
AvailableSize	Uint64		Число байт, доступных для выделения данному VSlot
Status	String		Состояние запущенного VDisk в данном VSlot (`INIT_PENDING`, `REPLICATING`, `READY`, `ERROR`)
Kind	String		Предустановленная настройка режима работы VDisk (`Default`, `Log`, ...)

Стоит заметить, что кортеж `(NodeId, PDiskId)` формирует внешний ключ к представлению `ds_pdisks`, а `(GroupId)` — к представлению `ds_groups`.

ds_groups

Поле	Тип	Ключ	Значение
GroupId	Uint32	0	Номер группы хранения в кластере
Generation	Uint32		Поколение конфигурации группы хранения
ErasureSpecies	String		Режим кодирования избыточности для группы (block-4-2, mirror-3-dc, mirror-3of4, ...)
BoxId	Uint64		Идентификатор Box, в котором создана данная группа
StoragePoolId	Uint64		Идентификатор пула хранения внутри Box, в рамках которого работает данная группа
EncryptionMode	Uint32		Наличие шифрования данных в группе и алгоритм шифрования, если оно включено
LifeCyclePhase	Uint32		Наличие выработанного ключа шифрования, если шифрование включено
AllocatedSize	Uint64		Количество выделенных байт данных в группе (приводится к пользовательским байтам, то есть до избыточности)
AvailableSize	Uint64		Количество доступных для выделения байт пользовательских данных (также до избыточности)
SeenOperational	Bool		Булевой флаг, показывающий, была ли группа в рабочем состоянии после её создания
PutTabletLogLatency	Interval		90 процентиль времени выполнения запроса PutTabletLog
PutUserDataLatency	Interval		90 процентиль времени выполнения запроса PutUserData
GetFastLatency	Interval		90 процентиль времени выполнения запроса GetFast
OperatingStatus	String		Статус группы по последним отчетам VDisk (`UNKNOWN`, `FULL`, `PARTIAL`, `DEGRADED`, `DISINTEGRATED`)
ExpectedStatus	String		Статус, основанный не только на операционном отчете, но и на статусе PDisk и планах (`UNKNOWN`, `FULL`, `PARTIAL`, `DEGRADED`, `DISINTEGRATED`)

В данном представлении кортеж `(BoxId, StoragePoolId)` формирует внешний ключ к представлению `ds_storage_pools`.

ds_storage_pools

Поле	Тип	Ключ	Значение
BoxId	Uint64	0	Идентификатор Box, в который входит данный пул хранения
StoragePoolId	Uint64	1	Идентификатор пула хранения внутри Box
Name	String		Название пула хранения, задаваемое пользователем (используется при связывании таблеток и пулов хранения)

Generation	Uint64		Поколение конфигурации пула хранения (количество изменений)
ErasureSpecies	String		Режим кодирования избыточности для всех групп внутри данного пула хранения
VDiskKind	String		Предустановленная настройка режима работы всех VDisk для данного пула хранения
Kind	String		Строковое описание предназначения пула, задаваемое пользователем, также может использоваться для фильтрации
NumGroups	Uint32		Количество групп внутри данного пула хранения
EncryptionMode	Uint32		Настройка шифрования данных для всех групп (аналогично `ds_groups.EncryptionMode`)
SchemeshardId	Uint64		Идентификатор SchemeShard объекта схемы, к которому относится данный пул хранения (сейчас всегда `NULL`)
PathId	Uint64		Идентификатор узла объекта схемы внутри указанного SchemeShard, к которому относится данный пул хранения

ds_storage_stats

В отличие от других представлений, показывающих физические сущности, `ds_storage_stats` показывает агрегированную информацию о хранилище.

Поле	Тип	Ключ	Значение
BoxId	Uint64	0	Идентификатор Box, по которому считается статистика
PDiskFilter	String	1	Строковое описание фильтров, отбирающих PDisk для создания групп (например, по типу носителя)
ErasureSpecies	String	2	Режим кодирования избыточности, по которому собирается статистика
CurrentGroupsCreated	Uint32		Число созданных групп с указанными характеристиками
CurrentAllocatedSize	Uint64		Суммарное занятое место по всем группам, входящим в `CurrentGroupsCreated`
CurrentAvailableSize	Uint64		Суммарное доступное для выделения место по всем группам, входящим в `CurrentGroupsCreated`
AvailableGroupsToCreate	Uint32		Число групп с указанными характеристиками, которое можно создать с учётом необходимости резерва
AvailableSizeToCreate	Uint64		Число доступных байт, которое получится при создании всех групп из `AvailableGroupsToCreate`

Здесь стоит заметить, что `AvailableGroupsToCreate` показывают максимальное количество групп, которое можно создать, если не создавать другие виды групп. Таким образом, при расширении одного пула хранения могут поменяться числа `AvailableGroupsToCreate` в нескольких строках статистики.

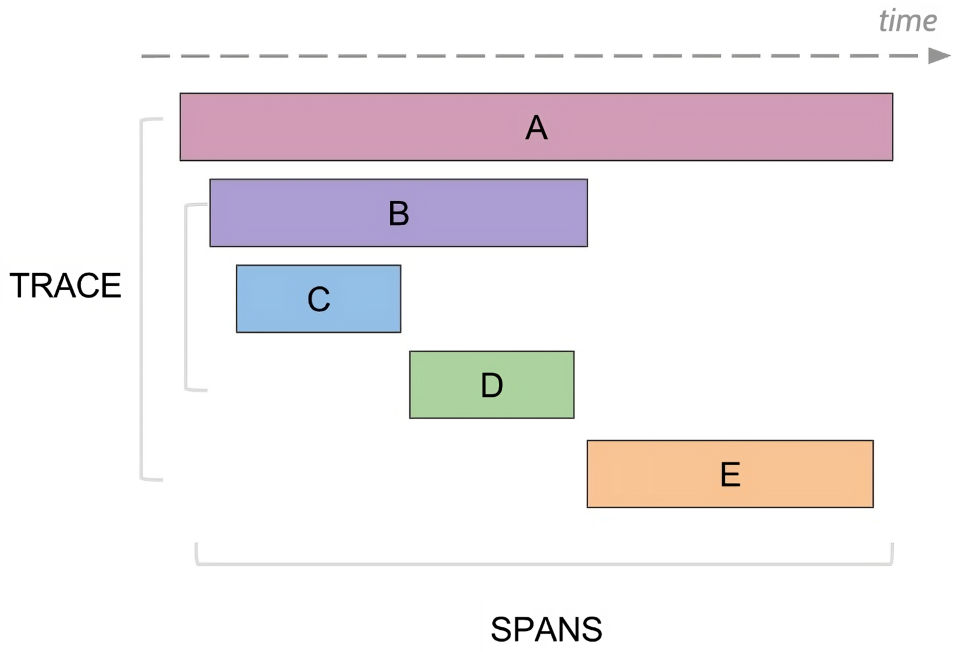
Примечание

Обращение к системным представлениям имеет скорее аналитический характер нагрузки. Частое обращение к ним в больших базах будет существенно расходовать системные ресурсы. Рекомендуемая нагрузка не более 1-2 RPS.

Трассировка в YDB

> **Примечание**
>
> Подробно трассировка описана на сайте opentelemetry в разделе observability primer.

Трассировка – инструмент, позволяющий детально посмотреть путь следования запроса по распределенной системе. Путь следования одного запроса (трасса) описывается набором спанов. Спан является отрезком времени, обычно связанным с временем выполнения определенной операции (например запись информации на диск, выполнение транзакции). Спаны образуют дерево, часто поддерево спана является его детализацией, однако это не всегда так.



Для аггрегации разрозненных спанов в трассы они отправляются в *коллектор*. Это сервис, который аггрегирует и сохраняет получаемые спаны для последующего анализа трасс. YDB не включает в себя этот сервис, администратор должен поднять его самостоятельно. Обычно в качестве коллектора используется jaeger.

Минимальная конфигурация

Для включения трассировки в YDB добавьте в конфигурацию следующую секцию:

```
tracing_config:
  backend:
    opentelemetry:
      collector_url: grpc://example.com:4317
      service_name: ydb
  external_throttling:
    - max_traces_per_minute: 10
```

В ней поле `collector_url` задает url OTLP-совместимого коллектора спанов. Более подробно с описанием секции backend можно ознакомиться в соответствующей секции.

При данной настройке никакие запросы не сэмплируются, не более 10 запросов в минуту с внешним trace-id трассируются каждым узлом кластера.

Описание секций

Backend

Пример секции

```
tracing_config:
  # ...
  backend:
    opentelemetry:
      collector_url: grpc://example.com:4317
      service_name: ydb
```

Описание

В этой секции описывается коллектор спанов. На данный момент единственная опция – `opentelemetry`. Отправка спанов с узла кластера в коллектор происходит по push-модели, от коллектора требуется совместимость с OTLP.

В секции `opentelemetry`:

- `collector_url` — URL коллектора спанов. Схема может быть либо `grpc://` для небезопасного подключения, либо `grpcs://` для TLS-подключения.
- `service_name` — имя сервиса, которым будут подписываться все спаны.

Оба параметра являются обязательными.

Uploader

Пример секции

```
tracing_config:
  # ...
  uploader:
    max_exported_spans_per_second: 30
    max_spans_in_batch: 100
    max_bytes_in_batch: 10485760 # 10 MiB
    max_export_requests_inflight: 3
    max_batch_accumulation_milliseconds: 5000
    span_export_timeout_seconds: 120
```

Описание

Uploader – компонент узла кластера, который отвечает за отправку спанов в коллектор. Во избежание перегрузки коллектора спанов, uploader не будет отправлять больше, чем `max_exported_spans_per_second` спанов в секунду в среднем.

В целях оптимизации uploader отправляет спаны батчами. Каждый батч содержит не более `max_spans_in_batch` спанов с суммарным сериализованным размеров не более `max_bytes_in_batch` байт. Каждый батч накапливается на протяжении не более, чем `max_batch_accumulation_milliseconds` миллисекунд. Батчи могут отправляться параллельно, максимальное количество одновременно отправляемых батчей контролируется параметром `max_export_requests_inflight`. Если с момента получения спана uploader'ом прошло больше `span_export_timeout_seconds` секунд, uploader может удалить его в пользу отправки более свежих спанов.

Значения по умолчанию:

- `max_exported_spans_per_second = inf` (отсутствуют какие-либо ограничения)
- `max_spans_in_batch = 150`
- `max_bytes_in_batch = 20000000`
- `max_batch_accumulation_milliseconds = 1000`
- `span_export_timeout_seconds = inf` (никакие спаны не удаляются uploader'ом)
- `max_export_requests_inflight = 1`

Секция `uploader` может полностью отсутствовать, в таком случае для каждого параметра будет использоваться значение по умолчанию.

> **Примечание**
>
> Uploader – локальный для узла компонент, как следствие, описанные ограничения применяются к каждому узлу отдельно, а не ко всему кластеру в целом.

External throttling

Пример секции

```
tracing_config:
  # ...
  external_throttling:
    - scope:
        database: /Root/db1
      max_traces_per_minute: 60
      max_traces_burst: 3
```

Описание

YDB поддерживает передачу внешних trace-id для построения цельной трассы запроса. Способ передачи внешнего trace-id описан на странице Передача внешнего trace-id в YDB. Для избежания перегрузки коллектора, в YDB есть механизм ограничения количества трассируемых извне запросов. Ограничения описываются в данной секции и являются последовательностью правил, каждое правило содержит:

- `scope` – набор селекторов для фильтрации запроса.
- `max_traces_per_minute` – наибольшее среднее количество запросов в минуту, трассируемое по данному правилу. Ожидается положительное целое число.
- `max_traces_burst` – максимальный всплеск количества трассируемых извне запросов. Ожидается неотрицательное целое число.

Обязательным параметром является только `max_traces_per_minute`.

Подробное описание этих опций приведено в секции Семантика правил

Секция `external_throttling` не является обязательной, в случае ее отсутствия все trace-id в запросах **игнорируются** (никакие внешние трассы не продолжаются).

Данную секцию можно изменять без перезагрузки узла с помощью механизма конфигурации.

Sampling

Пример секции

```
tracing_config:
  # ...
  sampling:
    - fraction: 0.01
```

```
      level: 10
      max_traces_per_minute: 5
      max_traces_burst: 2
    - scope:
        request_types:
          - KeyValue.ExecuteTransaction
          - KeyValue.Read
      fraction: 0.1
      level: 15
      max_traces_per_minute: 5
      max_traces_burst: 2
```

Описание

Для диагностирования проблем в системе, может быть полезно посмотреть на пример трассы запроса вне зависимости от того, трассируют пользователи свои запросы, или нет. Для этого в YDB есть механизм сэмплирования запросов. Для сэмплированного запроса генерируется случайный trace-id. Данная секция контролирует сэмплирование запросов в формате аналогичном `external_throttling`, каждое правило имеет два дополнительные поля:

- `fraction` – доля запросов, сэмплированных по данному правилу. Ожидается дробное число от 0 до 1.
- `level` — уровень детализации трассы. Ожидается целое число от 0 до 15 включительно. Подробнее этот параметр описан в секции Уровни детализации

Оба поля являются обязательными.

Секция `sampling` не является обязательной, в случае ее отсутствия никакие запросы не будут сэмплироваться.

Данную секцию можно изменять без перезагрузки узла с помощью механизма динамической конфигурации.

Семантика правил

Селекторы

Каждое правило включает в себя опциональное поле `scope` с набором селекторов, которые определяют, к каким запросам применяется данное правило. На данный момент поддерживаются селекторы:

- `request_types`

  Принимает список типов запросов. Запрос подходит под данный селектор если его тип содержится в списке.

Возможные значения

- KeyValue.CreateVolume
- KeyValue.DropVolume
- KeyValue.AlterVolume
- KeyValue.DescribeVolume
- KeyValue.ListLocalPartitions
- KeyValue.AcquireLock
- KeyValue.ExecuteTransaction
- KeyValue.Read
- KeyValue.ReadRange
- KeyValue.ListRange
- KeyValue.GetStorageChannelStatus
- Table.CreateSession
- Table.KeepAlive
- Table.AlterTable
- Table.CreateTable
- Table.DropTable
- Table.DescribeTable
- Table.CopyTable
- Table.CopyTables
- Table.RenameTables
- Table.ExplainDataQuery
- Table.ExecuteSchemeQuery
- Table.BeginTransaction
- Table.DescribeTableOptions
- Table.DeleteSession
- Table.CommitTransaction
- Table.RollbackTransaction
- Table.PrepareDataQuery
- Table.ExecuteDataQuery
- Table.BulkUpsert
- Table.StreamExecuteScanQuery
- Table.StreamReadTable
- Table.ReadRows
- Query.ExecuteQuery
- Query.ExecuteScript
- Query.FetchScriptResults
- Query.CreateSession
- Query.DeleteSession
- Query.AttachSession
- Query.BeginTransaction
- Query.CommitTransaction

- Query.RollbackTransaction
- Discovery.WhoAmI
- Discovery.NodeRegistration
- Discovery.ListEndpoints

> **Примечание**
>
> Трассировка поддерживается не только для типов запросов, приведенных выше. В этом списке перечислены типы запросов, которые поддерживатюся селектором `request_types`.

> **Важно**
>
> Обратите внимание, что QueryService API экспериментальное и может быть изменено в будущем.

- `database`

  Фильтрует запросы к указанной базе данных.

Запрос подходит под правило, если он подходит под все селекторы. `scope` может отсутствовать, это равносильно пустому набору селекторов, все запросы будут попадать под это правило.

Rate limiting

Параметры `max_traces_per_minute` и `max_traces_burst` используются для ограничения сверху на количество запросов. В случае sampling они ограничивают количество запросов, сэмплированных по данному правилу. В случае external throttling они ограничивают количество внешних трасс, которые попадают в систему.

Для ограничений используется вариация leaky bucket с размером бакета равным `max_traces_burst + 1`. Например, если `max_traces_per_minute = 60` и `max_traces_burst = 0`, то при потоке в 10000 запросов в минуту будет трассироваться один запрос каждую секунду. Если же `max_traces_burst = 20`, то при поступлении аналогичного потока запросов первые 21 будут оттрассированы, далее аналогично будет трассироваться один запрос в секунду.

> **Важно**
>
> Ограничения на количество трассируемых запросов локальные для узла кластера. Например, если на каждом узле кластера есть правило, в котором указано `max_traces_per_minute = 1`, по этому правилу будет трассироваться не больше одного запроса в минуту **с каждого узла кластера**.

Уровни детализации

Как и в случае с логами, для диагностирования большинства проблем с системой не требуется наличие максимально подробной трассы, поэтому в YDB каждый спан имеет свой уровень, описываемый целым числом от 0 до 15 включительно. Каждое правило в секции `sampling` должно включать в себя уровень детализации генерируемой трассы (`level`), в нее будут включаться спаны с уровнем меньшим, либо равным `level`.

В разделе об архитектуре YDB приводится разделение системы на 5 слоев:

Слой	Копоненты
1	gRPC Proxies
2	Query Processor
3	Distributes Transactions
4	Tablet, System tablet
5	Distributed Storage

Для каждого слоя существует 7 уровней детализации:

Уровень	Значение
`Off`	Отсутствие трассировки
`TopLevel`	Самая низкая детализация, не более 2 спанов на каждый запрос к компоненту
`Basic`	Спаны главных операций компонента
`Detailed`	Самая высокая детализация, применимая для диагностирования проблем в production
`Diagnostic`	Детальная отладочная информация для разработчиков
`Trace`	Очень детальная отладочная информация

Ниже приведено распределение уровней детализации слоев системы по уровням детализации трассы:

Уровень детализации трассы	gRPC Proxies	Query Processor	Distributed Transactions	Tablets	Distributed Storage
0	`TopLevel`	`Off`	`Off`	`Off`	`Off`
1	`TopLevel`	**`TopLevel`**	`Off`	`Off`	`Off`

2	TopLevel	TopLevel	**TopLevel**	Off	Off
3	TopLevel	TopLevel	TopLevel	**TopLevel**	Off
4	TopLevel	TopLevel	TopLevel	TopLevel	**TopLevel**
5	**Basic**	TopLevel	TopLevel	TopLevel	TopLevel
6	Basic	**Basic**	TopLevel	TopLevel	TopLevel
7	Basic	Basic	**Basic**	TopLevel	TopLevel
8	Basic	Basic	Basic	**Basic**	TopLevel
9	Basic	Basic	Basic	Basic	**Basic**
10	**Detailed**	**Detailed**	Basic	Basic	Basic
11	Detailed	Detailed	**Detailed**	Basic	Basic
12	Detailed	Detailed	Detailed	**Detailed**	Basic
13	Detailed	Detailed	Detailed	Detailed	**Detailed**
14	**Diagnostic**	**Diagnostic**	**Diagnostic**	**Diagnostic**	**Diagnostic**
15	**Trace**	**Trace**	**Trace**	**Trace**	**Trace**

Правила

External Throttling

Семантика каждого правила такова: оно выделяет квоту на количество запросов данной категории. Например, если секция `external_throttling` выглядит следующим образом:

```
tracing_config:
  external_throttling:
    - max_traces_per_minute: 60
    - scope:
        request_types:
          - KeyValue.ReadRange
      max_traces_per_minute: 20
```

При достаточном потоке запросов с внешним trace-id, будут трассироваться хотя бы 60 запросов в минуту и хотя бы 20 запросов в минуту типа `KeyValue.ReadRange`. Всего трассироваться будут не более 80 запросов в минуту.

Алгоритм решения следующий: для запроса, имеющего внешний trace-id, определяется набор правил, под которые подходит данный запрос. Запрос потребляет квоту у всех правил, у которых она еще есть. Запрос не трассируется только в том случае, если ни в у одного из правил не осталось квоты.

Sampling

В случае сэмплирования семантика правила аналогична: при достаточно маленьком потоке запросов данной категории, будет сэплироваться хотя бы `fraction` запросов с уровнем хотя бы `level`.

Алгоритм решения аналогичный: для запроса, не имеющего внешнего trace-id (либо из-за его изначального отсутствия, либо из-за принятого ранее решения о не трассировании данного запроса), определяется набор правил, под которые подходит данный запрос. Запрос потребляет квоту у всех правил, у которых она еще есть и которые случайным образом "решили" его сэмплировать. Если ни по одному правилу запрос не нужно сэмлировать (во всех правилах, которые случайным образом "решили" сэмплировать запрос, не осталось квоты), он не сэмплируется. В противном случае уровень детализации определяется, как максимальный среди правил, в квоту которых попал запрос.

Например, при данной конфигурации `sampling`:

```
tracing_config:
  sampling:
    - scope:
        database: /Root/db1
      fraction: 0.5
      level: 5
      max_traces_per_minute: 100
    - scope:
        database: /Root/db1
      fraction: 0.01
      level: 15
      max_traces_per_minute: 5
```

При достаточно маленьком потоке запросов к базе `/Root/db1`, будут сэмплироваться:

- 1% запросов с уровнем детализации 15
- 49.5% запросов с уровнем детализации 5

При достаточно большом потоке запросов к базе `/Root/db1`, будут сэмплироваться:

- 5 запросов в минуту с уровнем детализации 15
- от 95 до 100 запросов в минуту с уровнем детализации 5

Алертинг

Алертинг — это механизм автоматических оповещений, предназначенный для информирования администраторов или специалистов DevOps о событиях, требующих их внимания или вмешательства.

Он позволяет в реальном времени отслеживать критически важные параметры работы базы данных (такие как общее состояние системы, утилизация CPU, ошибки аутентификации, использование дискового пространства), а также быстро реагировать на сбои, ошибки или превышения допустимых порогов нагрузки.

Ниже приведены примеры настраиваемых правил алертинга и особенности их реализации в YDB.

Алертинг в YDB с использованием Prometheus Alerting Rules

Prometheus Alerting Rules — это набор правил, написанных в формате YAML, которые определяют условия для генерации оповещений (алертов). Эти правила основаны на языке запросов PromQL (Prometheus Query Language) и позволяют автоматически выявлять проблемы на основе собранных метрик. Например, вы можете настроить правило, которое сработает, если CPU нагрузка превысит 70% или если диск заполнен более чем на 60%.

Перед началом работы

- Необходим базово настроенный сервер Prometheus. Подробнее об установке можно прочитать здесь
- Требуется базовое понимание написания и использования правил записи
- Настроенные уведомления, чтобы получать оповещения в удобные для вас каналы. Подробнее про настройку в документации

Ключевые параметры правил:

```
- alert: YDBAuthTicketErrors -- Название алерта
  expr: auth_TicketsErrors > 2 -- Условие срабатывания алерта
  for: 1m -- Время, в течение которого условие должно выполняться для активации алерта
  ...
```

- **Где они хранятся?** Правила обычно определяются в отдельных файлах (например, rules.yml) и загружаются в конфигурацию Prometheus.
- **Как они оцениваются?** Сервер Prometheus периодически (по умолчанию каждые 1-2 минуты) вычисляет выражения `expr`, указанные в правилах. Если условие истинно в течение заданного времени `for`, генерируется алерт.

Состояния алерта:

- **Pending** (ожидание): Условие `expr` для срабатывания алерта выполняется (вычисление `expr` вернуло `true`), но заданный промежуток времени `for` ещё не истёк.
- **Firing** (активный): Условие `expr` продолжает выполняться на протяжении всего времени, указанного в параметре `for`, и алерт считается активным.
- **Resolved** (решён): Условие `expr` больше не выполняется, и алерт считается закрытым.

Более подробно об особенности системы и структуре правил смотрите в официальной документации.

Примеры правил алертинга, которые рекомендуются к настройке для каждого кластера

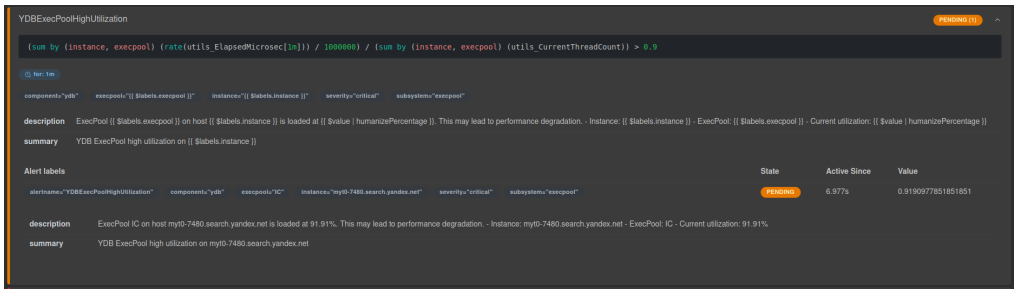
YDB ExecPool High Utilization

Описание: Правило контролирует загрузку пулов в YDB. Срабатывает при превышении 90% утилизации, что может привести к деградации производительности и увеличению времени отклика запросов.
Что делать: Проанализировать нагрузку. В сообщении алерта будет указано название пула, в котором превышена нагрузка. Подробнее про диагностику проблем с CPU можно прочитать в статье

> **Примечание**
>
> Если вы меняете конфигурацию алерта, обязательно обновляйте параметр for вместе с аргументом utils_ElapsedMicrosec, чтобы их значения соответствовали друг другу.

```
- alert: YDBExecPoolHighUtilization
  expr: |
    (
      sum by (instance, execpool) (
        rate(utils_ElapsedMicrosec[1m])
      ) / 1000000
    ) / (
      sum by(instance, execpool) (
        utils_CurrentThreadCount
      )
    ) > 0.9
  for: 1m
  labels:
    severity: critical
    component: ydb
    subsystem: execpool
    instance: "{{ $labels.instance }}"
    execpool: "{{ $labels.execpool }}"
  annotations:
    summary: "YDB ExecPool high utilization on {{ $labels.instance }}"
    description: |
      ExecPool {{ $labels.execpool }} on host {{ $labels.instance }}
      is loaded at {{ $value | humanizePercentage }}.
      This may lead to performance degradation.
      - Instance: {{ $labels.instance }}
      - ExecPool: {{ $labels.execpool }}
      - Current utilization: {{ $value | humanizePercentage }}
```

Пример сработавшего алерта:

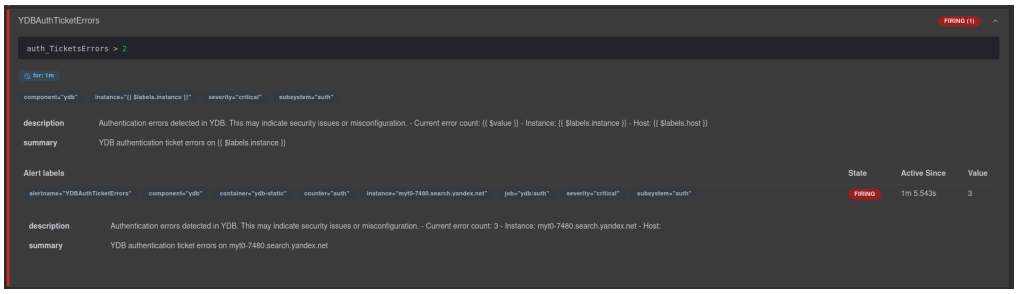


YDB Authentication Errors

Описание: Правило отслеживает ошибки аутентификации в YDB. Срабатывает при появлении более 2 ошибок за период `for`. Это может означать, что кто-то неправильно вводит пароль, есть проблемы с настройками безопасности или что-то не так с конфигурацией системы.
Что делать: Искать в логах ошибки аутентификации и разбираться в причинах.

```
- alert: YDBAuthTicketErrors
  expr: auth_TicketsErrors > 2
  for: 1m
  labels:
    severity: critical
    component: ydb
    subsystem: auth
    instance: "{{ $labels.instance }}"
  annotations:
    summary: "YDB authentication ticket errors on {{ $labels.instance }}"
    description: |
      Authentication errors detected in YDB.
      This may indicate security issues or misconfiguration.
      - Current error count: {{ $value }}
      - Instance: {{ $labels.instance }}
      - Host: {{ $labels.host }}
```

Пример сработавшего алерта:



YDB Storage Usage

Описание: Это правило отслеживает уровень использования дискового пространства. Оно срабатывает, когда заполненность хранилища достигает критических значений. Рекомендуется настроить два порога: уведомление при использовании 80% диска и критическое оповещение — при 90%.
Что делать: Найти причину превышения базой ожидаемого размера. Удалить ненужные записи (например, старые логи) или увеличить лимит на размер базы, если это требуется. Подробнее про диагностику проблем с дисковым пространством можно прочитать в статье

Warning Storage Usage (80%)

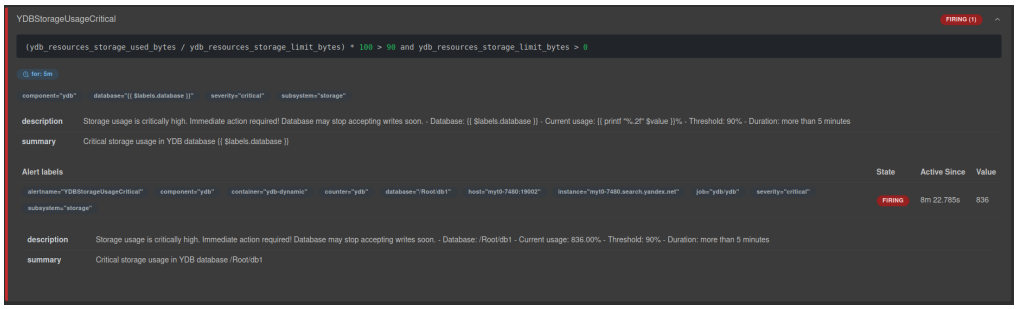
```
- alert: YDBStorageUsageWarning
  expr: |
    (ydb_resources_storage_used_bytes / ydb_resources_storage_limit_bytes) * 100 > 80
    and ydb_resources_storage_limit_bytes > 0
  for: 5m
  labels:
    severity: warning
    component: ydb
    subsystem: storage
    database: "{{ $labels.database }}"
  annotations:
    summary: "High storage usage in YDB database {{ $labels.database }}"
    description: |
      Storage usage is above warning threshold.
      Consider cleaning up old data or increasing storage capacity.
      - Database: {{ $labels.database }}
      - Current usage: {{ printf "%.2f" $value }}%
      - Threshold: 80%
      - Duration: more than 5 minutes
```

Critical Storage Usage (90%)

```
- alert: YDBStorageUsageCritical
  expr: |
```

```
    (ydb_resources_storage_used_bytes / ydb_resources_storage_limit_bytes) * 100 > 90
    and ydb_resources_storage_limit_bytes > 0
  for: 5m
  labels:
    severity: critical
    component: ydb
    subsystem: storage
    database: "{{ $labels.database }}"
  annotations:
    summary: "Critical storage usage in YDB database {{ $labels.database }}"
    description: |
      Storage usage is critically high. Immediate action required!
      Database may stop accepting writes soon.
      - Database: {{ $labels.database }}
      - Current usage: {{ printf "%.2f" $value }}%
      - Threshold: 90%
      - Duration: more than 5 minutes
```

Пример сработавшего алерта:



Полный конфигурационный файл

Полный конфигурационный файл со всеми правилами указанными выше доступен по ссылке

Рекомендации по настройке

- **Пороговые значения:** Настройте пороги в параметре `expr` в соответствии с вашими SLA и характеристиками нагрузки.
- **Время ожидания:** Увеличьте параметр `for` для менее критичных алертов, чтобы избежать ложных срабатываний.

Диагностика изолированного кластера

Если кластер скрыт за сетевыми барьерами, прямой мониторинг его состояния невозможен. В таком случае для выявления проблем во внутренней среде кластера можно применять механизм сохранения его состояния в файл.

Сбор диагностической информации

Для сбора диагностической информации используйте команду:

```
ydb [global options...] admin cluster diagnostics collect \
  --duration 200 \
  --period 15 \
  --output diagnostics.tar
```

Параметры команды

- `--duration` — количество секунд, в течение которых будет собираться информация о кластере.
- `--period` — интервал в секундах между сборами метрик.
- `--output` — путь и имя файла в формате `.tar`, в который будет записана вся собранная информация о кластере.
- `--no-sanitize` — позволяет получить полный отчёт, отключая очистку от пользовательских имён таблиц, колонок и запросов (которые по умолчанию вырезаются).

Команда записывает в указанный файл внутреннее состояние узлов кластера на протяжении заданного времени. Каждые `period` секунд производится сбор метрик с каждого узла кластера и запись в файл.

Содержимое архива

Полученный файл является архивом и содержит:

- Метрики, собранные с узлов кластера.
- Информацию о кластере.

Все данные записаны в формате JSON в человекочитаемом виде.

Информация о кластере

Информация о кластере включает:

- Отчет Health Check.
- Запросы к таблицам раздела кластера `/Root/.sys` и их ответы.
- Текущую конфигурацию подсистем кластера.
- Текущее состояние узлов кластера.

Резервное копирование и восстановление

YDB изначально спроектирована для сохранности данных при отказах оборудования: доступны варианты избыточности для разного числа зон доступности, стоек, хостов, дисков и других компонентов (см. режимы работы кластера). Это снижает риск потери данных из-за аппаратных сбоев.

Однако данные могут быть утрачены или испорчены **логически**: ошибки или злоумышленные действия могут привести к массовому удалению/искажению данных легитимными для СУБД операциями. В таких случаях отказоустойчивость кластера не заменяет **отдельную копию** данных вне кластера.

Резервное копирование применяется для защиты от таких сценариев и позволяет восстановить данные из резервной копии. Рекомендуется хранить копии на отдельном носителе или в облачном объектном хранилище (например, через выгрузку в файлы или export в S3).

YDB предоставляет несколько решений для выполнения резервного копирования и восстановления. Концептуальную информацию и сравнение методов резервного копирования см. в концепциях резервного копирования.

- Резервное копирование в файлы и восстановление с помощью YDB CLI.
- Резервное копирование в S3-совместимое хранилище и восстановление с помощью YDB CLI.
- Резервное копирование системных таблеток на локальную файловую систему хостов кластера.

YDB CLI

Файлы

Для выполнения резервного копирования в файлы применяются команды:

- `ydb admin cluster dump` — для резервного копирования метаданных кластера
- `ydb admin database dump` — для резервного копирования базы данных
- `ydb tools dump` — для резервного копирования отдельных схемных объектов или директорий

Узнать больше об этих командах можно в Выгрузка в файловую систему.

Для выполнения восстановления из файловой резервной копии применяются команды:

- `ydb admin cluster restore` — для восстановления метаданных кластера из резервной копии
- `ydb admin database restore` — для восстановления базы данных из резервной копии
- `ydb tools restore` — для восстановления отдельных схемных объектов или директорий из резервной копии

Узнать больше об этих командах можно в Загрузка из файловой системы.

S3-совместимое хранилище

Для выполнения резервного копирования в S3-совместимое хранилище (например, AWS S3) применяется команда `ydb export s3`. Перейдите по ссылке в справочник по YDB CLI для получения информации о данной команде.

Для выполнения восстановления из резервной копии, созданной в S3-совместимом хранилище, применяется команда `ydb import s3`. Перейдите по ссылке в справочник по YDB CLI для получения информации о данной команде.

NFS

Для выполнения резервного копирования в NFS на хостах, где запущена YDB, применяется команда `ydb export nfs`. Перейдите по ссылке в справочник по YDB CLI для получения информации о данной команде.

Для выполнения восстановления из резервной копии, созданной в NFS на хостах, где запущена YDB, применяется команда `ydb import nfs`. Перейдите по ссылке в справочник по YDB CLI для получения информации о данной команде.

Подробнее про настройку NFS для резервного копирования и восстановления см. рецепт Резервное копирование и восстановление через NFS.

Примечание

Скорость операций резервного копирования и восстановления в/из S3-совместимого хранилища или NFS подобрана таким образом, чтобы минимизировать влияние на пользовательскую нагрузку. Для управления скоростью операций настройте лимиты для соответствующей очереди брокера ресурсов.

Примечание

При выполнении выгрузки в корневом каталоге базы данных создается директория с именем `export_*`, где `*` — это числовая часть идентификатора выгрузки. В данной директории размещаются таблицы, содержащие консистентный снапшот выгружаемых данных на момент начала выгрузки. После успешного завершения операции резервного копирования директория `export_*` вместе с содержимым удаляется.

Коллекции резервных копий

Коллекции резервных копий обеспечивают инкрементальное резервное копирование и восстановление на момент времени для производственных нагрузок. Концептуальную информацию и детали архитектуры см. в Коллекциях резервных копий.

Коллекции резервных копий рекомендуются для производственных сред с регулярным расписанием резервного копирования и больших наборов данных, где инкрементальные изменения значительно меньше общего объёма. Для более простых сценариев (разовые миграции, среды разработки, небольшие базы данных) рассмотрите использование export/import или dump/restore.

Пошаговые инструкции по настройке и использованию коллекций резервных копий см. в:

- Коллекциях резервных копий — архитектура, концепции и ограничения
- Рецептах для коллекций резервных копий — типовые сценарии и примеры

Резервное копирование системных таблеток

Механизм резервного копирования системных таблеток обеспечивает инкрементальное копирование метаданных кластера на локальную файловую систему хостов кластера.

Концептуальную информацию и принцип работы см. в Концепциях резервного копирования.

Пошаговые инструкции по включению и восстановлению см. в Рецептах для резервного копирования системных таблеток.

Первоначальная синхронизация асинхронной репликации с помощью резервного копирования и восстановления

При этом способе первоначальной синхронизации исходные данные переносятся в приёмник из резервной копии в S3-совместимом хранилище. Затем асинхронная репликация применяет накопленные изменения и продолжает синхронизировать базы данных. Во время переноса приложения могут продолжать записывать данные в источник.

Этот способ позволяет использовать серверные операции резервного копирования и восстановления для переноса большого объёма данных. При первоначальном CDC-сканировании все исходные данные передаются через потоки изменений, а отдельная резервная копия не требуется. Различия в ресурсных и эксплуатационных затратах приведены в сравнении способов первоначальной синхронизации.

Подготовка

В примере таблица `original_table` базы `/Root/source` переносится в таблицу `replica_table` базы `/Root/target`. Экземпляр репликации `my_replication` создаётся в приёмнике. Команды используют профили YDB CLI `source` и `target` для подключения к соответствующим базам данных.

Перед началом работы:

- Настройте профили и проверьте подключение к обеим базам с помощью команды `scheme ls`.
- Подготовьте права для асинхронной репликации, экспорта в источнике и импорта в приёмнике. Пользователю, создающему экземпляр репликации, потребуется возможность изменять восстановленные таблицы для подключения их в качестве реплик.
- В приёмнике создайте секрет `source_token` с токеном для подключения репликации к источнику.
- Подготовьте бакет `mybucket` и аутентификацию в S3. В примерах используется эндпоинт `storage.yandexcloud.net`, а ключи доступа читаются из переменных окружения или файла `~/.aws/credentials`. Эндпоинт должен быть доступен с серверов обеих баз данных: источник записывает резервную копию, приёмник читает её.
- Выберите свободный префикс `replication-bootstrap` в бакете. Не изменяйте содержимое выгрузки до завершения импорта.
- Убедитесь, что таблица `replica_table` отсутствует в приёмнике. Команда `import s3` создаёт таблицу вместе с данными и не перезаписывает существующие таблицы. Проверить путь можно командой `scheme ls`.

На время первоначальной синхронизации сохраните состав реплицируемых таблиц, их столбцов и индексов. Схемы таблиц в резервной копии должны соответствовать схемам источника при запуске репликации.

> **Важно**
>
> До создания репликации выберите период доступности неподтверждённых изменений. При `AVAILABILITY_PERIOD = 'UNLIMITED'` изменения сохраняются до подтверждения обработки; обеспечьте место для их накопления в источнике. При конечном периоде планируйте экспорт, импорт и обработку накопленных изменений так, чтобы репликация успела подтвердить их обработку до истечения этого периода. Состояние `PAUSED` не останавливает отсчёт возраста сообщений.

1. Создайте репликацию без первоначального сканирования

Выполните в приёмнике запрос CREATE ASYNC REPLICATION с параметром `INITIAL_SCAN = FALSE`. YQL-запросы из этого руководства можно выполнить командой `sql` с профилем `target`.

```
CREATE ASYNC REPLICATION my_replication
FOR original_table AS replica_table
WITH (
    CONNECTION_STRING = 'grpcs://source.example.com:2135/?database=/Root/source',
    TOKEN_SECRET_PATH = 'source_token',
    INITIAL_SCAN = FALSE,
    AVAILABILITY_PERIOD = 'UNLIMITED'
);
```

Адрес `source.example.com:2135` в примере соответствует эндпоинту источника. Параметры `CONNECTION_STRING` и `TOKEN_SECRET_PATH` задают подключение, которое приёмник будет использовать для создания и чтения потоков изменений.

В примере явно выбран период доступности `'UNLIMITED'`, который также используется по умолчанию. Для конечного периода укажите положительное значение типа `Interval`, например `AVAILABILITY_PERIOD = Interval('PT48H')`. Изменить этот параметр через `ALTER ASYNC REPLICATION` после создания нельзя.

Репликация создаётся в состоянии `PAUSED` и подготавливает потоки изменений в источнике без выгрузки первоначальных данных. Таблица `replica_table` пока не создаётся и не подключается к репликации. Её создаст операция импорта.

2. Дождитесь готовности потоков изменений

Создание потоков выполняется асинхронно. До запуска экспорта проверьте, что они готовы для всех реплицируемых таблиц, включая таблицы реализации индексов, если они входят в репликацию.

Получите описание репликации в формате JSON:

```
ydb -p target scheme describe my_replication --format proto-json-base64
```

Проверьте следующие поля:

- `paused`: репликация ожидает запуска после импорта. Наличие этого поля само по себе не подтверждает готовность потоков.
- `items`: перечень объектов с путями `source_path` и `destination_path`. Дождитесь появления всех ожидаемых объектов.
- `source_changefeed_name` у каждого объекта: имя потока изменений. Имя назначается автоматически; не задавайте его вручную.

Для таблицы `original_table` проверьте состояние потока в источнике:

```
ydb -p source scheme describe original_table --format proto-json-base64
```

В массиве `changefeeds` найдите поток с именем из `source_changefeed_name`. Его поле `state` должно иметь значение `STATE_ENABLED`.

Для элемента, соответствующего синхронному или уникальному индексу, `source_path` указывает на индекс. Его таблица реализации находится по пути `source_path/indexImplTable`. Проверяйте массив `changefeeds` в описании этой таблицы.

Также проверьте параметры топика потока изменений: наличие читателя `replicationConsumer`, его политику доступности и значение `RetentionPeriod`. Путь к потоку зависит от объекта:

Объект	Путь к потоку изменений
Таблица	`source_path/source_changefeed_name`
Таблица реализации индекса	`source_path/indexImplTable/source_changefeed_name`

В этих путях используйте значения полей из описания репликации. Для проверки выполните команду:

```
ydb -p source scheme describe <changefeed-path>
```

Где:

- `<changefeed-path>`: путь к потоку изменений, например `/Root/source/original_table/stream-name`. Вместо `stream-name` используйте имя из описания репликации.

В таблице читателей для `replicationConsumer` проверьте:

- При `AVAILABILITY_PERIOD = 'UNLIMITED'` поле `Important` должно иметь значение `Yes`.
- При конечном периоде поле `Important` должно иметь значение `No`, а `Availability period` — соответствовать выбранному интервалу.

Повторите проверки для каждого объекта из `items`. Неподтверждённые сообщения защищены политикой читателя, поэтому оценивать доступность накопленных изменений только по `RetentionPeriod` нельзя. Подробнее см. Хранение изменений в источнике.

> **Важно**
>
> Запускайте экспорт только после готовности всех потоков изменений. Если резервная копия сделана раньше, изменения между её созданием и запуском потоков могут быть потеряны для репликации. Использовать произвольную ранее созданную резервную копию для этой процедуры нельзя.

3. Экспортируйте исходные данные

Запустите экспорт в S3 с профилем источника:

```
ydb -p source export s3 \
  --s3-endpoint storage.yandexcloud.net \
  --bucket mybucket \
  --destination-prefix replication-bootstrap \
  --include original_table \
  --include-index-data true
```

Выгрузка таблицы будет размещена под префиксом `replication-bootstrap/original_table`. Параметр `--include-index-data true` сохраняет также данные индексных таблиц, если у исходной таблицы есть индексы. На шаге импорта они будут загружены из резервной копии.

Для нескольких таблиц укажите `--include` для каждой из них в одной операции экспорта: сервер создаёт согласованную копию таблиц. Состав выгрузки должен соответствовать набору объектов репликации.

Команда возвращает идентификатор серверной операции. Дождитесь её успешного завершения с помощью `operation get`:

```
ydb -p source operation get <export-operation-id>
```

Где:

- `<export-operation-id>`: идентификатор, полученный при запуске экспорта.

Переходите к импорту только при значениях `ready = true`, `status = SUCCESS` и `progress = Done` в табличном выводе. Успешный запуск команды экспорта ещё не означает завершение выгрузки.

Приложения могут продолжать изменять данные источника. Потоки изменений сохраняют эти изменения согласно выбранному `AVAILABILITY_PERIOD`, в том числе пока репликация находится в состоянии `PAUSED`.

4. Импортируйте данные в приёмник

После успешного экспорта запустите импорт из S3 с профилем приёмника:

```
ydb -p target import s3 \
  --s3-endpoint storage.yandexcloud.net \
  --bucket mybucket \
  --item src=replication-bootstrap/original_table,dst=replica_table \
  --index-population-mode import
```

Параметр `--item` сопоставляет префикс таблицы в резервной копии с целевым путём `replica_table`, указанным в `CREATE ASYNC REPLICATION`. Режим `--index-population-mode import` загружает данные индексов, сохранённые на предыдущем шаге. Для нескольких таблиц укажите отдельный `--item` для каждой пары путей.

Дождитесь успешного завершения операции с помощью `operation get`:

```
ydb -p target operation get <import-operation-id>
```

Где:

- `<import-operation-id>`: идентификатор, полученный при запуске импорта.

Убедитесь, что `ready = true`, `status = SUCCESS` и `progress = Done`. Проверьте созданную таблицу:

```
ydb -p target scheme describe replica_table
```

До запуска репликации восстановленная таблица является обычной таблицей. Не записывайте в неё данные приложения: такие записи не синхронизируются с источником и могут быть перезаписаны при применении изменений.

5. Запустите репликацию изменений

После завершения импорта всех таблиц выполните в приёмнике запрос ALTER ASYNC REPLICATION:

```
ALTER ASYNC REPLICATION my_replication SET (STATE = 'STANDBY');
```

Приёмник проверит соответствие схем восстановленных таблиц источнику и подключит их как объекты-реплики. Проверка включает первичный ключ, столбцы, их типы, семейства столбцов и индексы. Подключённые таблицы становятся доступны только для чтения, а репликация начинает применять накопленные изменения.

Поддерживаются синхронные вторичные индексы, в том числе уникальные. Восстановленные векторные индексы удаляются при подключении таблицы к репликации. Подробнее см. Ограничения.

Записи читаются из потоков изменений, созданных перед экспортом. Среди них могут быть изменения, уже отражённые в резервной копии. Пока накопленные записи не обработаны, содержимое приёмника может изменяться и ещё не соответствует текущему состоянию источника.

Проверьте состояние и отставание:

```
ydb -p target scheme describe my_replication --stats
```

После запуска в описании поле `paused` сменяется полем `running`. Статистика содержит отставание репликации и отдельных объектов. Дождитесь обработки накопленных изменений; отсутствие значения отставания не означает, что оно равно нулю. Значение может быть недоступно при остановке читателя, потере его сессии или отсутствии свежих отчётов. Подробнее см. Репликация изменений.

После этого репликация продолжает переносить новые изменения. Если цель переноса состоит в миграции приложений, выполните отдельную процедуру завершения асинхронной репликации перед переключением пишущей нагрузки в приёмник.

Ошибки и повторные попытки

- Если экспорт или импорт завершился ошибкой, проверьте статус и описание ошибки через `operation get`. Сохраняйте репликацию в состоянии `PAUSED`, пока импорт всех таблиц не завершится успешно. При повторном импорте учитывайте, что целевые таблицы не должны существовать; ошибка операции не является подтверждением отсутствия созданных объектов.
- Если запуск репликации завершился ошибкой, получите JSON-описание репликации и проверьте поле `error`. В частности, проверьте наличие всех целевых таблиц, соответствие схем и права доступа. Не запускайте импорт поверх таблиц, которые уже подключены как реплики.
- Если нужные CDC-записи удалены из-за истечения срока хранения, продолжение чтения не обеспечит полноту данных. Повторите первоначальную синхронизацию с новыми потоками изменений и новой резервной копией.
- Перезапуск контроллера репликации сохраняет состояние `PAUSED`; он не заменяет явный перевод в `STANDBY`. При прерывании сессии CLI проверяйте ранее запущенные серверные операции по их идентификаторам, прежде чем запускать новые.

Подробнее о временных и критичных ошибках см. в разделе Обработка ошибок в процессе асинхронной репликации.

YDB для разработчиков приложений

В этом разделе документации YDB описано всё, что нужно знать для разработки приложений, взаимодействующих с YDB.

Основные материалы:

- Начало работы с YDB в роли разработчика приложений
- Тестовое приложение
- Генерация кода из YQL (sqlc-ydb)
- Туториал по YQL - Обзор
- Выбор первичного ключа для:
  - Строчных таблиц
  - Колоночных таблиц
- Партицирование
- Вторичные индексы
- Загрузка данных в YDB
- Постраничный вывод
- Использование таймаутов
- Системные представления базы данных
- Change Data Capture
- Пользовательские атрибуты таблиц
- Workload Manager — управление потреблением ресурсов
- Оптимизация запросов
- Справка:
  - YQL - Обзор
  - Справка по YDB SDK
  - YDB CLI
  - Kafka API

Если же вам интересная разработка ядра YDB или сопутствующих проектов, обратитесь к документации для контрибьюторов.

Начало работы с YDB в роли разработчика приложений

Прежде всего, вам необходимо получить доступ к кластеру YDB. Следуйте инструкциям по быстрому запуску, чтобы получить базовый локальный инстанс. Также вы можете совместно со своей DevOps командой создать production кластер или воспользоваться Yandex Managed Service for YDB.

Второй шаг — это разработка схемы данных, если вы создаёте приложение с нуля, или адаптирование схемы данных существующего приложения, если вы переходите с другой системы управления базами данных.

Параллельно с проектированием схемы вам необходимо настроить среду разработки для взаимодействия с YDB. Есть несколько основных аспектов, которые рассматриваются ниже.

Выбор API

Выберите YDB API, который вы будете использовать; есть несколько вариантов:

- Для основных языков программирования рекомендуется использовать YDB SDK. Они предоставляют высокоуровневые API и реализуют рекомендованные подходы по работе с YDB. YDB SDK доступны для нескольких популярных языков и стремятся к равенству функциональности, но не все из них являются полнофункциональными. Обратитесь к таблице сравнения функций SDK, чтобы проверить, соответствует ли SDK для выбранного языка программирования вашим потребностям, также с её помощью можно выбрать язык программирования с лучшим охват функций, если вы гибки в этом вопросе.
- Если вас интересует функция YDB топиков, стоит отметить, что они предоставляют Kafka-совместимый API. Перейдите по этой ссылке, если этот вариант использования для вас актуален.
- В крайнем случае, собственный API YDB основан на протоколе gRPC, вокруг которого есть экосистема, включая генерацию кода клиентов. gRPC спецификации YDB API размещены на GitHub, и вы можете использовать их в своём приложении. Сгенерированные клиенты являются низкоуровневыми и потребуют дополнительной работы для правильной обработки таких аспектов, как ретраи и таймауты. Следуйте по этому пути только если другие варианты невозможны, и вы знаете, что делаете.

Установите пререквизиты

Выберите конкретный язык программирования, который вы будете использовать. Установите соответствующий YDB SDK.

Кроме того, рекомендуется настроить хотя бы один из доступных способов запуска произвольных запросов в целях отладки. Выберите часть из них по своим предпочтениям:

- YDB CLI
- YDB UI
- Любая среда разработки SQL, поддерживающая JDBC.

Начать писать код

Для пути с YDB SDK

– Пройдите туториал по YQL, чтобы познакомиться с SQL диалектом YDB.

- Изучите примеры приложений, чтобы узнать, как выглядит работа с SDK.
- Ознакомьтесь с рецептами SDK для типичных случаев использования SDK, к которым вы сможете обратиться позже.
- Ознакомьтесь с рекомендациями по обработке ошибок YDB SDK.
- Ориентируйтесь на возможности вашего IDE по навигации по коду при работе с SDK.

Тестирование

Как написать тесты на приложения, работающие с YDB:

- Для функциональных тестов вы можно имитировать (mock) ответы YDB, используя подходящий фреймворк тестирования для выбранного языка программирования.
- Для интеграционных тестов можно запустить одноузловой YDB в вашем CI/CD окружении с помощью Docker образа или исполняемого файла аналогично тому, как это делается в статье быстрого старта. Рекомендуется тестироваться с несколькими версиями YDB: той, которая у вас есть в production, чтобы проверить наличие проблем, с которыми вы можете столкнуться при обновлении своего приложения, а также более новые версиями YDB, чтобы на ранней стадии выявлять потенциальные проблемы обновления YDB.
- Для тестов производительности рекомендуется использовать кластер, развёрнутый в соответствии с инструкциями для production использования, поскольку использование одного узла не даст реалистичных результатов. Вы можете запускать ydb workload, если хотите увидеть, как ваш кластер YDB работает на общих сценариях ещё до разработки своего приложения. Затем вы можете использовать исходный код библиотеки, лежащей в основе этого инструмента в качестве примера того, как написать собственные тесты на производительность с YDB. Было бы здорово, если бы вы могли законтрибьютить анонимизированную версию вашей рабочей нагрузки, чтобы её можно было включить в тестирование производительности самого YDB.

Что дальше

Вышеизложенного должно быть достаточно, чтобы начать разработку приложений, взаимодействующих с YDB. В процессе работы используйте справочную документацию YQL и YDB SDK, а также другую ресурсы в этом разделе документации.

В случае возникновения каких-либо проблем, не стесняйтесь обсуждать их в YDB Discord или Telegram чате.

Потоковые запросы

Практические аспекты работы с потоковыми запросами:

- Локальные и внешние топики — обращение к топикам текущей и другой базы в YQL-запросах
- Типичные шаблоны — минимальные примеры для быстрого старта
- Запись в таблицы — как потоковые запросы позволяют записывать данные в таблицы YDB в режиме реального времени.
- Обогащение данных — способы обогащения данных в потоке с использованием справочников.
- Форматы данных при чтении/записи топиков — поддерживаемые форматы данных при работе с топиками, примеры их использования.
- Гарантии доставки данных — уровень гарантий, наблюдаемые аномалии при оконной агрегации и рекомендации.
- Чекпоинты — механизм сохранения состояния обработки потока для обеспечения отказоустойчивости и возможности восстановления.
- Watermarks — механизм отслеживания прогресса времени в потоке данных.

См. также

- Рецепты работы с потоковыми запросами
- Описание потоковых запросов

Типичные шаблоны потоковых запросов

В этом разделе собраны минимальные примеры потоковых запросов для наиболее распространённых сценариев. Сначала описан базовый шаблон чтения данных из топика, затем — варианты полноценной работы с данными: обработка данных и запись результатов в топик в формате JSON, в топик в виде строки и в таблицу. Каждый пример можно использовать как отправную точку для собственных задач.

В примерах ниже используются локальные и внешние топики. Обозначения:

- `ext_source` — заранее созданный `external data source`;
- `input_topic` — топик, откуда читаются данные;
- `output_topic` — топик, куда записываются результаты;
- `output_table` — таблица YDB, куда записываются результаты.

Чтение данных из топика

Чтение структурированных сообщений выполняется с помощью `SELECT ... FROM ... WITH (FORMAT, SCHEMA)`. Блок `WITH` задаёт формат входных данных и схему — какие поля ожидаются в каждом сообщении и их типы.

Следующий фрагмент используется внутри CREATE STREAMING QUERY в блоке `DO BEGIN ... END DO`:

```
SELECT
    Id,
    Name
FROM
    topic_name  -- локальный топик; для внешнего: ext_source.topic_name
WITH (
    FORMAT = json_each_row,
    SCHEMA = (
        Id Uint64 NOT NULL,
        Name Utf8 NOT NULL
    )
);
```

Подробнее о форматах: Форматы данных при чтении/записи из топиков.

Этот шаблон используется во всех последующих примерах.

Запись в топик (JSON)

Запрос читает события из входного топика, формирует JSON-объект из отдельных полей и записывает результат в выходной топик. Функция `AsStruct` создаёт структуру из указанных полей, `Yson::From` преобразует её в Yson, `Yson::SerializeJson` сериализует в JSON-строку, а `ToBytes` конвертирует в тип `String`, который требуется для записи в топик.

```
CREATE STREAMING QUERY write_json_example AS
DO BEGIN

INSERT INTO ext_source.output_topic -- или локальный топик output_topic
SELECT
    -- Формирование JSON из отдельных полей
    ToBytes(Unwrap(Yson::SerializeJson(Yson::From(
        AsStruct(Id AS id, Name AS name)
    ))))
FROM
    input_topic -- или внешний топик ext_source.input_topic
WITH (
    FORMAT = json_each_row,  -- Формат входных данных
    SCHEMA = (               -- Схема входных данных
        Id Uint64 NOT NULL,
        Name Utf8 NOT NULL
    )
);

END DO
```

Подробнее о функциях:

- AsStruct
- Yson::From
- Yson::SerializeJson
- Unwrap
- ToBytes.

Запись в топик (строка)

Запрос читает события из входного топика и записывает в выходной топик одно поле в виде строки. Для записи в топик строк `SELECT` должен возвращать одну колонку типа `String` или `Utf8`.

```
CREATE STREAMING QUERY write_utf8_example AS
DO BEGIN

INSERT INTO output_topic -- или внешний топик ext_source.output_topic
SELECT
    Name
FROM
    ext_source.input_topic -- или локальный топик input_topic
WITH (
    FORMAT = json_each_row,  -- Формат входных данных
```

```
    SCHEMA = (               -- Схема входных данных
        Id Uint64 NOT NULL,
        Name Utf8 NOT NULL
    )
);

END DO
```

Подробнее о форматах записи: Форматы при записи данных.

Запись в таблицу

Запрос читает события из топика и записывает их в таблицу `output_table`. Таблица должна быть создана заранее со схемой, соответствующей выбираемым колонкам.

> **Важно**
>
> Запись в таблицы в потоковых запросах поддерживается **только в режиме UPSERT**. Операция `INSERT INTO` не поддерживается, так как при повторной обработке событий (гарантия at-least-once) она привела бы к дублированию строк. При `UPSERT`, если строка с таким первичным ключом уже существует, она будет обновлена, иначе будет вставлена новая строка, а `INSERT INTO` завершится с ошибкой.

```
CREATE STREAMING QUERY write_table_example AS
DO BEGIN

-- Запись в таблицу (только UPSERT, INSERT не поддерживается)
UPSERT INTO output_table
SELECT
    Id,
    Name
FROM
    ext_source.input_topic -- или локальный топик input_topic
WITH (
    FORMAT = json_each_row,  -- Формат входных данных
    SCHEMA = (               -- Схема входных данных
        Id Uint64 NOT NULL,
        Name Utf8 NOT NULL
    )
);

END DO
```

Подробнее: Запись в таблицы.

См. также

- Локальные и внешние топики
- CREATE STREAMING QUERY
- Быстрый старт: чтение и запись в топики

Запись в таблицы

Запись в таблицы позволяет сохранять результаты потокового запроса для последующего анализа обычными SQL-запросами. Например, можно агрегировать события из потока и сохранять итоги в таблицу.

Для записи используется UPSERT INTO — вставка новой строки или обновление существующей по первичному ключу. Операция UPSERT идемпотентна по первичному ключу: повторная запись той же строки приводит к обновлению, а не к дублированию. Это важно, потому что потоковые запросы обеспечивают гарантию at-least-once — при восстановлении из чекпоинта часть событий может быть обработана повторно.

Внимание

Не поддерживаются:

- команда INSERT INTO — используйте UPSERT INTO. INSERT INTO не применяется, потому что при повторной обработке событий (гарантия at-least-once) она привела бы к дублированию строк;
- запись в таблицы YDB, находящиеся во внешних БД. В текущей версии запись возможна только в локальные таблицы.

Пример

Запрос читает события из топика и записывает их в таблицу `output_table`. Поле `Ts` преобразуется из строки в тип `Timestamp` с помощью `CAST`, а Unwrap снимает опциональность результата.

```
CREATE STREAMING QUERY query_with_table_write AS
DO BEGIN

-- Чтение из топика и запись в таблицу
UPSERT INTO
    output_table
SELECT
    -- Преобразование строки в Timestamp
    Unwrap(CAST(Ts AS Timestamp)) AS Ts,
    Country,
    Count
FROM
    -- Чтение событий из топика
    input_topic
WITH (
    -- Формат данных в топике
    FORMAT = json_each_row,
    -- Схема данных
    SCHEMA = (
        Ts String NOT NULL,
        Count Uint64 NOT NULL,
        Country Utf8 NOT NULL
    )
);

END DO
```

Ограничения

В одном потоковом запросе:

- Нельзя использовать одну и ту же таблицу для обогащения потока с использованием `JOIN` и для записи результата запроса.
- Нельзя писать в одну и ту же таблицу несколько раз.

Обогащение данных

Обогащение данных — добавление к событиям из потока дополнительной информации из справочника. Например, событие содержит только идентификатор, а справочник позволяет добавить к нему название или другие атрибуты. В качестве справочника можно использовать данные из локальной таблицы или из объектного хранилища S3.

В потоковых запросах справочник подключают с помощью конструкции `JOIN`. Поток должен быть слева, справочник — справа.

> **Важно**
>
> Справочник полностью загружается в память при запуске запроса. Если данные в справочнике изменились, для получения актуальной версии справочника необходимо перезапустить запрос — удалить его с помощью DROP STREAMING QUERY и создать заново с помощью CREATE STREAMING QUERY.

Возможно обогащение данных из локальных и внешних топиков.

В примерах ниже:

- `ext_source` — заранее созданный внешний источник данных для топиков в другой базе YDB;
- `input_topic` и `output_topic` — топики в текущей или внешней базе YDB

Потоковые запросы для обогащения данных

Запросы в примерах ниже читают события из входного топика, присоединяют к каждому событию название сервиса из справочника по `ServiceId` и записывают результат в выходной топик.

Подробнее об использованных в запросах функциях:

- TableRow
- Yson::From
- Yson::SerializeJson
- Unwrap
- ToBytes.

Обогащение из локальной таблицы

В данном примере справочник хранится в таблице `services_dict` в текущей базе данных.

Создайте потоковый запрос, выполняющий обогащение:

```
CREATE STREAMING QUERY query_with_table_join AS
DO BEGIN

-- Чтение событий из входного топика
$topic_data = SELECT
    *
FROM
    ext_source.input_topic -- или локальный топик input_topic
WITH (
    FORMAT = json_each_row,
    SCHEMA = (
        Time String NOT NULL,
        ServiceId Uint32 NOT NULL,
        Message String NOT NULL
    )
);

-- Присоединение справочника к потоку по ServiceId
$joined_data = SELECT
    s.Name AS Name,
    t.*
FROM
    $topic_data AS t
LEFT JOIN
    services_dict AS s
ON
    t.ServiceId = s.ServiceId;

-- Запись в выходной топик (JSON)
INSERT INTO
    output_topic -- или внешний топик ext_source.output_topic
SELECT
    ToBytes(Unwrap(Yson::SerializeJson(Yson::From(TableRow()))))
FROM
    $joined_data;

END DO
```

Обогащение из S3

Справочник хранится в S3 и подключается через внешний источник данных.

Создайте дополнительный внешний источник данных для чтения справочника из S3:

```
-- Источник данных S3 для чтения справочника
CREATE EXTERNAL DATA SOURCE s3_source WITH (
    SOURCE_TYPE = "ObjectStorage",
    LOCATION = "<s3_endpoint>",
```

```
    AUTH_METHOD = "NONE"
);
```

Где:

- `<s3_endpoint>` — URL S3-хранилища, например `https://storage.yandexcloud.net/<bucket>/` для Yandex Cloud.

Создайте потоковый запрос, выполняющий обогащение:

```
CREATE STREAMING QUERY query_with_join AS
DO BEGIN

-- Чтение событий из входного топика
$topic_data = SELECT
    *
FROM
    input_topic -- или внешний топик ext_source.input_topic
WITH (
    FORMAT = json_each_row,
    SCHEMA = (
        Time String NOT NULL,
        ServiceId Uint32 NOT NULL,
        Message String NOT NULL
    )
);

-- Чтение справочника сервисов из S3
$s3_data = SELECT
    *
FROM
    s3_source.`file.csv`
WITH (
    FORMAT = csv_with_names,
    SCHEMA = (
        ServiceId Uint32,
        Name Utf8
    )
);

-- Присоединение справочника к потоку по ServiceId
$joined_data = SELECT
    s.Name AS Name,
    t.*
FROM
    $topic_data AS t
LEFT JOIN
    $s3_data AS s
ON
    t.ServiceId = s.ServiceId;

-- Запись результата в выходной топик в формате JSON
INSERT INTO
    ext_source.output_topic -- или локальный топик output_topic
SELECT
    ToBytes(Unwrap(Yson::SerializeJson(Yson::From(TableRow()))))
FROM
    $joined_data;

END DO
```

Подробнее о форматах данных (`json_each_row`, `csv_with_names` и др.): Форматы данных при чтении/записи из топиков.

Форматы данных при чтении/записи из топиков

В данном разделе описываются форматы данных потоковых запросов, поддерживаемые в YDB при чтении из топиков, и список поддерживаемых YQL типов для каждого формата данных.

Поддерживаемые форматы данных

YDB поддерживает встроенные форматы данных (предопределённые по умолчанию) и пользовательские форматы (настраиваемые пользователем под специфические задачи).

Список встроенных в YDB форматов данных приведен ниже:

- `csv_with_names`
- `tsv_with_names`
- `json_list`
- `json_each_row`
- `json_as_string`
- `parquet`
- `raw`

Примеры парсинга данных в пользовательских форматах.

В примерах на этой странице `input_topic` и `output_topic` — топики в текущей базе данных, а `output_table` — таблица YDB. Для топиков в другой базе см. локальные и внешние топики.

Форматы при записи данных

При записи в топик `SELECT` должен возвращать одну колонку типа `String`, `Utf8`, `Json` или `Yson`. Колонка не может быть `Optional`.

Запись одной колонки:

```
CREATE STREAMING QUERY write_string_example AS
DO BEGIN

    INSERT INTO output_topic
    SELECT
        CAST(Data AS String)
    FROM
        input_topic
    WITH (
        FORMAT = raw,
        SCHEMA = (
            Data String
        )
    );

END DO
```

Чтобы записать несколько колонок, сериализуйте их в JSON:

```
CREATE STREAMING QUERY write_json_example AS
DO BEGIN

    INSERT INTO output_topic
    SELECT
        ToBytes(Unwrap(Yson::SerializeJson(Yson::From(TableRow()))))
    FROM
        input_topic
    WITH (
        FORMAT = json_each_row,
        SCHEMA = (
            Id Uint64 NOT NULL,
            Name Utf8 NOT NULL
        )
    );

END DO
```

Подробнее о функциях: TableRow, Yson::From, Yson::SerializeJson, Unwrap, ToBytes.

Если в одном потоковом запросе используется запись в несколько топиков, все выходные топики должны быть различны.

Форматы при чтении данных

Формат csv_with_names

Данный формат основан на формате CSV. Данные размещены в колонках, разделены запятыми, в первой строке находятся имена колонок.

Пример данных (в одном сообщении):

```
Year,Manufacturer,Model,Price
1997,Man_1,Model_1,3000.00
1999,Man_2,Model_2,4900.00
```

Пример запроса:

```
CREATE STREAMING QUERY csv_example AS
DO BEGIN

    UPSERT INTO output_table
    SELECT
        *
    FROM
        input_topic
    WITH (
        FORMAT = csv_with_names,
        SCHEMA = (
            Year Int32 NOT NULL,
            Manufacturer String NOT NULL,
            Model String NOT NULL,
            Price Double NOT NULL
        )
    );

END DO
```

Формат tsv_with_names

Данный формат основан на формате TSV. Данные размещены в колонках, разделены символами табуляции (код `0x9`), в первой строке находятся имена колонок.

Пример данных (в одном сообщении):

```
Year    Manufacturer    Model   Price
1997    Man_1   Model_1    3000.00
1999    Man_2   Model_2    4900.00
```

Пример запроса:

```
CREATE STREAMING QUERY tsv_example AS
DO BEGIN

    UPSERT INTO output_table
    SELECT
        *
    FROM
        input_topic
    WITH (
        FORMAT = tsv_with_names,
        SCHEMA = (
            Year Int32 NOT NULL,
            Manufacturer String NOT NULL,
            Model String NOT NULL,
            Price Double NOT NULL
        )
    );

END DO
```

Формат json_list

Данный формат основан на JSON-представлении данных. В этом формате каждое сообщение должно представлять собой JSON-массив объектов.

Пример корректных данных (в виде списка объектов JSON):

```
[
    { "Year": 1997, "Manufacturer": "Man_1", "Model": "Model_1", "Price": 3000.0 },
    { "Year": 1999, "Manufacturer": "Man_2", "Model": "Model_2", "Price": 4900.00 }
]
```

Пример некорректных данных (в каждом сообщении находится отдельный объект в формате JSON, но эти объекты не объединены в список):

```
{ "Year": 1997, "Manufacturer": "Man_1", "Model": "Model_1", "Price": 3000.0 }
{ "Year": 1999, "Manufacturer": "Man_2", "Model": "Model_2", "Price": 4900.00 }
```

Пример запроса:

```
CREATE STREAMING QUERY json_list_example AS
DO BEGIN

    UPSERT INTO output_table
    SELECT
        *
    FROM
        input_topic
    WITH (
        FORMAT = json_list,
        SCHEMA = (
            Year Int32 NOT NULL,
            Manufacturer String NOT NULL,
```

```
            Model String NOT NULL,
            Price Double NOT NULL
        )
    );

END DO
```

Формат json_each_row

Данный формат основан на JSON-представлении данных. В этом формате каждое сообщение должно представлять собой JSON-объект. Такой формат используется при передаче данных через потоковые системы, например, Apache Kafka или Топики YDB.
Несколько отдельных JSON в одном сообщении не поддерживается; JSON-список также не поддерживается.

Пример корректных данных (в одном сообщении):

```
{ "Year": 1997, "Manufacturer": "Man_1", "Model": "Model_1", "Price": 3000.0 }
```

Пример некорректных данных:

```
{ "Year": 1997, "Manufacturer": "Man_1", "Model": "Model_1", "Price": 3000.0 }
{ "Year": 1999, "Manufacturer": "Man_2", "Model": "Model_2", "Price": 4900.00 }
```

Пример запроса:

```
CREATE STREAMING QUERY json_each_row_example AS
DO BEGIN

    UPSERT INTO output_table
    SELECT
        *
    FROM
        input_topic
    WITH (
        FORMAT = json_each_row,
        SCHEMA = (
            Year Int32 NOT NULL,
            Manufacturer Utf8 NOT NULL,
            Model Utf8 NOT NULL,
            Price Double NOT NULL
        )
    );

END DO
```

Формат json_as_string

Данный формат основан на JSON-представлении данных.

В этом формате внутри каждого сообщения должен находиться:

- объект в корректном JSON-представлении в каждой отдельной строке файла;
- объекты в корректном JSON-представлении, объединенные в список.

Формат `json_as_string` не разбивает входной JSON-документ на поля, а представляет каждое сообщение в виде одного объекта JSON (или одной строки). Такой формат удобен, если список полей не одинаков во всех строках, а может изменяться.

Пример корректных данных:

```
{ "Year": 1997, "Attrs": { "Manufacturer": "Man_1", "Model": "Model_1" }, "Price": 3000.0 }
{ "Year": 1999, "Attrs": { "Manufacturer": "Man_2", "Model": "Model_2" }, "Price": 4900.00 }
```

В этом формате схема читаемых данных должна состоять только из одной колонки с одним из разрешённых типов данных, подробнее см. ниже.

Пример запроса:

```
CREATE STREAMING QUERY json_as_string_example AS
DO BEGIN

    INSERT INTO output_topic
    SELECT
        *
    FROM
        input_topic
    WITH (
        FORMAT = json_as_string,
        SCHEMA = (
            Data Json
        )
    );

END DO
```

Формат parquet

Данный формат позволяет считывать содержимое сообщений в формате Apache Parquet.

Пример запроса:

```
CREATE STREAMING QUERY parquet_example AS
DO BEGIN

    UPSERT INTO output_table
    SELECT
        *
    FROM
        input_topic
    WITH (
        FORMAT = parquet,
        SCHEMA = (
            Year Int32 NOT NULL,
            Manufacturer Utf8 NOT NULL,
            Model Utf8 NOT NULL,
            Price Double NOT NULL
        )
    );

END DO
```

Формат raw

Данный формат позволяет считывать содержимое сообщений как есть, в "сыром" виде. Считанные таким образом данные можно обработать средствами YQL. Схема по умолчанию: `SCHEMA(Data String)`.

Пример запроса:

```
CREATE STREAMING QUERY raw_example AS
DO BEGIN

    INSERT INTO output_topic
    SELECT
        *
    FROM
        input_topic
    WITH (
        FORMAT = raw,
        SCHEMA = (
            Data String
        )
    );

END DO
```

Поддерживаемые типы данных

Таблица всех поддерживаемых типов в схеме запроса:

Тип	csv_with_names	tsv_with_names	json_list	json_each_row	json_as_string	parquet	raw
`Int8`, `Int16`, `Int32`, `Int64`, `Uint8`, `Uint16`, `Uint32`, `Uint64`, `Float`, `Double`	✓	✓	✓	✓	✓	✓	
`Bool`	✓	✓	✓	✓		✓	
`DyNumber`							
`String`, `Utf8`	✓	✓	✓	✓	✓	✓	✓
`Json`					✓		✓
`JsonDocument`							
`Yson`							
`Uuid`				✓			
`Date`, `Datetime`, `Timestamp`, `TzDate`, `TzDateTime`, `TzTimestamp`							
`Interval`							

`Date32`, `Datetime64`, `Timestamp64`, `Interval64`, `TzDate32`, `TzDateTime64`, `TzTimestamp64`							
`Optional<T>`	✓	✓	✓	✓	✓	✓	✓

Примеры парсинга данных в пользовательских форматах

Парсинг JSON встроенными функциями

Пример данных:

```
{"key": 1997, "value": "42"}
```

Пример запроса:

```
CREATE STREAMING QUERY json_builtins_example AS
DO BEGIN

    INSERT INTO output_topic
    SELECT
        JSON_VALUE(Data, "$.key") AS Key,
        JSON_VALUE(Data, "$.value") AS Value
    FROM
        input_topic
    WITH (
        FORMAT = raw,
        SCHEMA = (
            Data Json
        )
    );

END DO
```

Подробнее о функциях: JSON_VALUE.

Парсинг JSON библиотекой Yson

Пример данных (формат Change Data Capture):

```
{"update":{"volume":10,"product":"bWlsaw=="},"key":[6],"ts":[1765192622420,18446744073709551615]}
```

Пример запроса:

```
CREATE STREAMING QUERY json_yson_example AS
DO BEGIN

    $input = SELECT
        Yson::ConvertTo(
            Data, Struct<
                update: Struct<volume: Uint64>,
                key: List<Uint64>,
                ts: List<Uint64>
            >
        )
    FROM
        input_topic
    WITH (
        FORMAT = json_as_string,
        SCHEMA = (
            Data Json
        )
    );

    INSERT INTO output_topic
    SELECT
        ts[0] AS Ts,
        update.volume AS Volume
    FROM
        $input
    FLATTEN COLUMNS;

END DO
```

Подробнее о функциях:

- Yson::ConvertTo
- FLATTEN COLUMNS.

Парсинг DSV (TSKV)

Пример данных:

```
name=Elena  uid=95792365232151958
name=Denis  uid=78086244452810046
name=Mikhail    uid=70609792906901286
```

Пример запроса:

```
CREATE STREAMING QUERY dsv_example AS
DO BEGIN

    $input = SELECT
        Dsv::Parse(Data, "\t") AS Data
    FROM
        input_topic
    FLATTEN LIST BY (
        String::SplitToList(Data, "\n", TRUE AS SkipEmpty) AS Data
    );

    INSERT INTO output_topic
    SELECT
        DictLookup(Data, "name") AS Name,
        DictLookup(Data, "uid") AS Uid
    FROM
        $input;

END DO
```

Подробнее о функциях:

- String::SplitToList
- DictLookup
- FLATTEN LIST BY.

Гарантии доставки данных

Гарантии доставки определяют, сколько раз каждое событие из входного топика будет обработано потоковым запросом. Понимание гарантий системы критически важно при проектировании конвейеров обработки данных.

> **Примечание**
>
> Мы постоянно работаем над развитием механизмов потоковой обработки. В будущих версиях предоставляемые гарантии будут улучшены.

Гарантии обработки данных (dataplane):

- at-least-once — для всех типов запросов каждое событие обрабатывается минимум один раз.

Аномалии при модификации запросов (control plane):

- Потеря событий при пересоздании запроса — при пересоздании запроса через DROP + CREATE часть событий, поступивших между удалением и созданием, будет пропущена.
- Частичное первое окно агрегаций — при старте запроса первое окно агрегации содержит неполные данные.

Чекпоинты и восстановление

YDB периодически сохраняет чекпоинт — снимок состояния запроса, содержащий:

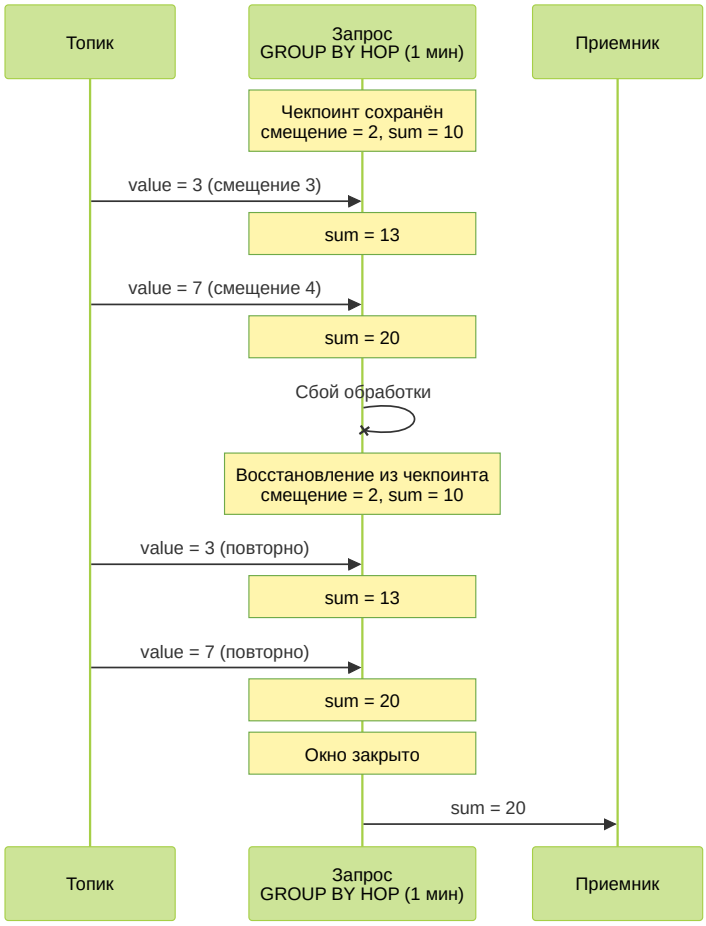
- смещения во входных топиках — позиции, до которых события были прочитаны и обработаны;
- состояния агрегаций — промежуточные результаты операций, например накопленные значения в GROUP BY HOP.

YDB хранит смещения чтения в собственных чекпоинтах, а не полагается на смещения потребителя (consumer) во внешней системе.

При восстановлении запрос откатывается к последнему чекпоинту: возобновляет чтение с сохранённых смещений и восстанавливает состояния агрегаций. События, поступившие между чекпоинтом и сбоем, будут обработаны повторно. Подробнее о механизме чекпоинтов — в разделе Чекпоинты.

Гарантии обработки данных (dataplane) — at-least-once

Если в процессе обработки потока происходит сбой (перезапуск вычислительного узла, сетевой разрыв, таймаут), YDB автоматически восстанавливает запрос из последнего чекпоинта. Гарантия at-least-once обеспечивается для всех типов потоковых запросов — каждое событие будет обработано как минимум один раз. Запрос возобновляет чтение с сохранённого смещения и отправляет результаты обработки повторно. Это относится ко всем видам запросов: к запросам без агрегации (фильтрация, обогащение, трансформация), и к запросам с оконной агрегацией.



При записи результата в таблицу через UPSERT повторная обработка не приводит к дублированию: UPSERT обновляет существующую строку по первичному ключу. Данные не теряются, дубли не накапливаются.

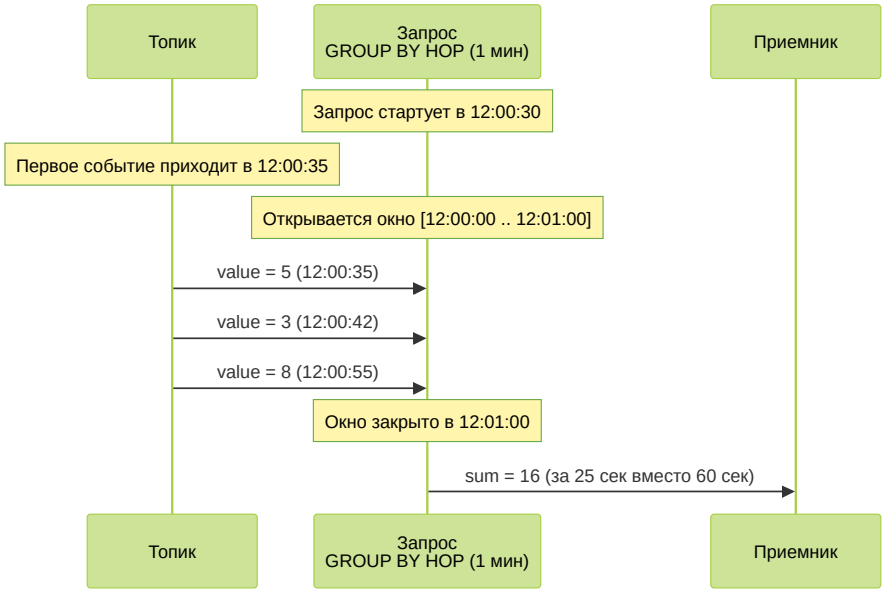
При записи результата в выходной топик повторная обработка приводит к появлению дубликатов: одни и те же события будут записаны в топик более одного раза. Потребитель выходного топика должен учитывать это и при необходимости выполнять дедупликацию самостоятельно.

Гарантии при модификации запроса (control plane)

В настоящий момент изменение текста запроса без его остановки не поддерживается. Для обновления запроса используется сочетание команд DROP + CREATE, в этом случае гарантия `at-least-once` не выполняется: часть событий может быть пропущена. Ниже описаны сценарии, в которых это происходит.

Частичные результаты первого окна при старте запроса

Временные окна (GROUP BY HOP) рассчитывают свои границы по абсолютному (wall-clock) времени. Границы окон выравниваются по кратным интервалам от начала эпохи: например, при окне в 1 минуту границы всегда проходят в 12:00:00, 12:01:00, 12:02:00 и т.д., независимо от того, когда запрос был запущен. Если запрос стартует в 12:00:30, он попадает в уже идущее окно [12:00:00 .. 12:01:00], но данные начинают поступать только с 12:00:30. В результате агрегат первого окна вычисляется по данным за 30 секунд вместо полной минуты.

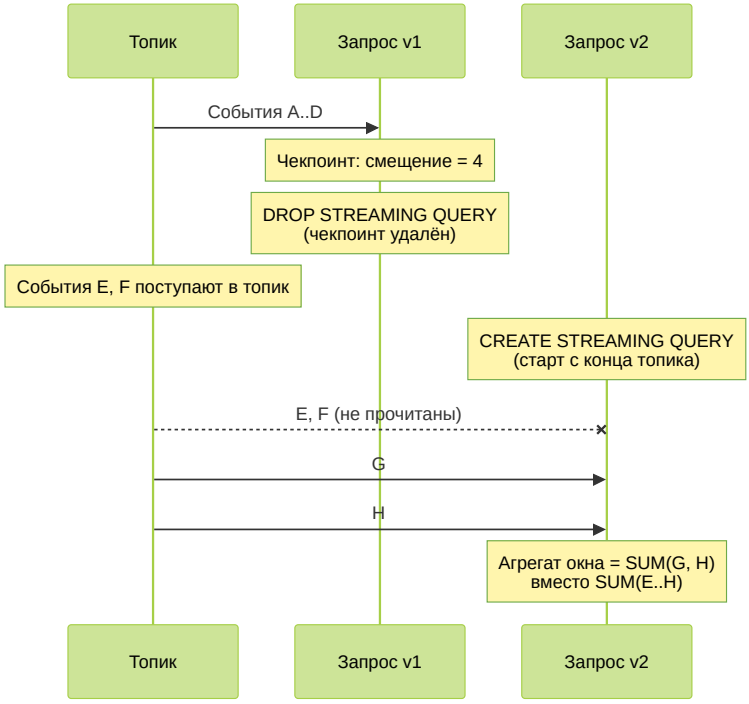


Это ожидаемое поведение при первом запуске — все последующие окна получат данные за полный интервал, который важно учитывать при пересоздании запроса.

Потеря событий при пересоздании запроса

Для изменения текста запроса используется сочетание команд DROP + CREATE. При `DROP` чекпоинт удаляется вместе с запросом, так как YDB использует внутреннее хранение смещений чтения из источника, то эти смещения удаляются вместе с запросом. Новый запрос не имеет сохранённой позиции и начинает чтение с конца топика. Все события, поступившие в топик между удалением старого запроса и стартом нового, не будут прочитаны.

Аналогичная ситуация возникает, если данные, на которые указывает смещение в чекпоинте, уже удалены из топика по TTL.



Для запросов с оконной агрегацией первые окна после пересоздания будут содержать пропуски данных и заниженные агрегаты.

См. также

- Потоковые запросы — общее описание потоковых запросов.
- Чекпоинты — механизм чекпоинтов, обеспечивающий восстановление после сбоев.
- Запись в таблицы — запись в таблицы и идемпотентность UPSERT.

Чекпоинты

Чекпоинт — это сохранённое состояние работающего потокового запроса, необходимое для восстановления после сбоев обработки. YDB периодически сохраняет чекпоинты всех запущенных потоковых запросов.

Содержимое чекпоинта

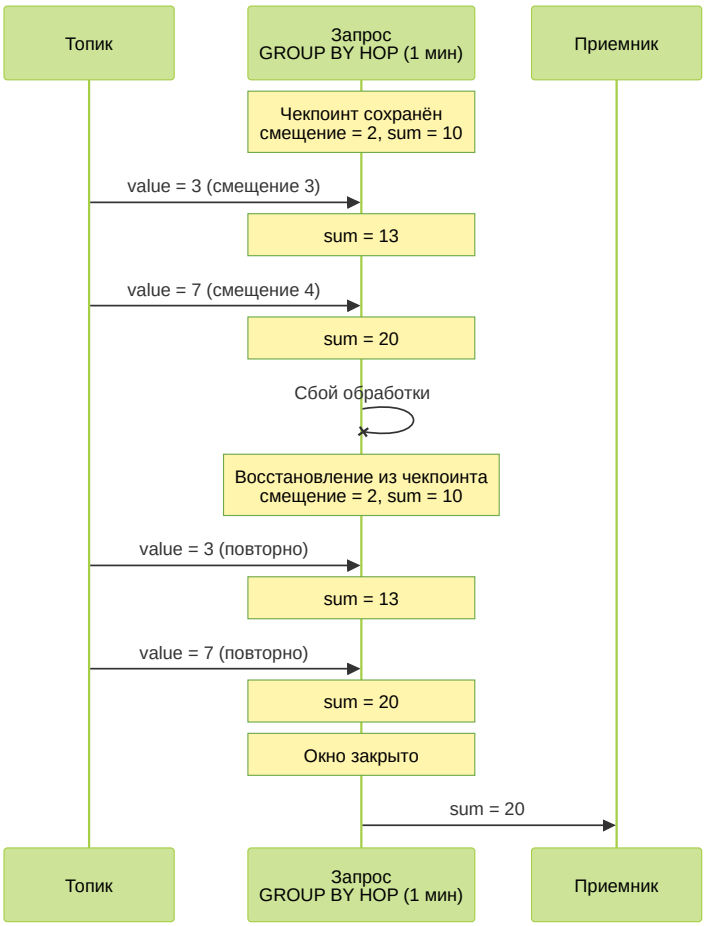
Чекпоинт содержит:

- смещения во входных топиках — позиции, до которых события были прочитаны и обработаны;
- состояния агрегаций — промежуточные результаты операций, например накопленные значения в GROUP BY HOP.

YDB хранит смещения чтения в собственных чекпоинтах, а не полагается на смещения потребителя (consumer) во внешней системе. Это означает, что при удалении запроса (DROP STREAMING QUERY) смещения удаляются вместе с чекпоинтом — внешняя система не знает, до какого места запрос дочитал топик.

Восстановление после сбоя

При сбое обработки (перезапуск вычислительного узла, сетевой разрыв, таймаут) запрос автоматически перезапускается и восстанавливает состояние из последнего чекпоинта: возобновляет чтение с сохранённых смещений и восстанавливает состояния агрегаций.

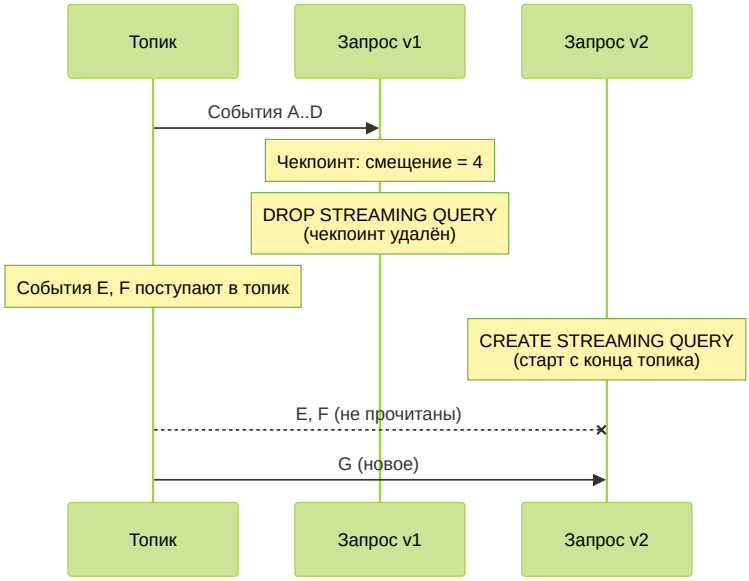


События, поступившие между последним чекпоинтом и моментом сбоя, будут обработаны повторно. Это обеспечивает гарантию at-least-once — каждое событие будет обработано минимум один раз.

Сохранение и выбор чекпоинта для восстановления происходит автоматически. Старые чекпоинты удаляются после успешного сохранения нового.

Удаление чекпоинта при пересоздании запроса

При удалении запроса (DROP STREAMING QUERY) чекпоинт удаляется вместе с ним. Поскольку смещения хранятся только в чекпоинте, новый запрос (CREATE STREAMING QUERY) не имеет сохранённой позиции и начинает чтение с конца топика. Все события, поступившие в топик между удалением старого запроса и стартом нового, не будут прочитаны.



Аналогичная ситуация возникает, если данные, на которые указывает смещение в чекпоинте, уже удалены из топика по TTL.

Подробнее о влиянии этого поведения на гарантии доставки — в разделе Потеря событий при пересоздании запроса.

Отключение чекпоинтов

Для снижения накладных расходов можно отключить сохранение чекпоинтов с помощью прагмы `ydb.DisableCheckpoints`.

> **Важно**
>
> При отключении чекпоинтов отсутствуют гарантии консистентности данных при пользовательских или внутренних перезапусках запроса. Используйте исключительно для отладки.

```
CREATE STREAMING QUERY query_without_checkpoints AS
DO BEGIN

PRAGMA ydb.DisableCheckpoints = "TRUE";

INSERT INTO
    output_topic
SELECT
    *
FROM
    input_topic;

END DO
```

См. также

- Гарантии доставки данных — гарантии доставки данных и наблюдаемые аномалии.
- Потоковые запросы — общее описание потоковых запросов.

Watermarks

Watermark — монотонно возрастающая нижняя оценка времён событий в потоке (подробнее о концепции: Watermarks). В данном разделе описана настройка watermarks в потоковых запросах YDB.

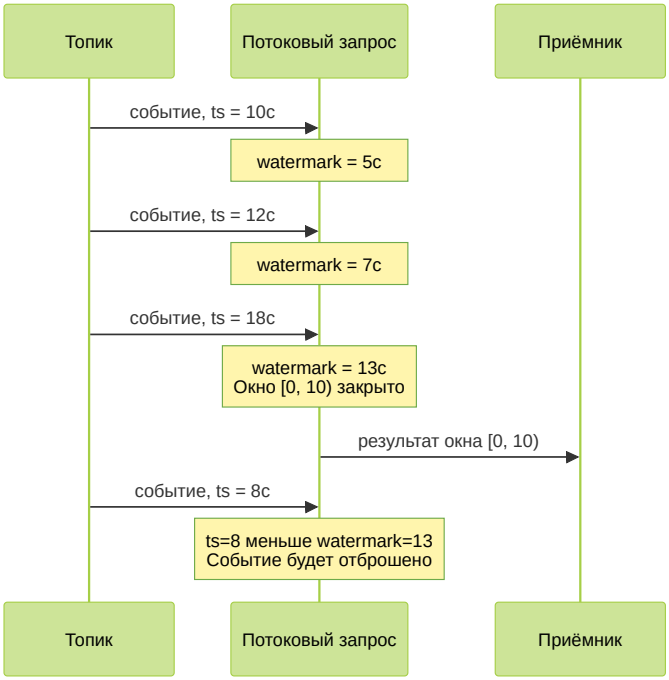
Время события

В потоковой обработке каждое событие имеет временную метку, по которой система отслеживает прогресс времени в потоке. В текущей реализации источником времени события может быть только время записи события в топик, доступное через системную колонку `__ydb_write_time`.

> **Примечание**
>
> Поддержка произвольных выражений для извлечения времени из данных события (например, поля `event.created_at`) планируется в следующих версиях.

Использование

Watermark используется операциями, зависящими от прогресса времени события в потоке. В YDB к таким операциям относится оконная агрегация HoppingWindow — она определяет скользящие временные окна, по которым группируются события. При получении watermark `HoppingWindow` закрывает все окна, которые полностью покрыты этим значением.



Вычисление watermark

Когда система получает событие, она обновляет watermark — **продвигает** его вперёд по временной оси. Watermark вычисляется как `максимальное наблюдаемое время события − delay`, где `delay` — величина отставания, заданная в выражении `WATERMARK` (например, `Interval("PT5S")` в `WATERMARK = __ydb_write_time - Interval("PT5S")`; `__ydb_write_time` — служебная колонка топика).

События в потоке могут приходить не в хронологическом порядке: событие с временем 10:00:03 может быть обработано после события с временем 10:00:05. Причины: расхождение часов в распределённой системе, сетевые задержки, неравномерная нагрузка на партиции топика.

Параметр `delay` задаёт допустимый «запас» времени для событий, поступающих с задержкой. Например, при `delay` в 5 секунд событие с временем 00:00:48 будет принято, даже если уже пришли события с временем 00:00:50: watermark ещё не дошёл до 00:00:48. Если то же событие придёт позже, когда watermark уже продвинулся за 00:00:48, оно будет признано опоздавшим и отброшено.

О компромиссе между точностью и задержкой выдачи результатов: Компромисс Точности и Задержки.

Простаивающие партиции

Если входной топик содержит несколько партиций, каждая из них продвигает watermark независимо. Общий watermark запроса не обгоняет самую медленную партицию: окна не закрываются, пока хотя бы одна партиция не достигла соответствующего момента времени.

Если одна из партиций перестаёт получать данные, её watermark перестаёт продвигаться вперёд. Такая партиция называется простаивающей (idle). Пока простаивающая партиция учитывается при вычислении общего watermark, он тоже перестаёт продвигаться, и результаты не выдаются, несмотря на поступление данных от других партиций.

Чтобы избежать этой блокировки, простаивающая партиция исключается из вычисления общего watermark по истечении настраиваемого периода ожидания (параметр `WATERMARK_IDLE_TIMEOUT`, подробнее в разделе Настройка).

Настройка

Watermarks включаются и настраиваются в секции WITH при чтении из топика.

Параметры настройки:

- `WATERMARK` — выражение для вычисления watermark. Сейчас поддерживается только время записи в топик с константной задержкой. Формат: `__ydb_write_time - Interval("<delay>")`, где `<delay>` задаётся в формате ISO

8601.

- `WATERMARK_GRANULARITY` — периодичность генерации watermarks. Чем она меньше, тем больше потребление CPU запросом и тем меньше задержка ответа. Имеет смысл только для потоковых запросов. Задаётся в формате ISO 8601. Значение по умолчанию — 1 секунда.
- `WATERMARK_IDLE_TIMEOUT` — период, после которого простаивающая партиция будет исключена из вычисления объединённого watermark. Имеет смысл только для потоковых запросов. Задаётся в формате ISO 8601. Значение по умолчанию — 5 секунд.

> **Важно**
>
> При использовании HoppingWindow первый параметр (time extractor) и источник времени в выражении WATERMARK должны совпадать. В текущей реализации оба должны использовать `__ydb_write_time`.

Пример

Ниже приведён пример потокового запроса с watermark и оконной агрегацией. Запрос читает события из топика, фильтрует их по полю `pass` и агрегирует значения `payload` в окнах по 10 секунд с шагом 5 секунд. Watermark настроен с отставанием в 5 секунд.

Входные данные

```
{"pass": 1, "payload": "a"} // время записи: 1970-01-01T00:00:40Z
{"pass": 1, "payload": "b"} // время записи: 1970-01-01T00:00:42Z
{"pass": 0, "payload": "c"} // время записи: 1970-01-01T00:00:50Z
{"pass": 1, "payload": "d"} // время записи: 1970-01-01T00:00:40Z
```

Запрос

```
CREATE STREAMING QUERY example AS
DO BEGIN
    $input = (
        SELECT
            t.*,
            __ydb_write_time AS ts
        FROM
            Input
        WITH (
            FORMAT = json_each_row,
            SCHEMA = (
                pass Int64,
                payload String
            ),
            WATERMARK = __ydb_write_time - Interval("PT5S")
        ) AS t
    );

    $output = (
        SELECT
            AGGREGATE_LIST(payload) AS result,
            CAST(HOP_END() AS String) AS ts
        FROM
            $input
        WHERE pass > 0
        GROUP BY
            HoppingWindow(ts, "PT5S", "PT10S")
    );

    INSERT INTO Output
    SELECT
        ToBytes(Unwrap(Yson2::SerializeJson(Yson::From(TableRow()))))
    FROM $output;
END DO;
```

Где:

- `CREATE STREAMING QUERY` — создаёт именованный потоковый запрос.
- `__ydb_write_time` — системная колонка, содержащая время записи события в топик.
- `FORMAT = json_each_row` — формат данных в топике, каждая строка содержит отдельный JSON-объект.
- `WATERMARK = __ydb_write_time - Interval("PT5S")` — watermark с отставанием 5 секунд. `Interval("PT5S")` задаёт интервал в формате ISO 8601.
- `AGGREGATE_LIST` — агрегатная функция, собирающая значения в список.
- `HOP_END()` — возвращает временную метку конца текущего окна.
- `HoppingWindow(ts, "PT5S", "PT10S")` — оконная функция с шагом 5 секунд и размером окна 10 секунд.

Результат

```
{"result": ["a", "b"], "ts": "1970-01-01T00:00:45.000000Z"}
```

Пояснение

1. Первое событие (`"a"` , время записи 40с) проходит фильтр (`pass > 0`) и попадает в окна `[35; 45)` и `[40; 50)` . Watermark продвигается до 35с и не закрывает ни одного окна.

2. Второе событие (`"b"` , время записи 42с) аналогично попадает в окна `[35; 45)` и `[40; 50)` . Watermark продвигается до 37с.
3. Третье событие (`"c"` , время записи 50с) отбрасывается на фильтре (`pass = 0`). Несмотря на это, событие всё равно продвигает watermark до 45с. Watermark закрывает окно `[35; 45)` — результат `["a", "b"]` выдаётся.
4. Четвёртое событие (`"d"` , время записи 40с) не продвигает watermark: он уже находится на отметке 45с. Событие проходит фильтр, но отбрасывается как опоздавшее — его время записи (40с) меньше текущего watermark (45с). Хотя окно `[40; 50)` ещё открыто, watermark уже обещал, что событий с временем < 45с больше не будет, поэтому `d` не учитывается ни в одном из своих окон.

См. также

- GROUP BY ... HoppingWindow — оконная функция, использующая watermarks.
- WITH — секция WITH для настройки параметров чтения из топика.
- Гарантии доставки данных — гарантии доставки данных.
- Чекпоинты — механизм чекпоинтов.

Тестовое приложение

В данном разделе разбирается код однотипного тестового приложения, реализованного с использованием YDB SDK на разных языках программирования:

- C++
- C#
- Go
- Java
- JavaScript
- Python
- Rust

> **Примечание**
>
> Дополнительную информацию о том, как работают эти примеры приложений, можно найти в справочной документации YDB SDK.

Тестовое приложение выполняет следующие шаги:

Инициализация соединения с базой данных

Для взаимодействия с YDB создается экземпляр драйвера, клиента и сессии:

- Драйвер YDB отвечает за взаимодействие приложения и YDB на транспортном уровне. Драйвер должен существовать на всем протяжении жизненного цикла работы с YDB и должен быть инициализирован перед созданием клиента и сессии.
- Клиент YDB работает поверх драйвера YDB и отвечает за работу с сущностями и транзакциями.
- Сессия YDB содержит информацию о выполняемых транзакциях и подготовленных запросах и содержится в контексте клиента YDB.

C++ | C# | Go | Java | JavaScript | PHP | Python | Rust

Создание строковых таблиц

Выполняется создание строковых таблиц, которые используются в дальнейших операциях тестового приложения. В результате исполнения шага в базе данных будут созданы строковые таблицы модели данных справочника сериалов:

- `series` - Сериалы
- `seasons` - Сезоны
- `episodes` - Эпизоды

После создания вызывается метод получения информации об объекте схемы данных, и выводится результат его выполнения.

C++ | C# | Go | Java | JavaScript | PHP | Python | Rust

Запись данных

Выполняется запись данных в созданные строковые таблицы с использованием команды `UPSERT` языка запросов YQL. Применяется режим передачи запроса на изменение данных с автоматическим подтверждением транзакции в одном запросе к серверу.

C++ | C# | Go | Java | JavaScript | PHP | Python | Rust

Получение выборки данных

Выполняется запрос на получение выборки данных с использованием команды `SELECT` языка запросов YQL. Демонстрируется обработка полученной выборки в приложении.

C++ | C# | Go | Java | JavaScript | PHP | Python | Rust

Параметризованные запросы

Выполняется запрос к данным с использованием параметров. Этот вариант выполнения запросов является предпочтительным, так как позволяет серверу переиспользовать план исполнения запроса при последующих его вызовах, а также спасает от уязвимостей вида SQL Injection.

C++ | C# | Go | Java | JavaScript | PHP | Python | Rust

Многошаговые транзакции

Выполняется несколько команд в рамках одной многошаговой транзакции. Между выполнением запросов допустимо выполнение работы кода клиентского приложения. Использование транзакции позволяет гарантировать, что выполненные в её контексте выборки консистентны между собой.

C++ | C# | Go | Java | JavaScript | PHP | Python

Управление транзакциями

Выполняются вызовы операторов управления транзакциями TCL - Begin и Commit.

В большинстве случаев вместо явного использования вызовов Begin и Commit лучше использовать параметры контроля транзакций в вызовах execute. Это позволит избежать лишних обращений к YDB и эффективней выполнять запросы.

C++ | C# | Go | Java | JavaScript | PHP | Python | Rust

Обработка ошибок

Подробно об обработке ошибок написано в разделе Обработка ошибок в API.

Приложение на C++

На этой странице подробно разбирается код тестового приложения, доступного в составе C++ SDK YDB.

Инициализация соединения с базой данных

Для взаимодействия с YDB создается экземпляр драйвера, клиента и сессии:

- Драйвер YDB отвечает за взаимодействие приложения и YDB на транспортном уровне. Драйвер должен существовать на всем протяжении жизненного цикла работы с YDB и должен быть инициализирован перед созданием клиента и сессии.
- Клиент YDB работает поверх драйвера YDB и отвечает за работу с сущностями и транзакциями.
- Сессия YDB содержит информацию о выполняемых транзакциях и подготовленных запросах и содержится в контексте клиента YDB.

Фрагмент кода приложения для инициализации драйвера:

```
    auto connectionParams = TConnectionsParams()
        .SetEndpoint(endpoint)
        .SetDatabase(database)
        .SetAuthToken(GetEnv("YDB_TOKEN"));

    TDriver driver(connectionParams);
```

Фрагмент кода приложения для создания клиента:

```
    TClient client(driver);
```

Создание строковых таблиц

Выполняется создание строковых таблиц, которые используются в дальнейших операциях тестового приложения. В результате исполнения шага в базе данных будут созданы строковые таблицы модели данных справочника сериалов:

- `series` - Сериалы
- `seasons` - Сезоны
- `episodes` - Эпизоды

После создания вызывается метод получения информации об объекте схемы данных, и выводится результат его выполнения.

```
    //! Creates sample tables with the ExecuteQuery method
    ThrowOnError(client.RetryQuerySync([](TSession session) {
        auto query = Sprintf(R"(
            CREATE TABLE series (
                series_id Uint64,
                title Utf8,
                series_info Utf8,
                release_date Uint64,
                PRIMARY KEY (series_id)
            );
        )");
        return session.ExecuteQuery(query, TTxControl::NoTx()).GetValueSync();
    }));
```

Запись данных

Выполняется запись данных в созданные строковые таблицы с использованием команды `UPSERT` языка запросов YQL. Применяется режим передачи запроса на изменение данных с автоматическим подтверждением транзакции в одном запросе к серверу.

Фрагмент кода, демонстрирующий выполнение запроса на запись/изменение данных:

```
//! Shows basic usage of mutating operations.
void UpsertSimple(TQueryClient client) {
    ThrowOnError(client.RetryQuerySync([](TSession session) {
        auto query = Sprintf(R"(
            UPSERT INTO episodes (series_id, season_id, episode_id, title) VALUES
                (2, 6, 1, "TBD");
        )");

        return session.ExecuteQuery(query,
            TTxControl::BeginTx(TTxSettings::SerializableRW()).CommitTx()).GetValueSync();
    }));
}
```

`PRAGMA TablePathPrefix` добавляет указанный префикс к путям таблиц внутри БД. Работает по принципу объединения путей в файловой системе — поддерживает ссылки на родительский каталог и не требует добавления слеша справа. Например:

```
PRAGMA TablePathPrefix = "/cluster/database";
SELECT * FROM episodes;
```

Подробнее о PRAGMA YQL можно прочитать в документации YQL.

Получение выборки данных

Выполняется запрос на получение выборки данных с использованием команды `SELECT` языка запросов YQL. Демонстрируется обработка полученной выборки в приложении.

Для выполнения YQL-запросов используется метод `ExecuteQuery`.
SDK позволяет в явном виде контролировать выполнение транзакций и настраивать необходимый режим выполнения транзакций с помощью класса `TTxControl`.

В приведенном ниже фрагменте кода параметры транзакции определены с помощью метод `TTxControl::BeginTx`. С помощью `TTxSettings` устанавливается режим выполнения транзакции `SerializableRW`. После завершения всех запросов транзакции она будет автоматически завершена явным указанием: `CommitTx()`. Запрос `query`, описанный с помощью синтаксиса YQL, передается методу `ExecuteQuery` для выполнения.

```
void SelectSimple(TQueryClient client) {
    TMaybe<TResultSet> resultSet;
    ThrowOnError(client.RetryQuerySync([&resultSet](TSession session) {
        auto query = Sprintf(R"(
            SELECT series_id, title, CAST(release_date AS Date) AS release_date
            FROM series
            WHERE series_id = 1;
        )");

        auto txControl =
            // Begin a new transaction with SerializableRW mode
            TTxControl::BeginTx(TTxSettings::SerializableRW())
            // Commit the transaction at the end of the query
            .CommitTx();

        auto result = session.ExecuteQuery(query, txControl).GetValueSync();
        if (!result.IsSuccess()) {
            return result;
        }
        resultSet = result.GetResultSet(0);
        return result;
    }));
```

Обработка результатов выполнения

Для обработки результатов выполнения запроса используется класс `TResultSetParser`.
Фрагмент кода, приведенный ниже, демонстрирует обработку результатов запроса с помощью объекта `parser`:

```
    TResultSetParser parser(*resultSet);
    while (parser.TryNextRow()) {
        Cout << "> SelectSimple:" << Endl << "Series"
            << ", Id: " << parser.ColumnParser("series_id").GetOptionalUint64()
            << ", Title: " << parser.ColumnParser("title").GetOptionalUtf8()
            << ", Release date: " << parser.ColumnParser("release_date").GetOptionalDate()->FormatLocalTime
("%Y-%m-%d")
            << Endl;
    }
}
```

Приведенный фрагмент кода при запуске выводит на консоль текст:

```
> SelectSimple:
series, Id: 1, title: IT Crowd, Release date: 2006-02-03
```

Параметризованные запросы

Выполняется запрос к данным с использованием параметров. Этот вариант выполнения запросов является предпочтительным, так как позволяет серверу переиспользовать план исполнения запроса при последующих его вызовах, а также спасает от уязвимостей вида SQL Injection.

Фрагмент кода демонстрирует использование параметризованных запросов и `TParamsBuilder` для формирования параметров и передачи их в `ExecuteQuery`:

```
void SelectWithParams(TQueryClient client) {
    TMaybe<TResultSet> resultSet;
    ThrowOnError(client.RetryQuerySync([&resultSet](TSession session) {
        ui64 seriesId = 2;
        ui64 seasonId = 3;
        auto query = Sprintf(R"(
            DECLARE $seriesId AS Uint64;
            DECLARE $seasonId AS Uint64;

            SELECT sa.title AS season_title, sr.title AS series_title
            FROM seasons AS sa
            INNER JOIN series AS sr
            ON sa.series_id = sr.series_id
            WHERE sa.series_id = $seriesId AND sa.season_id = $seasonId;
        )");

        auto params = TParamsBuilder()
            .AddParam("$seriesId")
                .Uint64(seriesId)
                .Build()
            .AddParam("$seasonId")
                .Uint64(seasonId)
                .Build()
            .Build();
```

```
        auto result = session.ExecuteQuery(
            query,
            TTxControl::BeginTx(TTxSettings::SerializableRW()).CommitTx(),
            params).GetValueSync();

        if (!result.IsSuccess()) {
            return result;
        }
        resultSet = result.GetResultSet(0);
        return result;
    }));

    TResultSetParser parser(*resultSet);
    if (parser.TryNextRow()) {
        Cout << "> SelectWithParams:" << Endl << "Season"
            << ", Title: " << parser.ColumnParser("season_title").GetOptionalUtf8()
            << ", Series title: " << parser.ColumnParser("series_title").GetOptionalUtf8()
            << Endl;
    }
}
```

Приведенный фрагмент кода при запуске выводит на консоль текст:

```
> SelectWithParams:
Season, title: Season 3, series title: Silicon Valley
```

Потоковые запросы

Выполняется потоковый запрос данных, результатом исполнения которого является стрим. Стрим позволяет считать неограниченное количество строк и объем данных.

> **Важно**
>
> Не используйте метод `StreamExecuteQuery` без оборачивания вызова в `RetryQuery` или `RetryQuerySync`.

```
void StreamQuerySelect(TQueryClient client) {
    Cout << "> StreamQuery:" << Endl;

    ThrowOnError(client.RetryQuerySync([](TQueryClient client) -> TStatus {
        auto query = Sprintf(R"(
            DECLARE $series AS List<UInt64>;

            SELECT series_id, season_id, title, CAST(first_aired AS Date) AS first_aired
            FROM seasons
            WHERE series_id IN $series
            ORDER BY season_id;
        )");

        auto paramsBuilder = TParamsBuilder();
        auto& listParams = paramsBuilder
                                    .AddParam("$series")
                                    .BeginList();

        for (auto x : {1, 10}) {
            listParams.AddListItem().Uint64(x);
        }

        auto parameters = listParams
                                .EndList()
                                .Build()
                                .Build();

        // Executes stream query
        auto resultStreamQuery = client.StreamExecuteQuery(query, TTxControl::NoTx(), parameters).GetValueSync();

        if (!resultStreamQuery.IsSuccess()) {
            return resultStreamQuery;
        }

        // Iterates over results
        bool eos = false;

        while (!eos) {
            auto streamPart = resultStreamQuery.ReadNext().ExtractValueSync();

            if (!streamPart.IsSuccess()) {
                eos = true;
                if (!streamPart.EOS()) {
                    return streamPart;
                }
                continue;
            }

            // It is possible for lines to be duplicated in the output stream due to an external retrier
            if (streamPart.HasResultSet()) {
                auto rs = streamPart.ExtractResultSet();
                TResultSetParser parser(rs);
```

```
                while (parser.TryNextRow()) {
                    Cout << "Season"
                            << ", SeriesId: " << parser.ColumnParser("series_id").GetOptionalUint64()
                            << ", SeasonId: " << parser.ColumnParser("season_id").GetOptionalUint64()
                            << ", Title: " << parser.ColumnParser("title").GetOptionalUtf8()
                            << ", Air date: " << parser.ColumnParser("first_aired").GetOptionalDate()->Format
LocalTime("%Y-%m-%d")
                            << Endl;
                }
            }
        }
        return TStatus(EStatus::SUCCESS, NYql::TIssues());
    }));

}
```

Приведенный фрагмент кода при запуске выводит на консоль текст (возможны дубликаты строк в потоке вывода из-за внешнего `RetryQuerySync`):

```
> StreamQuery:
Season, SeriesId: 1, SeasonId: 1, Title: Season 1, Air date: 2006-02-03
Season, SeriesId: 1, SeasonId: 2, Title: Season 2, Air date: 2007-08-24
Season, SeriesId: 1, SeasonId: 3, Title: Season 3, Air date: 2008-11-21
Season, SeriesId: 1, SeasonId: 4, Title: Season 4, Air date: 2010-06-25
```

Многошаговые транзакции

Выполняется несколько команд в рамках одной многошаговой транзакции. Между выполнением запросов допустимо выполнение работы кода клиентского приложения. Использование транзакции позволяет гарантировать, что выполненные в её контексте выборки консистентны между собой.

Первый шаг — подготовка и выполнение первого запроса:

```
//! Shows usage of transactions consisting of multiple data queries with client logic between them.
void MultiStep(TQueryClient client) {
    TMaybe<TResultSet> resultSet;
    ThrowOnError(client.RetryQuerySync([&resultSet](TSession session) {
        ui64 seriesId = 2;
        ui64 seasonId = 5;
        auto query1 = Sprintf(R"(
            DECLARE $seriesId AS Uint64;
            DECLARE $seasonId AS Uint64;

            SELECT first_aired AS from_date FROM seasons
            WHERE series_id = $seriesId AND season_id = $seasonId;
        )");

        auto params1 = TParamsBuilder()
            .AddParam("$seriesId")
                .Uint64(seriesId)
                .Build()
            .AddParam("$seasonId")
                .Uint64(seasonId)
                .Build()
            .Build();

        // Execute the first query to retrieve the required values for the client.
        // Transaction control settings do not set the CommitTx flag, allowing the transaction to remain acti
ve
        // after query execution.
        auto result = session.ExecuteQuery(
            query1,
            TTxControl::BeginTx(TTxSettings::SerializableRW()),
            params1);

        auto resultValue = result.GetValueSync();

        if (!resultValue.IsSuccess()) {
            return resultValue;
        }
```

Для продолжения работы в рамках текущей транзакции необходимо получить идентификатор транзакции:

```
        // Get the active transaction id
        auto txId = resultValue.GetTransaction()->GetId();

        // Processing the request result
        TResultSetParser parser(resultValue.GetResultSet(0));
        parser.TryNextRow();
        auto date = parser.ColumnParser("from_date").GetOptionalUint64();

        // Perform some client logic on returned values
        auto userFunc = [] (const TInstant fromDate) {
            return fromDate + TDuration::Days(15);
        };

        TInstant fromDate = TInstant::Days(*date);
        TInstant toDate = userFunc(fromDate);
```

Следующий шаг — создание следующего запроса, использующего результаты выполнения кода на стороне клиентского приложения:

```
        // Construct next query based on the results of client logic
        auto query2 = Sprintf(R"(
            DECLARE $seriesId AS Uint64;
            DECLARE $fromDate AS Uint64;
            DECLARE $toDate AS Uint64;

            SELECT season_id, episode_id, title, air_date FROM episodes
            WHERE series_id = $seriesId AND air_date >= $fromDate AND air_date <= $toDate;
        )");

        auto params2 = TParamsBuilder()
            .AddParam("$seriesId")
                .Uint64(seriesId)
                .Build()
            .AddParam("$fromDate")
                .Uint64(fromDate.Days())
                .Build()
            .AddParam("$toDate")
                .Uint64(toDate.Days())
                .Build()
            .Build();

        // Execute the second query.
        // The transaction control settings continue the active transaction (tx)
        // and commit it at the end of the second query execution.
        auto result2 = session.ExecuteQuery(
            query2,
            TTxControl::Tx(txId).CommitTx(),
            params2).GetValueSync();

        if (!result2.IsSuccess()) {
            return result2;
        }
        resultSet = result2.GetResultSet(0);
        return result2;
    })); // The end of the retried lambda

    TResultSetParser parser(*resultSet);
    Cout << "> MultiStep:" << Endl;
    while (parser.TryNextRow()) {
        auto airDate = TInstant::Days(*parser.ColumnParser("air_date").GetOptionalUint64());

        Cout << "Episode " << parser.ColumnParser("episode_id").GetOptionalUint64()
            << ", Season: " << parser.ColumnParser("season_id").GetOptionalUint64()
            << ", Title: " << parser.ColumnParser("title").GetOptionalUtf8()
            << ", Air date: " << airDate.FormatLocalTime("%a %b %d, %Y")
            << Endl;
    }
}
```

Приведенные фрагменты кода при запуске выводят на консоль текст:

```
> MultiStep:
Episode 1, Season: 5, title: Grow Fast or Die Slow, Air date: Sun Mar 25, 2018
Episode 2, Season: 5, title: Reorientation, Air date: Sun Apr 01, 2018
Episode 3, Season: 5, title: Chief Operating Officer, Air date: Sun Apr 08, 2018
```

Управление транзакциями

Выполняются вызовы операторов управления транзакциями TCL - Begin и Commit.

В большинстве случаев вместо явного использования вызовов Begin и Commit лучше использовать параметры контроля транзакций в вызовах execute. Это позволит избежать лишних обращений к YDB и эффективней выполнять запросы.

Фрагмент кода, демонстрирующий явное использование вызовов `BeginTransaction` и `tx.Commit()`:

```
void ExplicitTcl(TQueryClient client) {
    // Demonstrate the use of explicit Begin and Commit transaction control calls.
    // In most cases, it's preferable to use transaction control settings within ExecuteDataQuery calls inste
ad,
    // as this avoids additional hops to the YDB cluster and allows for more efficient query execution.
    ThrowOnError(client.RetryQuerySync([](TQueryClient client) -> TStatus {
        auto airDate = TInstant::Now();
        auto session = client.GetSession().GetValueSync().GetSession();
        auto beginResult = session.BeginTransaction(TTxSettings::SerializableRW()).GetValueSync();
        if (!beginResult.IsSuccess()) {
            return beginResult;
        }

        // Get newly created transaction id
        auto tx = beginResult.GetTransaction();

        auto query = Sprintf(R"(
            DECLARE $airDate AS Date;

            UPDATE episodes SET air_date = CAST($airDate AS Uint16) WHERE title = "TBD";
```

```
        )");

        auto params = TParamsBuilder()
            .AddParam("$airDate")
                .Date(airDate)
                .Build()
            .Build();

        // Execute query.
        // Transaction control settings continues active transaction (tx)
        auto updateResult = session.ExecuteQuery(query,
            TTxControl::Tx(tx.GetId()),
            params).GetValueSync();

        if (!updateResult.IsSuccess()) {
            return updateResult;
        }
        // Commit active transaction (tx)
        return tx.Commit().GetValueSync();
    }));
}
```

Приложение на C#

На этой странице подробно разбирается код тестового приложения, использующего C# SDK YDB.

Инициализация соединения с базой данных

Для взаимодействия с YDB создается экземпляр драйвера, клиента и сессии:

- Драйвер YDB отвечает за взаимодействие приложения и YDB на транспортном уровне. Драйвер должен существовать на всем протяжении жизненного цикла работы с YDB и должен быть инициализирован перед созданием клиента и сессии.
- Клиент YDB работает поверх драйвера YDB и отвечает за работу с сущностями и транзакциями.
- Сессия YDB содержит информацию о выполняемых транзакциях и подготовленных запросах и содержится в контексте клиента YDB.

Фрагмент кода приложения для подключения к базе данных:

```
using Ydb.Sdk.Ado;

await using var dataSource = new YdbDataSource("Host=localhost;Port=2136;Database=/local");
await using var connection = await dataSource.OpenConnectionAsync();
```

Создание строковых таблиц

Выполняется создание строковых таблиц, которые используются в дальнейших операциях тестового приложения. В результате исполнения шага в базе данных будут созданы строковые таблицы модели данных справочника сериалов:

- `series` - Сериалы
- `seasons` - Сезоны
- `episodes` - Эпизоды

После создания вызывается метод получения информации об объекте схемы данных, и выводится результат его выполнения.

Для создания таблиц используется `YdbCommand` с DDL (Data Definition Language) YQL-запросом:

```
await using var command = new YdbCommand(connection)
{
    CommandText = @"
        CREATE TABLE series (
            series_id Uint64 NOT NULL,
            title Utf8,
            series_info Utf8,
            release_date Date,
            PRIMARY KEY (series_id)
        );

        CREATE TABLE seasons (
            series_id Uint64,
            season_id Uint64,
            title Utf8,
            first_aired Date,
            last_aired Date,
            PRIMARY KEY (series_id, season_id)
        );

        CREATE TABLE episodes (
            series_id Uint64,
            season_id Uint64,
            episode_id Uint64,
            title Utf8,
            air_date Date,
            PRIMARY KEY (series_id, season_id, episode_id)
        );"
};
await command.ExecuteNonQueryAsync();
```

Запись данных

Выполняется запись данных в созданные строковые таблицы с использованием команды `UPSERT` языка запросов YQL. Применяется режим передачи запроса на изменение данных с автоматическим подтверждением транзакции в одном запросе к серверу.

Фрагмент кода, демонстрирующий выполнение запроса на запись/изменение данных:

```
await using var command = new YdbCommand(@"
    UPSERT INTO series (series_id, title, release_date) VALUES
        ($id, $title, $release_date);
    ", connection);
command.Parameters.Add(new YdbParameter("$id", YdbDbType.Uint64, 1UL));
command.Parameters.Add(new YdbParameter("$title", YdbDbType.Text, "NewTitle"));
command.Parameters.Add(new YdbParameter("$release_date", YdbDbType.Date, DateTime.UtcNow));
await command.ExecuteNonQueryAsync();
```

Получение выборки данных

Выполняется запрос на получение выборки данных с использованием команды `SELECT` языка запросов YQL. Демонстрируется обработка полученной выборки в приложении.

Для выполнения YQL-запросов с чтением данных используется метод `ExecuteReaderAsync`. Параметры запроса передаются через коллекцию `Parameters` объекта `YdbCommand`:

```
await using var command = new YdbCommand(@"
    SELECT
        series_id,
        title,
        release_date
    FROM series
    WHERE series_id = $id;
    ", connection);
command.Parameters.Add(new YdbParameter("$id", YdbDbType.Uint64, id));
await using var reader = await command.ExecuteReaderAsync();
```

Обработка результатов выполнения

Результат выполнения запроса обрабатывается через `DbDataReader`. Пример обработки результата:

```
while (await reader.ReadAsync())
{
    Console.WriteLine($"> Series, " +
        $"series_id: {reader.GetUint64(0)}, " +
        $"title: {reader.GetString(1)}, " +
        $"release_date: {reader.GetDateTime(2)}");
}
```

Для последовательного чтения строк из другого запроса:

```
await using var command = new YdbCommand(
    "SELECT title FROM seasons ORDER BY series_id, season_id;", connection);
await using var reader = await command.ExecuteReaderAsync();
while (await reader.ReadAsync())
{
    Console.WriteLine(reader.GetString(0));
}
```

Приложение на Go

На этой странице подробно разбирается код тестового приложения, использующего YDB Go SDK.

Скачивание и запуск

Приведенный ниже сценарий запуска использует git и Go. Предварительно должен быть установлен YDB Go SDK.

Создайте рабочую директорию и выполните в ней из командной строки команду клонирования репозитория с GitHub:

```
git clone https://github.com/ydb-platform/ydb-go-sdk.git
```

Далее из этой же рабочей директории выполните команду запуска тестового приложения, которая будет отличаться в зависимости от того, к какой базе данных необходимо подключиться.

Local Docker

Для соединения с развернутой локальной базой данных YDB по сценарию Docker в конфигурации по умолчанию выполните следующую команду:

```
( export YDB_ANONYMOUS_CREDENTIALS=1 && cd ydb-go-sdk/examples && \
go run ./basic/native/query -ydb="grpc://localhost:2136/local" )
```

Любая база данных

Для выполнения примера с использованием любой доступной базы данных YDB вам потребуется знать эндпоинт и путь базы данных.

Если в базе данных включена аутентификация, то вам также понадобится выбрать режим аутентификации и получить секреты - токен или логин/пароль.

Выполните команду по следующему образцу:

```
( export <auth_mode_var>="<auth_mode_value>" && cd ydb-go-sdk/examples && \
go run ./basic -ydb="<endpoint>?database=<database>" )
```

где

- `<endpoint>` - эндпоинт.
- `<database>` - путь базы данных.
- `<auth_mode_var>` - переменная окружения, определяющая режим аутентификации.
- `<auth_mode_value>` - значение параметра аутентификации для выбранного режима.

Например:

```
( export YDB_ACCESS_TOKEN_CREDENTIALS="t1.9euelZqOnJuJlc..." && cd ydb-go-sdk/examples && \
go run ./basic -ydb="grpcs://ydb.example.com:2135/somepath/somelocation" )
```

Инициализация соединения с базой данных

Для взаимодействия с YDB создается экземпляр драйвера, клиента и сессии:

- Драйвер YDB отвечает за взаимодействие приложения и YDB на транспортном уровне. Драйвер должен существовать на всем протяжении жизненного цикла работы с YDB и должен быть инициализирован перед созданием клиента и сессии.
- Клиент YDB работает поверх драйвера YDB и отвечает за работу с сущностями и транзакциями.
- Сессия YDB содержит информацию о выполняемых транзакциях и подготовленных запросах и содержится в контексте клиента YDB.

Для работы с YDB в `go` следует импортировать пакет драйвера `ydb-go-sdk`:

```
import (
 "context"
 "log"
 "path"

 "github.com/ydb-platform/ydb-go-sdk/v3"
 "github.com/ydb-platform/ydb-go-sdk/v3/query"
)
```

Для взаимодействия с YDB необходимо создать экземляр YDB-драйвера:

```
db, err := ydb.Open(context.Background(), "grpc://localhost:2136/local")
if err != nil {
  // обработка ошибки подключения
}

// При завершении работы с базой (например, выходе из программы) закройте драйвер
defer db.Close(context.Background())
```

Метод `ydb.Open` возвращает в случае успеха экземпляр драйвера, который выполняет ряд служебных функций, таких как актуализация сведений о кластере YDB и клиентская балансировка.

Метод `ydb.Open` принимает два обязательных аргумента:

- контекст;

- строка подключения к YDB.

Также доступно множество опций подключения, позволяющих переопределить значения по умолчанию.

По умолчанию используется анонимная аутентификация. А подключение к кластеру YDB с использованием токена будет иметь следующий вид:

```
db, err := ydb.Open(context.Background(), clusterEndpoint,
 ydb.WithAccessTokenCredentials(token),
)
```

Полный список провайдеров аутентификации приведён в документации ydb-go-sdk и в разделе рецептов.

В конце работы приложения для очистки ресурсов следует закрыть драйвер:

```
defer db.Close(ctx)
```

Объект `db` является входной точкой для работы с YDB, а для запросов к таблицам используется Query-сервис `db.Query()`.

Выполнение YQL-запросов осуществляется на специальных объектах — сессиях `query.Session`. Сессии хранят контекст выполнения запросов (например, транзакции) и позволяют осуществлять серверную балансировку нагрузки на узлы кластера YDB.

Клиент Query-сервиса предоставляет API для выполнения запросов к таблицам:

- Метод `db.Query().Do(ctx, op)` реализует фоновое создание сессий и повторные попытки выполнить пользовательскую операцию `op func(ctx context.Context, s query.Session) error`, в которую пользовательскому коду передаётся подготовленная сессия `query.Session`.
- Метод `db.Query().DoTx(ctx, op)` принимает пользовательскую операцию `op func(ctx context.Context, tx query.TxActor) error`, в которую пользовательскому коду передаётся подготовленная (заранее открытая) транзакция `query.TxActor`. Автоматическое выполнение `Commit` транзакции происходит, если из пользовательской операции возвращается `nil`. В случае возврата ошибки для текущей транзакции автоматически вызывается `Rollback`.
- Метод `db.Query().Exec` является вспомогательным и предназначен для выполнения единичного запроса **без результата** с автоматическими повторными попытками. Метод `Exec` возвращает `nil` в случае успешного выполнения запроса и ошибку, если операция не удалась.
- Метод `db.Query().Query` является вспомогательным и предназначен для выполнения единичного запроса с повторными попытками при необходимости. Текст запроса может содержать несколько выражений с результатами. Метод `Query` возвращает, в случае успеха, материализованный результат запроса (все данные уже прочитаны с сервера и доступны в локальной памяти) `query.Result` и позволяет итерироваться по вложенным спискам строк `query.ResultSet`. Для широких SQL-запросов, возвращающих большое количество строк, материализация результата может привести к проблеме OOM.
- Метод `db.Query().QueryResultSet` является вспомогательным и предназначен для выполнения единичного запроса с повторными попытками при необходимости. В запросе должно быть ровно одно выражение, возвращающее результат (дополнительно могут присутствовать выражения, не возвращающие результат, например `UPSERT`). Метод `QueryResultSet` возвращает, в случае успеха, материализованный список результатов `query.ResultSet`. Для широких SQL-запросов, возвращающих большое количество строк, материализация результата может привести к проблеме OOM.
- Метод `db.Query().QueryRow` является вспомогательным и предназначен для выполнения единичного запроса с повторными попытками при необходимости. Метод `QueryRow` возвращает, в случае успеха, единственную строку `query.Row`.

Создание строковых таблиц

Выполняется создание строковых таблиц, которые используются в дальнейших операциях тестового приложения. В результате исполнения шага в базе данных будут созданы строковые таблицы модели данных справочника сериалов:

- `series` - Сериалы
- `seasons` - Сезоны
- `episodes` - Эпизоды

После создания вызывается метод получения информации об объекте схемы данных, и выводится результат его выполнения.

Пример запроса без возвращаемого результата (создание таблицы):

```
import "github.com/ydb-platform/ydb-go-sdk/v3/query"

err = db.Query().Exec(ctx, `
 CREATE TABLE IF NOT EXISTS series (
  series_id Bytes,
  title Text,
  series_info Text,
  release_date Date,
  comment Text,

  PRIMARY KEY(series_id)
 )`, query.WithTxControl(query.NoTx()),
)
if err != nil {
  // обработка ошибки выполнения запроса
}
```

Получение выборки данных

Выполняется запрос на получение выборки данных с использованием команды `SELECT` языка запросов YQL. Демонстрируется обработка полученной выборки в приложении.

Для выполнения YQL-запросов и чтения результатов используются методы `query.Session.Query`, `query.Session.QueryResultSet` и `query.Session.QueryRow`.

SDK позволяет явно контролировать выполнение транзакций и настраивать необходимый режим выполнения транзакций с помощью структуры `query.TxControl`.

```
readTx := query.TxControl(
 query.BeginTx(
  query.WithSnapshotReadOnly(),
 ),
 query.CommitTx(),
)
row, err := db.Query().QueryRow(ctx,`
 DECLARE $seriesID AS Uint64;
 SELECT
   series_id,
   title,
   release_date
 FROM
   series
 WHERE
   series_id = $seriesID;`,
 query.WithParameters(
  ydb.ParamsBuilder().Param("$seriesID").Uint64(1).Build(),
 ),
 query.WithTxControl(readTx),
)
if err != nil {
  // обработка ошибки выполнения запроса
}
```

Для получения данных строки `query.Row` можно использовать следующие методы:

- `query.Row.ScanStruct` — по названиям колонок, зафиксированным в тегах структуры.
- `query.Row.ScanNamed` — по названиям колонок.
- `query.Row.Scan` — по порядку колонок.

ScanStruct

```
var info struct {
 SeriesID    string    `sql:"series_id"`
 Title       string    `sql:"title"`
 ReleaseDate time.Time `sql:"release_date"`
}
err = row.ScanStruct(&info)
if err != nil {
  // обработка ошибки выполнения запроса
}
```

ScanNamed

```
var seriesID, title string
var releaseDate time.Time
err = row.ScanNamed(query.Named("series_id", &seriesID), query.Named("title", &title), query.Named("release_d
ate", &releaseDate))
if err != nil {
  // обработка ошибки выполнения запроса
}
```

Scan

```
var seriesID, title string
var releaseDate time.Time
err = row.Scan(&seriesID, &title, &releaseDate)
if err != nil {
  // обработка ошибки выполнения запроса
}
```

Важно

Если ожидаемый объём данных от запроса велик, не следует пытаться загружать их полностью в оперативную память с помощью вспомогательных методов, таких как `query.Client.Query` и `query.Client.QueryResultSet`. Эти методы возвращают уже материализованный результат, при котором все данные запроса загружены с сервера в локальную память клиентского приложения. При большом количестве возвращаемых строк материализация результата может привести к проблеме OOM.

Для таких запросов следует использовать методы `query.TxActor.Query` или `query.TxActor.QueryResultSet` на сессии или транзакции, которые предоставляют итератор по результату без полной материализации. Сессия `query.Session` доступна только из метода `query.Client.Do`, реализующего механизмы повторных попыток при ошибках. Нужно учитывать, что чтение может быть прервано в любой момент, и в таком случае весь процесс выполнения запроса начнётся заново. То есть функция, переданная в `Do`, может вызываться более одного раза.

```
err = db.Query().Do(ctx,
 func(ctx context.Context, s query.Session) error {
  rows, err := s.QueryResultSet(ctx,`
   SELECT series_id, season_id, title, first_aired
```

```
    FROM seasons`,
   )
   if err != nil {
    return err
   }
   defer rows.Close(ctx)
   for row, err := range rows.Rows(ctx) {
    if err != nil {
     return err
    }
    var info struct {
     SeriesID    string    `sql:"series_id"`
     SeasonID    string    `sql:"season_id"`
     Title       string    `sql:"title"`
     FirstAired  time.Time `sql:"first_aired"`
    }
    err = row.ScanStruct(&info)
    if err != nil {
     return err
    }
    fmt.Printf("%+v\n", info)
   }
   return nil
  },
  query.WithIdempotent(),
 )
 if err != nil {
   // обработка ошибки выполнения запроса
 }
```

Приложение на Java

На этой странице подробно разбирается код тестового приложения, доступного в составе Java SDK Examples YDB.

Скачивание SDK Examples и запуск примера

Приведенный ниже сценарий запуска использует git и Maven.

Создайте рабочую директорию и выполните в ней из командной строки команду клонирования репозитория с GitHub:

```
git clone https://github.com/ydb-platform/ydb-java-examples
```

Далее выполните сборку SDK Examples

```
mvn package -f ./ydb-java-examples
```

Далее из этой же рабочей директории выполните команду запуска тестового приложения, которая будет отличаться в зависимости от того, к какой базе данных необходимо подключиться.

Local Docker

Для соединения с развернутой локальной базой данных YDB по сценарию Docker в конфигурации по умолчанию выполните следующую команду:

```
YDB_ANONYMOUS_CREDENTIALS=1 java -jar ydb-java-examples/query-example/target/ydb-query-example.jar grpc://loc
alhost:2136/local
```

Любая база данных

Для выполнения примера с использованием любой доступной базы данных YDB вам потребуется знать эндпоинт и путь базы данных.

Если в базе данных включена аутентификация, то вам также понадобится выбрать режим аутентификации и получить секреты - токен или логин/пароль.

Выполните команду по следующему образцу:

```
<auth_mode_var>="<auth_mode_value>" java -jar ydb-java-examples/query-example/target/ydb-query-example.jar gr
pcs://<endpoint>:<port>/<database>
```

, где

- `<endpoint>` - эндпоинт.
- `<database>` - путь базы данных.
- `<auth_mode_var>` - переменная окружения, определяющая режим аутентификации.
- `<auth_mode_value>` - значение параметра аутентификации для выбранного режима.

Например:

```
YDB_ACCESS_TOKEN_CREDENTIALS="..." java -jar ydb-java-examples/query-example/target/ydb-query-example.jar grp
cs://ydb.example.com:2135/somepath/somelocation
```

Инициализация соединения с базой данных

Для взаимодействия с YDB создается экземпляр драйвера, клиента и сессии:

- Драйвер YDB отвечает за взаимодействие приложения и YDB на транспортном уровне. Драйвер должен существовать на всем протяжении жизненного цикла работы с YDB и должен быть инициализирован перед созданием клиента и сессии.
- Клиент YDB работает поверх драйвера YDB и отвечает за работу с сущностями и транзакциями.
- Сессия YDB содержит информацию о выполняемых транзакциях и подготовленных запросах и содержится в контексте клиента YDB.

Основные параметры инициализации драйвера:

- Строка подключения с информацией об эндпоинте и базе данных. Единственный обязательный параметр.
- Провайдер аутенфикации. В случае отсутствия прямого указания будет использоваться анонимное подключение.
- Настройки пула сессий.

Фрагмент кода приложения для инициализации драйвера:

```
this.transport = GrpcTransport.forConnectionString(connectionString)
        .withAuthProvider(CloudAuthHelper.getAuthProviderFromEnviron())
        .build();
this.queryClient = QueryClient.newClient(transport).build();
```

Все операции с YDB рекомендуется выполнять с помощью класса-хелпера `SessionRetryContext`, который обеспечивает корректное повторное выполнение операции в случае частичной недоступности. Фрагмент кода для инициализации контекста ретраев:

```
this.retryCtx = SessionRetryContext.create(queryClient).build();
```

Создание строковых таблиц

Выполняется создание строковых таблиц, которые используются в дальнейших операциях тестового приложения. В результате исполнения шага в базе данных будут созданы строковые таблицы модели данных справочника сериалов:

- `series` - Сериалы
- `seasons` - Сезоны
- `episodes` - Эпизоды

После создания вызывается метод получения информации об объекте схемы данных, и выводится результат его выполнения.

Для создания таблиц используется режим транзакции `TxMode.NONE`, который позволяет выполнять схемные запросы:

```
private void createTables() {
    retryCtx.supplyResult(session -> session.createQuery(""
            + "CREATE TABLE series ("
            + "  series_id UInt64,"
            + "  title Text,"
            + "  series_info Text,"
            + "  release_date Date,"
            + "  PRIMARY KEY(series_id)"
            + ")", TxMode.NONE).execute()
    ).join().getStatus().expectSuccess("Can't create table series");

    retryCtx.supplyResult(session -> session.createQuery(""
            + "CREATE TABLE seasons ("
            + "  series_id UInt64,"
            + "  season_id UInt64,"
            + "  title Text,"
            + "  first_aired Date,"
            + "  last_aired Date,"
            + "  PRIMARY KEY(series_id, season_id)"
            + ")", TxMode.NONE).execute()
    ).join().getStatus().expectSuccess("Can't create table seasons");

    retryCtx.supplyResult(session -> session.createQuery(""
            + "CREATE TABLE episodes ("
            + "  series_id UInt64,"
            + "  season_id UInt64,"
            + "  episode_id UInt64,"
            + "  title Text,"
            + "  air_date Date,"
            + "  PRIMARY KEY(series_id, season_id, episode_id)"
            + ")", TxMode.NONE).execute()
    ).join().getStatus().expectSuccess("Can't create table episodes");
}
```

Запись данных

Выполняется запись данных в созданные строковые таблицы с использованием команды `UPSERT` языка запросов YQL. Применяется режим передачи запроса на изменение данных с автоматическим подтверждением транзакции в одном запросе к серверу.

Для выполнения YQL-запросов используется метод `QuerySession.createQuery()`. Он создаёт новый объект `QueryStream`, который позволяет выполнить запрос и подписаться на получение данных от сервера с результатами. Поскольку в запросах на запись никаких результатов не ожидается, используется метод `QueryStream.execute()` без параметров — он просто выполняет запрос и ждёт завершения стрима.
Фрагмент кода, демонстрирующий эту логику:

```
private void upsertSimple() {
    String query
            = "UPSERT INTO episodes (series_id, season_id, episode_id, title) "
            + "VALUES (2, 6, 1, \"TBD\");";

    // Executes data query with specified transaction control settings.
    retryCtx.supplyResult(session -> session.createQuery(query, TxMode.SERIALIZABLE_RW).execute())
        .join().getValue();
```

Получение выборки данных

Выполняется запрос на получение выборки данных с использованием команды `SELECT` языка запросов YQL. Демонстрируется обработка полученной выборки в приложении.

Прямое использование класса `QueryStream` для получения результатов не всегда может быть удобным — он подразумевает асинхронное получение данных от сервера через callback в методе `QueryStream.execute()`. Если ожидаемое количество строк в результате невелико, можно воспользоваться встроенным в SDK хелпером `QueryReader`, который сначала самостоятельно вычитывает все данные из стрима, а затем передаёт их пользователю в упорядоченном виде.

```
private void selectSimple() {
    String query
            = "SELECT series_id, title, release_date "
            + "FROM series WHERE series_id = 1;";

    // Executes data query with specified transaction control settings.
    QueryReader result = retryCtx.supplyResult(
            session -> QueryReader.readFrom(session.createQuery(query, TxMode.SERIALIZABLE_RW))
    ).join().getValue();
```

```
    logger.info("--[ SelectSimple ]--");

    ResultSetReader rs = result.getResultSet(0);
    while (rs.next()) {
        logger.info("read series with id {}, title {} and release_date {}",
                rs.getColumn("series_id").getUint64(),
                rs.getColumn("title").getText(),
                rs.getColumn("release_date").getDate()
        );
    }
}
```

В результате исполнения запроса формируется объект класса `QueryReader`, который может содержать несколько выборок, получаемых методом `getResultSet( <index> )`. Так как запрос содержал только одну команду `SELECT`, то результат содержит только одну выборку под индексом `0`. Приведенный фрагмент кода при запуске выводит на консоль текст:

```
12:06:36.548 INFO  App - --[ SelectSimple ]--
12:06:36.559 INFO  App - read series with id 1, title IT Crowd and release_date 2006-02-03
```

Параметризованные запросы

Выполняется запрос к данным с использованием параметров. Этот вариант выполнения запросов является предпочтительным, так как позволяет серверу переиспользовать план исполнения запроса при последующих его вызовах, а также спасает от уязвимостей вида SQL Injection.

Фрагмент кода, приведенный ниже, демонстрирует использование параметризованных запросов и класс `Params` для формирования параметров и передачи их методу `QuerySession.createQuery`.

```
private void selectWithParams(long seriesID, long seasonID) {
    String query
            = "DECLARE $seriesId AS Uint64; "
            + "DECLARE $seasonId AS Uint64; "
            + "SELECT sa.title AS season_title, sr.title AS series_title "
            + "FROM seasons AS sa INNER JOIN series AS sr ON sa.series_id = sr.series_id "
            + "WHERE sa.series_id = $seriesId AND sa.season_id = $seasonId";

    // Begin new transaction with SerializableRW mode
    TxControl txControl = TxControl.serializableRw().setCommitTx(true);

    // Type of parameter values should be exactly the same as in DECLARE statements.
    Params params = Params.of(
            "$seriesId", PrimitiveValue.newUint64(seriesID),
            "$seasonId", PrimitiveValue.newUint64(seasonID)
    );

    DataQueryResult result = retryCtx.supplyResult(session -> session.executeDataQuery(query, txControl, para
ms))
            .join().getValue();

    logger.info("--[ SelectWithParams ] -- ");

    ResultSetReader rs = result.getResultSet(0);
    while (rs.next()) {
        logger.info("read season with title {} for series {}",
                rs.getColumn("season_title").getText(),
                rs.getColumn("series_title").getText()
        );
    }
}
```

Асинхронное чтение

Если ожидаемое количество данных от запроса велико, не следует пытаться загружать их полностью в оперативную память с помощью `QueryReader`. В таком случае лучше использовать более сложную логику, предусматривающую обработку результата по мере получения данных. Важно отметить, что здесь, как и раньше, используется `SessionRetryContext` для повторных попыток выполнения запроса при ошибках. Соответственно, необходимо учитывать, что чтение может быть прервано в любой момент, и в таком случае весь процесс выполнения запроса начнётся заново.

```
private void asyncSelectRead(long seriesID, long seasonID) {
    String query
            = "DECLARE $seriesId AS Uint64; "
            + "DECLARE $seasonId AS Uint64; "
            + "SELECT ep.title AS episode_title, sa.title AS season_title, sr.title AS series_title "
            + "FROM episodes AS ep "
            + "JOIN seasons AS sa ON sa.season_id = ep.season_id "
            + "JOIN series AS sr ON sr.series_id = sa.series_id "
            + "WHERE sa.series_id = $seriesId AND sa.season_id = $seasonId;";

    // Type of parameter values should be exactly the same as in DECLARE statements.
    Params params = Params.of(
            "$seriesId", PrimitiveValue.newUint64(seriesID),
            "$seasonId", PrimitiveValue.newUint64(seasonID)
    );

    logger.info("--[ ExecuteAsyncQueryWithParams ]--");
    retryCtx.supplyResult(session -> {
        QueryStream asyncQuery = session.createQuery(query, TxMode.SNAPSHOT_RO, params);
        return asyncQuery.execute(part -> {
            ResultSetReader rs = part.getResultSetReader();
```

```
            logger.info("read {} rows of result set {}", rs.getRowCount(), part.getResultSetIndex());
            while (rs.next()) {
                logger.info("read episode {} of {} for {}",
                        rs.getColumn("episode_title").getText(),
                        rs.getColumn("season_title").getText(),
                        rs.getColumn("series_title").getText()
                );
            }
        });
    }).join().getStatus().expectSuccess("execute query problem");
}
```

Многошаговые транзакции

Выполняется несколько команд в рамках одной многошаговой транзакции. Между выполнением запросов допустимо выполнение работы кода клиентского приложения. Использование транзакции позволяет гарантировать, что выполненные в её контексте выборки консистентны между собой.

Для обеспечения корректности совместной работы транзакций и контекста ретраев каждая транзакция должна выполняться целиком внутри callback, передаваемого в `SessionRetryContext`. Возврат из callback должен происходить после полного завершения транзакции.

Шаблон кода по использованию сложных транзакций в `SessionRetryContext`

```
private void multiStepTransaction(long seriesID, long seasonID) {
    retryCtx.supplyStatus(session -> {
        QueryTransaction transaction = session.createNewTransaction(TxMode.SNAPSHOT_RO);

        //...

        return CompletableFuture.completedFuture(Status.SUCCESS);
    }).join().expectSuccess("multistep transaction problem");
}
```

Первый шаг — подготовка и выполнение первого запроса:

```
    String query1
            = "DECLARE $seriesId AS Uint64; "
            + "DECLARE $seasonId AS Uint64; "
            + "SELECT MIN(first_aired) AS from_date FROM seasons "
            + "WHERE series_id = $seriesId AND season_id = $seasonId;";

    // Execute first query to start a new transaction
    QueryReader res1 = QueryReader.readFrom(transaction.createQuery(query1, Params.of(
            "$seriesId", PrimitiveValue.newUint64(seriesID),
            "$seasonId", PrimitiveValue.newUint64(seasonID)
    ))).join().getValue();
```

Затем мы можем выполнить некоторую клиентскую обработку полученных данных:

```
    // Perform some client logic on returned values
    ResultSetReader resultSet = res1.getResultSet(0);
    if (!resultSet.next()) {
        throw new RuntimeException("not found first_aired");
    }
    LocalDate fromDate = resultSet.getColumn("from_date").getDate();
    LocalDate toDate = fromDate.plusDays(15);
```

И получить текущий `transaction id` для дальнейшей работы в рамках одной транзакции:

```
    // Get active transaction id
    logger.info("started new transaction {}", transaction.getId());
```

Следующий шаг — создание следующего запроса, использующего результаты выполнения кода на стороне клиентского приложения:

```
    // Construct next query based on the results of client logic
    String query2
            = "DECLARE $seriesId AS Uint64;"
            + "DECLARE $fromDate AS Date;"
            + "DECLARE $toDate AS Date;"
            + "SELECT season_id, episode_id, title, air_date FROM episodes "
            + "WHERE series_id = $seriesId AND air_date >= $fromDate AND air_date <= $toDate;";

    // Execute second query with commit at end.
    QueryReader res2 = QueryReader.readFrom(transaction.createQueryWithCommit(query2, Params.of(
        "$seriesId", PrimitiveValue.newUint64(seriesID),
        "$fromDate", PrimitiveValue.newDate(fromDate),
        "$toDate", PrimitiveValue.newDate(toDate)
    ))).join().getValue();

    logger.info("--[ MultiStep ]--");
    ResultSetReader rs = res2.getResultSet(0);
    while (rs.next()) {
        logger.info("read episode {} with air date {}",
                rs.getColumn("title").getText(),
                rs.getColumn("air_date").getDate()
```

```
        );
    }
```

Приведенные фрагменты кода при запуске выводят на консоль текст:

```
12:06:36.850 INFO  App - --[ MultiStep ]--
12:06:36.851 INFO  App - read episode Grow Fast or Die Slow with air date 2018-03-25
12:06:36.851 INFO  App - read episode Reorientation with air date 2018-04-01
12:06:36.851 INFO  App - read episode Chief Operating Officer with air date 2018-04-08
```

Управление транзакциями

Выполняются вызовы операторов управления транзакциями TCL - Begin и Commit.

В большинстве случаев вместо явного использования вызовов Begin и Commit лучше использовать параметры контроля транзакций в вызовах execute. Это позволит избежать лишних обращений к YDB и эффективней выполнять запросы.

Фрагмент кода, демонстрирующий явное использование вызовов `beginTransaction()` и `transaction.commit()`:

```
private void tclTransaction() {
    retryCtx.supplyResult(session -> {
        QueryTransaction transaction = session.beginTransaction(TxMode.SERIALIZABLE_RW)
            .join().getValue();

        String query
                = "DECLARE $airDate AS Date; "
                + "UPDATE episodes SET air_date = $airDate WHERE title = \"TBD\";";

        Params params = Params.of("$airDate", PrimitiveValue.newDate(Instant.now()));

        // Execute data query.
        // Transaction control settings continues active transaction (tx)
        QueryReader reader = QueryReader.readFrom(transaction.createQuery(query, params))
            .join().getValue();

        logger.info("get transaction {}", transaction.getId());

        // Commit active transaction (tx)
        return transaction.commit();
    }).join().getStatus().expectSuccess("tcl transaction problem");
}
```

Приложение на JavaScript

На этой странице представлено подробное описание кода тестового приложения, использующего YDB JavaScript SDK.

Примеры использования SDK доступны в репозитории ydb-js-sdk, а дополнительные примеры реальных сценариев использования — в репозитории ydb-js-examples.

Инициализация соединения с базой данных

Для взаимодействия с YDB создается экземпляр драйвера, клиента и сессии:

- Драйвер YDB отвечает за взаимодействие приложения и YDB на транспортном уровне. Драйвер должен существовать на всем протяжении жизненного цикла работы с YDB и должен быть инициализирован перед созданием клиента и сессии.
- Клиент YDB работает поверх драйвера YDB и отвечает за работу с сущностями и транзакциями.
- Сессия YDB содержит информацию о выполняемых транзакциях и подготовленных запросах и содержится в контексте клиента YDB.

Для работы с YDB необходимо создать экземпляр драйвера и клиента для выполнения запросов.

Установка необходимых пакетов:

```
npm install @ydbjs/core @ydbjs/query
```

Фрагмент кода приложения для инициализации драйвера:

Используя connectionString

```
import { Driver } from '@ydbjs/core'
import { query } from '@ydbjs/query'

const connectionString = 'grpc://localhost:2136/local'
const driver = new Driver(connectionString)
await driver.ready()

const sql = query(driver)
```

Используя аутентификацию

```
import { Driver } from '@ydbjs/core'
import { query } from '@ydbjs/query'
import { AnonymousCredentialsProvider } from '@ydbjs/auth'

const connectionString = 'grpc://localhost:2136/local'
const driver = new Driver(connectionString, {
  credentialsProvider: new AnonymousCredentialsProvider(),
})
await driver.ready()

const sql = query(driver)
```

Создание строковых таблиц

Выполняется создание строковых таблиц, которые используются в дальнейших операциях тестового приложения. В результате исполнения шага в базе данных будут созданы строковые таблицы модели данных справочника сериалов:

- `series` - Сериалы
- `seasons` - Сезоны
- `episodes` - Эпизоды

После создания вызывается метод получения информации об объекте схемы данных, и выводится результат его выполнения.

```
await sql`
    CREATE TABLE series (
        series_id Uint64,
        title Text,
        series_info Text,
        release_date Date,
        PRIMARY KEY (series_id)
    )
`

await sql`
    CREATE TABLE seasons (
        series_id Uint64,
        season_id Uint64,
        title Text,
        first_aired Date,
        last_aired Date,
        PRIMARY KEY (series_id, season_id)
    )
`

await sql`
    CREATE TABLE episodes (
        series_id Uint64,
        season_id Uint64,
        episode_id Uint64,
        title Text,
```

```
        air_date Date,
        PRIMARY KEY (series_id, season_id, episode_id)
    )
`
```

Запись данных

Выполняется запись данных в созданные строковые таблицы с использованием команды `UPSERT` языка запросов YQL. Применяется режим передачи запроса на изменение данных с автоматическим подтверждением транзакции в одном запросе к серверу.

Фрагмент кода, демонстрирующий выполнение запроса на запись/изменение данных:

```
await sql`
    UPSERT INTO episodes (series_id, season_id, episode_id, title)
    VALUES (2, 6, 1, "TBD")
`
```

Для вставки данных с использованием параметров:

```
import { Uint64, Text, Date as YdbDate } from '@ydbjs/value/primitive'

const data = [
  {
    series_id: new Uint64(1n),
    title: new Text('IT Crowd'),
    series_info: new Text('British sitcom'),
    release_date: new YdbDate(new Date('2006-02-03')),
  },
]

await sql`INSERT INTO series SELECT * FROM AS_TABLE(${data})`
```

Получение выборки данных

Выполняется запрос на получение выборки данных с использованием команды `SELECT` языка запросов YQL. Демонстрируется обработка полученной выборки в приложении.

Для выполнения YQL-запросов используется tagged template синтаксис. Результатом выполнения является массив наборов данных (YDB поддерживает несколько наборов результатов в одном запросе).

```
const resultSets = await sql`
    SELECT series_id, title, release_date
    FROM series
    WHERE series_id = 1
`

// resultSets[0] содержит первый набор результатов
const [firstResultSet] = resultSets
console.log(firstResultSet)
// [ { series_id: 1n, title: 'IT Crowd', release_date: 2006-02-03T00:00:00.000Z } ]
```

Для запросов с несколькими наборами результатов:

```
type Result = [[{ id: bigint }], [{ count: bigint }]]
const [rows, [{ count }]] = await sql<Result>`
    SELECT series_id as id FROM series;
    SELECT COUNT(*) as count FROM series;
`
```

Параметризованные запросы

Выполняется запрос к данным с использованием параметров. Этот вариант выполнения запросов является предпочтительным, так как позволяет серверу переиспользовать план исполнения запроса при последующих его вызовах, а также спасает от уязвимостей вида SQL Injection.

SDK автоматически привязывает параметры через интерполяцию в шаблонных строках. Поддерживаются нативные типы JavaScript, классы значений YDB, массивы и объекты.

```
const seriesId = 1n
const title = 'IT Crowd'

const resultSets = await sql`
    SELECT series_id, title, release_date
    FROM series
    WHERE series_id = ${seriesId} AND title = ${title}
`
```

Для именованных параметров и пользовательских типов:

```
import { Uint64 } from '@ydbjs/value/primitive'

const id = new Uint64(1n)
const resultSets = await sql`SELECT * FROM series WHERE series_id = $id`.parameter('id', id)
```

Запросы выполняются с потоковой передачей данных по умолчанию. Для работы с большими объёмами данных используйте стандартные запросы:

```
const resultSets = await sql`
    SELECT series_id, season_id, title, first_aired
    FROM seasons
    WHERE series_id IN (1, 2)
    ORDER BY season_id
`

for (const row of resultSets[0]) {
  console.log(`Season ${row.season_id}: ${row.title}`)
}
```

Управление транзакциями

Выполняются вызовы операторов управления транзакциями TCL - Begin и Commit.

В большинстве случаев вместо явного использования вызовов Begin и Commit лучше использовать параметры контроля транзакций в вызовах execute. Это позволит избежать лишних обращений к YDB и эффективней выполнять запросы.

Для выполнения запросов в рамках транзакции используются методы `sql.begin()` или `sql.transaction()`:

begin() — serializable read-write

```
const result = await sql.begin(async (tx) => {
  await tx`
      UPDATE episodes
      SET air_date = CurrentUtcDate()
      WHERE series_id = 2 AND season_id = 6 AND episode_id = 1
  `
  return await tx`SELECT * FROM episodes WHERE series_id = 2`
})
```

begin() с настройками изоляции

```
await sql.begin({ isolation: 'snapshotReadOnly', idempotent: true }, async (tx) => {
  return await tx`SELECT COUNT(*) FROM series`
})
```

Обработка ошибок

Подробно об обработке ошибок написано в разделе Обработка ошибок в API.

Для обработки ошибок используется класс `YDBError`:

```
import { YDBError } from '@ydbjs/error'

try {
  await sql`SELECT * FROM non_existent_table`
} catch (error) {
  if (error instanceof YDBError) {
    console.error('YDB Error:', error.message)
  }
}
```

Дополнительные возможности

Настройки запроса

```
import { StatsMode } from '@ydbjs/api/query'

await sql`SELECT * FROM series`
  .isolation('onlineReadOnly', { allowInconsistentReads: true })
  .idempotent(true)
  .timeout(5000)
  .withStats(StatsMode.FULL)
```

Динамические идентификаторы

Для динамических имён таблиц и колонок используйте метод `identifier`:

```
const tableName = 'series'
await sql`SELECT * FROM ${sql.identifier(tableName)}`
```

Закрытие драйвера

Всегда закрывайте драйвер по завершении работы:

```
driver.close()
```

Приложение на PHP

На этой странице подробно разбирается код тестового приложения, доступного в составе PHP SDK YDB.

Инициализация соединения с базой данных

Для взаимодействия с YDB создается экземпляр драйвера, клиента и сессии:

- Драйвер YDB отвечает за взаимодействие приложения и YDB на транспортном уровне. Драйвер должен существовать на всем протяжении жизненного цикла работы с YDB и должен быть инициализирован перед созданием клиента и сессии.
- Клиент YDB работает поверх драйвера YDB и отвечает за работу с сущностями и транзакциями.
- Сессия YDB содержит информацию о выполняемых транзакциях и подготовленных запросах и содержится в контексте клиента YDB.

Фрагмент кода приложения для инициализации драйвера:

```
<?php

use YdbPlatform\Ydb\Ydb;

$config = [
    // Database path
    'database'    => '/ru-central1/b1glxxxxxxxxxxxxxxx/etn0xxxxxxxxxxxxxxx',

    // Database endpoint
    'endpoint'    => 'ydb.serverless.yandexcloud.net:2135',

    // Auto discovery (dedicated server only)
    'discovery'   => false,

    // IAM config
    'iam_config'  => [
        //'root_cert_file' => './CA.pem',  Root CA file (uncomment for dedicated server)
    ],

    'credentials' => new AccessTokenAuthentication('AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA') // use from ref
erence/ydb-sdk/auth
];

$ydb = new Ydb($config);
```

Создание строковых таблиц

Выполняется создание строковых таблиц, которые используются в дальнейших операциях тестового приложения. В результате исполнения шага в базе данных будут созданы строковые таблицы модели данных справочника сериалов:

- `series` - Сериалы
- `seasons` - Сезоны
- `episodes` - Эпизоды

После создания вызывается метод получения информации об объекте схемы данных, и выводится результат его выполнения.

Для создания строковых таблиц используется метод `session->createTable()`:

```
protected function createTabels()
{
    $this->ydb->table()->retrySession(function (Session $session) {

        $session->createTable(
            'series',
            YdbTable::make()
                ->addColumn('series_id', 'UINT64')
                ->addColumn('title', 'UTF8')
                ->addColumn('series_info', 'UTF8')
                ->addColumn('release_date', 'UINT64')
                ->primaryKey('series_id')
        );

    }, true);

    $this->print('Table `series` has been created.');

    $this->ydb->table()->retrySession(function (Session $session) {

        $session->createTable(
            'seasons',
            YdbTable::make()
                ->addColumn('series_id', 'UINT64')
                ->addColumn('season_id', 'UINT64')
                ->addColumn('title', 'UTF8')
                ->addColumn('first_aired', 'UINT64')
                ->addColumn('last_aired', 'UINT64')
                ->primaryKey(['series_id', 'season_id'])
        );

    }, true);

    $this->print('Table `seasons` has been created.');

    $this->ydb->table()->retrySession(function (Session $session) {
```

```
        $session->createTable(
            'episodes',
            YdbTable::make()
                ->addColumn('series_id', 'UINT64')
                ->addColumn('season_id', 'UINT64')
                ->addColumn('episode_id', 'UINT64')
                ->addColumn('title', 'UTF8')
                ->addColumn('air_date', 'UINT64')
                ->primaryKey(['series_id', 'season_id', 'episode_id'])
        );

    }, true);

    $this->print('Table `episodes` has been created.');
}
```

Метод `session->createTable()` не позволяет создавать колоночные таблицы. Это можно сделать с помощью метода `session->query()`, который выполняет YQL-запросы.

Если вы создали строковую таблицу и хотите вывести информацию о её структуре и убедиться, что она успешно создалась, воспользуйтесь методом `session->describeTable()`:

```
protected function describeTable($table)
{
    $data = $ydb->table()->retrySession(function (Session $session) use ($table) {

        return $session->describeTable($table);

    }, true);

    $columns = [];

    foreach ($data['columns'] as $column) {
        if (isset($column['type']['optionalType']['item']['typeId'])) {
            $columns[] = [
                'Name' => $column['name'],
                'Type' => $column['type']['optionalType']['item']['typeId'],
            ];
        }
    }

    print('Table `' . $table . '`');
    print_r($columns);
    print('');
    print('Primary key: ' . implode(', ', (array)$data['primaryKey']));
}
```

Запись данных

Выполняется запись данных в созданные строковые таблицы с использованием команды `UPSERT` языка запросов YQL. Применяется режим передачи запроса на изменение данных с автоматическим подтверждением транзакции в одном запросе к серверу.

Фрагмент кода, демонстрирующий выполнение запроса на запись данных:

```
protected function upsertSimple()
{
    $ydb->table()->retryTransaction(function (Session $session) {
        $session->query('
        DECLARE $series_id AS Uint64;
        DECLARE $season_id AS Uint64;
        DECLARE $episode_id AS Uint64;
        DECLARE $title AS Utf8;
            UPSERT INTO episodes (series_id, season_id, episode_id, title)
            VALUES ($series_id, $season_id, $episode_id, $title);', [
                '$series_id' => (new Uint64Type(2))->toTypedValue(),
                '$season_id' => (new Uint64Type(6))->toTypedValue(),
                '$episode_id' => (new Uint64Type(1))->toTypedValue(),
                '$title' => (new Utf8Type('TBD'))->toTypedValue(),
            ]);
    }, true);

    print('Finished.');
}
```

Получение выборки данных

Выполняется запрос на получение выборки данных с использованием команды `SELECT` языка запросов YQL. Демонстрируется обработка полученной выборки в приложении.

Для выполнения YQL-запросов используется метод `session->query()`.

```
$result = $ydb->table()->retryTransaction(function (Session $session) {
        return $session->query('
        DECLARE $seriesID AS Uint64;
        $format = DateTime::Format("%Y-%m-%d");
        SELECT
            series_id,
```

```
            title,
            $format(DateTime::FromSeconds(CAST(release_date AS Uint32))) AS release_date
        FROM series
        WHERE series_id = $seriesID;', [
            '$seriesID' => (new Uint64Type(1))->toTypedValue()
        ]);
}, true, $params);

print_r($result->rows());
```

Параметризованные запросы

Выполняется запрос к данным с использованием параметров. Этот вариант выполнения запросов является предпочтительным, так как позволяет серверу переиспользовать план исполнения запроса при последующих его вызовах, а также спасает от уязвимостей вида SQL Injection.

Для использования параметризированных запросов можно использовать следующий код:

```
protected function selectPrepared($series_id, $season_id, $episode_id)
{
    $result = $ydb->table()->retryTransaction(function (Session $session) use ($series_id, $season_id, $episo
de_id) {

        $prepared_query = $session->prepare('
        DECLARE $series_id AS Uint64;
        DECLARE $season_id AS Uint64;
        DECLARE $episode_id AS Uint64;

        $format = DateTime::Format("%Y-%m-%d");
        SELECT
            title AS `Episode title`,
            $format(DateTime::FromSeconds(CAST(air_date AS Uint32))) AS `Air date`
        FROM episodes
        WHERE series_id = $series_id AND season_id = $season_id AND episode_id = $episode_id;');

        return $prepared_query->execute(compact(
            'series_id',
            'season_id',
            'episode_id'
        ));
    },true);

    $this->print($result->rows());
}
```

Приложение на Python

На этой странице подробно разбирается код тестового приложения, доступного в составе Python SDK YDB.

Скачивание и запуск

Приведенный ниже сценарий запуска использует git и Python3. Предварительно должен быть установлен YDB Python SDK.

Создайте рабочую директорию и выполните в ней из командной строки команды клонирования репозитория с GitHub и установки необходимых пакетов Python:

```
git clone https://github.com/ydb-platform/ydb-python-sdk.git
python3 -m pip install iso8601
```

Далее из этой же рабочей директории выполните команду запуска тестового приложения, которая будет отличаться в зависимости от того, к какой базе данных необходимо подключиться.

Local Docker

Для соединения с развернутой локальной базой данных YDB по сценарию Docker в конфигурации по умолчанию выполните следующую команду:

```
YDB_ANONYMOUS_CREDENTIALS=1 \
python3 ydb-python-sdk/examples/basic_example_v1/ -e grpc://localhost:2136 -d /local
```

Любая база данных

Для выполнения примера с использованием любой доступной базы данных YDB вам потребуется знать эндпоинт и путь базы данных.

Если в базе данных включена аутентификация, то вам также понадобится выбрать режим аутентификации и получить секреты - токен или логин/пароль.

Выполните команду по следующему образцу:

```
<auth_mode_var>="<auth_mode_value>" \
python3 ydb-python-sdk/examples/basic_example_v1/ -e <endpoint> -d <database>
```

, где

- `<endpoint>` - эндпоинт.
- `<database>` - путь базы данных.
- `<auth_mode_var>` - переменная окружения, определяющая режим аутентификации.
- `<auth_mode_value>` - значение параметра аутентификации для выбранного режима.

Например:

```
YDB_ACCESS_TOKEN_CREDENTIALS="t1.9euelZqOnJuJlc..." \
python3 ydb-python-sdk/examples/basic_example_v1/ -e grpcs://ydb.example.com:2135 -d /path/db )
```

Инициализация соединения с базой данных

Для взаимодействия с YDB создается экземпляр драйвера, клиента и сессии:

- Драйвер YDB отвечает за взаимодействие приложения и YDB на транспортном уровне. Драйвер должен существовать на всем протяжении жизненного цикла работы с YDB и должен быть инициализирован перед созданием клиента и сессии.
- Клиент YDB работает поверх драйвера YDB и отвечает за работу с сущностями и транзакциями.
- Сессия YDB содержит информацию о выполняемых транзакциях и подготовленных запросах и содержится в контексте клиента YDB.

Фрагмент кода приложения для инициализации драйвера:

Синхронный

```
def run(endpoint, database):
    driver_config = ydb.DriverConfig(
        endpoint, database, credentials=ydb.credentials_from_env_variables(),
        root_certificates=ydb.load_ydb_root_certificate(),
    )
    with ydb.Driver(driver_config) as driver:
        try:
            driver.wait(timeout=5)
        except TimeoutError:
            print("Connect failed to YDB")
            print("Last reported errors by discovery:")
            print(driver.discovery_debug_details())
            exit(1)
```

Асинхронный

```
async def run(endpoint, database):
    driver_config = ydb.DriverConfig(
        endpoint, database, credentials=ydb.credentials_from_env_variables(),
        root_certificates=ydb.load_ydb_root_certificate(),
    )
    async with ydb.aio.Driver(driver_config) as driver:
```

```
        try:
            await driver.wait(timeout=5)
        except TimeoutError:
            print("Connect failed to YDB")
            print("Last reported errors by discovery:")
            print(driver.discovery_debug_details())
            exit(1)
```

Фрагмент кода приложения для создания пула сессий:

Синхронный

```
with ydb.QuerySessionPool(driver) as pool:
    pass  # operations with pool here
```

Асинхронный

```
async with ydb.aio.QuerySessionPool(driver) as pool:
    pass  # operations with pool here
```

Выполнение запросов

YDB Python SDK поддерживает выполнение запросов с использованием синтаксиса YQL.
Существует два основных метода для выполнения запросов, которые имеют различные свойства и области применения:

- `pool.execute_with_retries`:
  - Буферизует весь результат в памяти клиента.
  - Автоматически перезапускает выполнение в случае ошибок, которые можно устранить перезапуском.
  - Не позволяет указать режим выполнения транзакции.
  - Рекомендуется для разовых запросов, которые возвращают небольшой по размеру результат.
- `tx.execute`:
  - Возвращает итератор над результатом запроса, что позволяет обработать результат, который может не поместиться в памяти клиента.
  - Перезапуски в случае ошибок должны обрабатываться вручную с помощью `pool.retry_operation_sync`.
  - Позволяет указать режим выполнения транзакции.
  - Рекомендуется для сценариев, где `pool.execute_with_retries` неэффективен.

Создание строковых таблиц

Выполняется создание строковых таблиц, которые используются в дальнейших операциях тестового приложения. В результате исполнения шага в базе данных будут созданы строковые таблицы модели данных справочника сериалов:

- `series` - Сериалы
- `seasons` - Сезоны
- `episodes` - Эпизоды

После создания вызывается метод получения информации об объекте схемы данных, и выводится результат его выполнения.

Для выполнения запросов `CREATE TABLE` стоит использовать метод `pool.execute_with_retries()`:

Синхронный

```
def create_tables(pool: ydb.QuerySessionPool):
    print("\nCreating table series...")
    pool.execute_with_retries(
        """
        CREATE TABLE `series` (
            `series_id` Int64,
            `title` Utf8,
            `series_info` Utf8,
            `release_date` Date,
            PRIMARY KEY (`series_id`)
        )
        """
    )

    print("\nCreating table seasons...")
    pool.execute_with_retries(
        """
        CREATE TABLE `seasons` (
            `series_id` Int64,
            `season_id` Int64,
            `title` Utf8,
            `first_aired` Date,
            `last_aired` Date,
            PRIMARY KEY (`series_id`, `season_id`)
        )
        """
    )

    print("\nCreating table episodes...")
    pool.execute_with_retries(
        """
        CREATE TABLE `episodes` (
            `series_id` Int64,
            `season_id` Int64,
```

```
            `episode_id` Int64,
            `title` Utf8,
            `air_date` Date,
            PRIMARY KEY (`series_id`, `season_id`, `episode_id`)
        )
        """
    )
```

Асинхронный

```
async def create_tables(pool: ydb.aio.QuerySessionPool):
    print("\nCreating table series...")
    await pool.execute_with_retries(
        """
        CREATE TABLE `series` (
            `series_id` Int64,
            `title` Utf8,
            `series_info` Utf8,
            `release_date` Date,
            PRIMARY KEY (`series_id`)
        )
        """
    )

    print("\nCreating table seasons...")
    await pool.execute_with_retries(
        """
        CREATE TABLE `seasons` (
            `series_id` Int64,
            `season_id` Int64,
            `title` Utf8,
            `first_aired` Date,
            `last_aired` Date,
            PRIMARY KEY (`series_id`, `season_id`)
        )
        """
    )

    print("\nCreating table episodes...")
    await pool.execute_with_retries(
        """
        CREATE TABLE `episodes` (
            `series_id` Int64,
            `season_id` Int64,
            `episode_id` Int64,
            `title` Utf8,
            `air_date` Date,
            PRIMARY KEY (`series_id`, `season_id`, `episode_id`)
        )
        """
    )
```

Запись данных

Выполняется запись данных в созданные строковые таблицы с использованием команды `UPSERT` языка запросов YQL. Применяется режим передачи запроса на изменение данных с автоматическим подтверждением транзакции в одном запросе к серверу.

Фрагмент кода, демонстрирующий выполнение запроса на запись/изменение данных:

Синхронный

```
def upsert_simple(pool: ydb.QuerySessionPool):
    print("\nPerforming UPSERT into episodes...")
    pool.execute_with_retries(
        """
        UPSERT INTO episodes (series_id, season_id, episode_id, title) VALUES (2, 6, 1, "TBD");
        """
    )
```

Асинхронный

```
async def upsert_simple(pool: ydb.aio.QuerySessionPool):
    print("\nPerforming UPSERT into episodes...")
    await pool.execute_with_retries(
        """
        UPSERT INTO episodes (series_id, season_id, episode_id, title) VALUES (2, 6, 1, "TBD");
        """
    )
```

Получение выборки данных

Выполняется запрос на получение выборки данных с использованием команды `SELECT` языка запросов YQL. Демонстрируется обработка полученной выборки в приложении.

Для выполнения YQL-запросов метод часто эффективен метод `pool.execute_with_retries()`.

Синхронный

```
def select_simple(pool: ydb.QuerySessionPool):
    print("\nCheck series table...")
    result_sets = pool.execute_with_retries(
        """
        SELECT
            series_id,
            title,
            release_date
        FROM series
        WHERE series_id = 1;
        """,
    )
    first_set = result_sets[0]
    for row in first_set.rows:
        print(
            "series, id: ",
            row.series_id,
            ", title: ",
            row.title,
            ", release date: ",
            row.release_date,
        )
    return first_set
```

Асинхронный

```
async def select_simple(pool: ydb.aio.QuerySessionPool):
    print("\nCheck series table...")
    result_sets = await pool.execute_with_retries(
        """
        SELECT
            series_id,
            title,
            release_date
        FROM series
        WHERE series_id = 1;
        """,
    )
    first_set = result_sets[0]
    for row in first_set.rows:
        print(
            "series, id: ",
            row.series_id,
            ", title: ",
            row.title,
            ", release date: ",
            row.release_date,
        )
    return first_set
```

В качестве результата выполнения запроса возвращается список из `result_set`, итерирование по которым выводит на консоль текст:

```
> SelectSimple:
series, Id: 1, title: IT Crowd, Release date: 2006-02-03
```

Параметризованные запросы

Для выполнения параметризованных запросов методы `pool.execute_with_retries()` и `tx.execute()` работают схожим образом - необходимо передать словарь с параметрами специального вида, где ключом служит имя параметра, а значение может быть одним из следующих:

1. Обычное значение
2. Кортеж со значением и типом
3. Специальный тип `ydb.TypedValue(value=value, value_type=value_type)`

В случае указания значения без типа, конвертация происходит по следующим правилам:

Python type	YDB type
`int`	`ydb.PrimitiveType.Int64`
`float`	`ydb.PrimitiveType.Double`
`str`	`ydb.PrimitiveType.Utf8`
`bytes`	`ydb.PrimitiveType.String`
`bool`	`ydb.PrimitiveType.Bool`
`list`	`ydb.ListType`
`dict`	`ydb.DictType`

Важно

Автоматическая конвертация списков и словарей возможна только в случае однородных структур. Тип вложенного значения будет вычисляться рекурсивно по вышеупомянутым правилам. В случае использования неоднородной структуры запросы будут падать с ошибкой типа `TypeError`.

Фрагмент кода, демонстрирующий возможность использования параметризованных запросов:

Синхронный

```
def select_with_parameters(pool: ydb.QuerySessionPool, series_id, season_id, episode_id):
    result_sets = pool.execute_with_retries(
        """
        DECLARE $seriesId AS Int64;
        DECLARE $seasonId AS Int64;
        DECLARE $episodeId AS Int64;

        SELECT
            title,
            air_date
        FROM episodes
        WHERE series_id = $seriesId AND season_id = $seasonId AND episode_id = $episodeId;
        """,
        {
            "$seriesId": series_id,  # data type could be defined implicitly
            "$seasonId": (season_id, ydb.PrimitiveType.Int64),  # could be defined via a tuple
            "$episodeId": ydb.TypedValue(episode_id, ydb.PrimitiveType.Int64),  # could be defined via a spec
ial class
        },
    )

    print("\n> select_with_parameters:")
    first_set = result_sets[0]
    for row in first_set.rows:
        print("episode title:", row.title, ", air date:", row.air_date)

    return first_set
```

Асинхронный

```
async def select_with_parameters(pool: ydb.aio.QuerySessionPool, series_id, season_id, episode_id):
    result_sets = await pool.execute_with_retries(
        """
        DECLARE $seriesId AS Int64;
        DECLARE $seasonId AS Int64;
        DECLARE $episodeId AS Int64;

        SELECT
            title,
            air_date
        FROM episodes
        WHERE series_id = $seriesId AND season_id = $seasonId AND episode_id = $episodeId;
        """,
        {
            "$seriesId": series_id,  # could be defined implicitly
            "$seasonId": (season_id, ydb.PrimitiveType.Int64),  # could be defined via a tuple
            "$episodeId": ydb.TypedValue(episode_id, ydb.PrimitiveType.Int64),  # could be defined via a spec
ial class
        },
    )

    print("\n> select_with_parameters:")
    first_set = result_sets[0]
    for row in first_set.rows:
        print("episode title:", row.title, ", air date:", row.air_date)

    return first_set
```

Фрагмент кода выше при запуске выводит на консоль текст:

```
> select_prepared_transaction:
('episode title:', u'To Build a Better Beta', ', air date:', '2016-06-05')
```

Управление транзакциями

Выполняются вызовы операторов управления транзакциями TCL - Begin и Commit.

В большинстве случаев вместо явного использования вызовов Begin и Commit лучше использовать параметры контроля транзакций в вызовах execute. Это позволит избежать лишних обращений к YDB и эффективней выполнять запросы.

Метод `session.transaction().execute()` так же может быть использован для выполнения YQL запросов. В отличие от `pool.execute_with_retries`, данный метод позволяет в явном виде контролировать выполнение транзакций и настраивать необходимый режим выполнения транзакций с помощью класса `TxControl`.

Доступные режимы транзакции:

- `ydb.QuerySerializableReadWrite()` (по умолчанию);

- `ydb.QueryOnlineReadOnly(allow_inconsistent_reads=False)`;
- `ydb.QuerySnapshotReadOnly()`;
- `ydb.QueryStaleReadOnly()`.

Подробнее про режимы транзакций описано в Режимы транзакций.

Результатом выполнения `tx.execute()` является итератор. Итератор позволяет считать неограниченное количество строк и объем данных, не загружая в память весь результат. Однако, для корректного сохранения состояния транзакции на стороне YDB итератор необходимо прочитывать до конца после каждого запроса. Если этого не сделать, пишущие запросы могут не выполниться на стороне YDB. Для удобства результат функции `tx.execute()` представлен в виде контекстного менеджера, который долистывает итератор до конца после выхода.

Синхронный

```
with tx.execute(query) as _:
    pass
```

Асинхронный

```
async with await tx.execute(query) as _:
    pass
```

Фрагмент кода, демонстрирующий явное использование вызовов `transaction().begin()` и `tx.commit()`:

Синхронный

```
def explicit_transaction_control(pool: ydb.QuerySessionPool, series_id, season_id, episode_id):
    def callee(session: ydb.QuerySession):
        query = """
        DECLARE $seriesId AS Int64;
        DECLARE $seasonId AS Int64;
        DECLARE $episodeId AS Int64;

        UPDATE episodes
        SET air_date = CurrentUtcDate()
        WHERE series_id = $seriesId AND season_id = $seasonId AND episode_id = $episodeId;
        """

        # Get newly created transaction id
        tx = session.transaction().begin()

        # Execute data query.
        # Transaction control settings continues active transaction (tx)
        with tx.execute(
            query,
            {
                "$seriesId": (series_id, ydb.PrimitiveType.Int64),
                "$seasonId": (season_id, ydb.PrimitiveType.Int64),
                "$episodeId": (episode_id, ydb.PrimitiveType.Int64),
            },
        ) as _:
            pass

        print("\n> explicit TCL call")

        # Commit active transaction(tx)
        tx.commit()

    return pool.retry_operation_sync(callee)
```

Асинхронный

```
async def explicit_transaction_control(
    pool: ydb.aio.QuerySessionPool, series_id, season_id, episode_id
):
    async def callee(session: ydb.aio.QuerySession):
        query = """
        DECLARE $seriesId AS Int64;
        DECLARE $seasonId AS Int64;
        DECLARE $episodeId AS Int64;

        UPDATE episodes
        SET air_date = CurrentUtcDate()
        WHERE series_id = $seriesId AND season_id = $seasonId AND episode_id = $episodeId;
        """

        # Get newly created transaction id
        tx = await session.transaction().begin()

        # Execute data query.
        # Transaction control settings continues active transaction (tx)
        async with await tx.execute(
            query,
            {
                "$seriesId": (series_id, ydb.PrimitiveType.Int64),
                "$seasonId": (season_id, ydb.PrimitiveType.Int64),
                "$episodeId": (episode_id, ydb.PrimitiveType.Int64),
```

```
            },
        ) as _:
            pass

        print("\n> explicit TCL call")

        # Commit active transaction(tx)
        await tx.commit()

    return await pool.retry_operation_async(callee)
```

Однако стоит помнить, что транзакция может быть открыта неявно при первом запросе. Завершиться же она может автоматически с явным указанием флага `commit_tx=True`.
Неявное управление транзакцией предпочтительно, так как требует меньше обращений к серверу. Пример неявного управления будет продемонстрирован в следующем блоке.

Итерирование по результатам запроса

Если ожидается, что результат `SELECT` запроса будет иметь потенциально большое количество найденных строк, рекомендуется использовать метод `tx.execute` вместо `pool.execute_with_retries` для избежания чрезмерного потребления памяти на стороне клиента.

Пример `SELECT` с неограниченным количеством данных и неявным контролем транзакции:

Синхронный

```
def huge_select(pool: ydb.QuerySessionPool):
    def callee(session: ydb.QuerySession):
        query = """SELECT * from episodes;"""

        with session.transaction(ydb.QuerySnapshotReadOnly()).execute(
            query,
            commit_tx=True,
        ) as result_sets:
            print("\n> Huge SELECT call")
            for result_set in result_sets:
                for row in result_set.rows:
                    print("episode title:", row.title, ", air date:", row.air_date)

    return pool.retry_operation_sync(callee)
```

Асинхронный

```
async def huge_select(pool: ydb.aio.QuerySessionPool):
    async def callee(session: ydb.aio.QuerySession):
        query = """SELECT * from episodes;"""

        async with await session.transaction(ydb.QuerySnapshotReadOnly()).execute(
            query,
            commit_tx=True,
        ) as result_sets:
            print("\n> Huge SELECT call")
            async for result_set in result_sets:
                for row in result_set.rows:
                    print("episode title:", row.title, ", air date:", row.air_date)

    return await pool.retry_operation_async(callee)
```

Пример приложения в Rust

На этой странице подробно разбирается код тестового приложения, использующего YDB Rust SDK.

Получение и запуск

Нужны Git и Rust 1.85+. Об установке SDK — см. Установка SDK.

Клонируйте репозиторий:

```
git clone https://github.com/ydb-platform/ydb-rs-sdk.git
```

Запуск примера:

Локальный Docker

Для подключения к локальной базе YDB в сценарии Docker выполните:

```
export YDB_CONNECTION_STRING=grpc://localhost:2136/local
cd ydb-rs-sdk/ydb
cargo run --example basic
```

Любая база

Для запуска примера на любой доступной базе YDB укажите endpoint и путь к базе.

Если включена аутентификация, выберите режим аутентификации и задайте переменные окружения.

```
export <auth_mode_var>="<auth_mode_value>"
export YDB_CONNECTION_STRING="<endpoint>/<database>"
cd ydb-rs-sdk/ydb
cargo run --example basic
```

где

- `<endpoint>` — endpoint.
- `<database>` — путь к базе.
- `<auth_mode_var>` — переменная окружения для режима аутентификации.
- `<auth_mode_value>` — значение учётных данных.

Например:

```
export YDB_ACCESS_TOKEN_CREDENTIALS="t1.9euelZqOnJuJlc..."
export YDB_CONNECTION_STRING="grpcs://ydb.example.com:2135/somepath/somelocation"
cd ydb-rs-sdk/ydb
cargo run --example basic
```

Инициализация соединения с базой данных

Для взаимодействия с YDB создается экземпляр драйвера, клиента и сессии:

- Драйвер YDB отвечает за взаимодействие приложения и YDB на транспортном уровне. Драйвер должен существовать на всем протяжении жизненного цикла работы с YDB и должен быть инициализирован перед созданием клиента и сессии.
- Клиент YDB работает поверх драйвера YDB и отвечает за работу с сущностями и транзакциями.
- Сессия YDB содержит информацию о выполняемых транзакциях и подготовленных запросах и содержится в контексте клиента YDB.

Импорт и инициализация клиента:

```rust
use ydb::{ClientBuilder, YdbResult};

#[tokio::main]
async fn main() -> YdbResult<()> {
    let connection_string = std::env::var("YDB_CONNECTION_STRING")
        .unwrap_or_else(|_| "grpc://localhost:2136/local".to_string());

    let client = ClientBuilder::new_from_connection_string(connection_string)?.client()?;
    client.wait().await?;

    let mut qc = client.query_client();
    // ...
    Ok(())
}
```

`ClientBuilder::new_from_connection_string` принимает строку подключения (`grpc://host:port/database`). `client.wait()` ждёт discovery эндпоинтов. `query_client()` — вход в API Query Service.

Для локального Docker по умолчанию используется анонимная аутентификация. Для токена — `ClientBuilder::with_credentials`, см. рецепты аутентификации.

Клиент Query Service

Одноразовые запросы — awaitable builders на `QueryClient`:

- `qc.exec(yql)` — без результирующего набора (DDL, DML).
- `qc.query_row(yql)` — одна строка.

- `qc.query(yql).await?` — потоковый `QueryStream`.

Повторы при ошибках: `.idempotent(true)` на builder каждого идемпотентного запроса (чтения, DDL).

Создание строковых таблиц

Выполняется создание строковых таблиц, которые используются в дальнейших операциях тестового приложения. В результате исполнения шага в базе данных будут созданы строковые таблицы модели данных справочника сериалов:

- `series` - Сериалы
- `seasons` - Сезоны
- `episodes` - Эпизоды

После создания вызывается метод получения информации об объекте схемы данных, и выводится результат его выполнения.

Создание таблицы (implicit session, DDL без явного tx_control):

```
qc.exec(format!(
    "CREATE TABLE IF NOT EXISTS `{}` (
        series_id Bytes,
        title Text,
        series_info Text,
        release_date Date,
        comment Text,
        PRIMARY KEY(series_id)
    )",
    "native/query/series"
))
.await?;
```

Запись данных

Выполняется запись данных в созданные строковые таблицы с использованием команды `UPSERT` языка запросов YQL. Применяется режим передачи запроса на изменение данных с автоматическим подтверждением транзакции в одном запросе к серверу.

Пакетная загрузка через `AS_TABLE`:

```
use ydb::{Value, ydb_struct};

let rows: Vec<Value> = /* ... */;
let list = Value::list_from(example_row, rows)?;
qc.exec("UPSERT INTO ... FROM AS_TABLE($seriesData);")
    .param("$seriesData", list)
    .await?;
```

Получение выборки данных

Выполняется запрос на получение выборки данных с использованием команды `SELECT` языка запросов YQL. Демонстрируется обработка полученной выборки в приложении.

Чтение материализованного результата в режиме изоляции snapshot read-only:

```
use ydb::TxMode;

let mut result = qc
    .query("SELECT series_id, title, release_date FROM `native/query/series`")
    .with_tx_mode(TxMode::SnapshotReadOnly)
    .idempotent(true)
    .await?;

while let Some(result_set) = result.next_result_set().await? {
    for mut row in result_set {
        // извлечение колонок из row
    }
}
result.close().await?;
```

Параметризованные запросы

Выполняется запрос к данным с использованием параметров. Этот вариант выполнения запросов является предпочтительным, так как позволяет серверу переиспользовать план исполнения запроса при последующих его вызовах, а также спасает от уязвимостей вида SQL Injection.

Параметры задаются функцией `.param(name, value)` или макросом `ydb_params!`:

```
use ydb::ydb_params;

// по одному параметру
qc.exec("UPSERT INTO `native/query/series` (series_id, title) VALUES ($id, $title)")
    .param("$id", b"series-1".to_vec())
    .param("$title", "Example title")
    .await?;

// несколько параметров через макрос
qc.exec("UPSERT INTO `native/query/series` (series_id, title) VALUES ($id, $title)")
    .params(ydb_params!(
        "$id" => b"series-2".to_vec(),
```

```
        "$title" => "Another title",
    ))
    .await?;
```

Управление транзакциями

Выполняются вызовы операторов управления транзакциями TCL - Begin и Commit.

В большинстве случаев вместо явного использования вызовов Begin и Commit лучше использовать параметры контроля транзакций в вызовах execute. Это позволит избежать лишних обращений к YDB и эффективней выполнять запросы.

В Rust SDK явное управление транзакциями (через `Begin` и `Commit`) недоступно для клиентского кода. Вместо этого используйте `retry_tx` с `.isolation(TxMode::…)` для интерактивных транзакций. Для одиночного SQL-запроса режим изоляции задаётся через `.with_tx_mode(...)`.

Один запрос в режиме snapshot read-only:

```rust
use ydb::TxMode;

let mut row = qc
    .query_row("SELECT title FROM `native/query/series` WHERE series_id = $id")
    .param("$id", b"series-1".to_vec())
    .with_tx_mode(TxMode::SnapshotReadOnly)
    .idempotent(true)
    .await?;
```

Интерактивная транзакция с несколькими операциями:

```rust
use ydb::TxMode;

let title: String = qc
    .retry_tx(async |tx| {
        let mut row = tx
            .query_row("SELECT title FROM `native/query/series` WHERE series_id = $id")
            .param("$id", b"series-1".to_vec())
            .await?;
        Ok(row.remove_field_by_name("title")?.try_into()?)
    })
    .isolation(TxMode::SerializableReadWrite)
    .idempotent(true)
    .await?;
```

По умолчанию для запросов на query-клиенте используется режим ImplicitTx, реальный режим изоляции определяет серверная сторона YDB.

Генерация кода из YQL (sqlc-ydb)

sqlc-ydb генерирует типизированный код для YDB по схеме и запросам на YQL. Вы пишете SQL, а утилита создаёт методы, передающие параметры и читающие результаты через выбранный SDK или драйвер. Это самостоятельный проект со знакомым подходом sqlc.

Ниже показан путь от схемы и двух запросов до сгенерированного кода и его вызовов в приложении.

Установка

На Linux и macOS выполните команды ниже. Генерация не требует SDK или подключения к базе. Установщик выбирает последнюю стабильную версию, проверяет SHA256 и помещает бинарник в `~/.local/bin`. Если нужна настройка `PATH`, выполните напечатанную установщиком инструкцию перед `sqlc-ydb version`.

```
curl -fsSL https://raw.githubusercontent.com/ydb-platform/sqlc-ydb/main/install.sh | bash
sqlc-ydb version
```

Для Windows скачайте подходящий архив со страницы последнего стабильного релиза. Другие способы установки и выбор конкретной версии описаны в инструкции установки.

Схема и запросы

Создайте пустой каталог проекта и сохраните общую для всех языков схему в `schema.sql`:

```
CREATE TABLE authors (
    id Uint64 NOT NULL,
    name Utf8 NOT NULL,
    PRIMARY KEY (id)
);
```

В `queries.sql` сохраните два запроса. `UpsertAuthor` добавляет автора или обновляет имя существующего, а `GetAuthor` читает запись по ключу. Имена параметров начинаются с `$`; их типы выводятся из схемы, поэтому `DECLARE` здесь не нужен.

```
-- name: UpsertAuthor :exec
UPSERT INTO authors (id, name) VALUES ($author_id, $author_name);

-- name: GetAuthor :one
SELECT id, name FROM authors WHERE id = $author_id;
```

Комментарий `-- name:` задаёт имя метода. `:exec` означает выполнение без строк результата, `:one` — получение одной строки. Все примеры ниже используют именно эти файлы.

1. Настройка генератора в sqlc.yaml

Сохраните конфигурацию из выбранной вкладки в `sqlc.yaml`. Пути входных файлов и каталога `out` вычисляются относительно этого файла. Список языков и фреймворков, доступных в установленной версии, показывает команда:

```
sqlc-ydb init --help
```

Выберите язык и runtime из вывода этой команды, затем запросите справку по параметрам генератора:

```
sqlc-ydb init --language <language> --runtime <runtime> --help
```

Замените `<language>` и `<runtime>` значениями из справки. Команда покажет назначение параметров, их типы, допустимые значения и значения по умолчанию. Если не указывать `--runtime`, справка будет использовать runtime по умолчанию для выбранного языка. Вызов с `--help` не создаёт файлов.

Вкладки ниже показывают примеры настройки. Актуальный список поддерживаемых языков, runtime и параметров предоставляет справка установленной версии sqlc-ydb.

C++

YDB SDK

```
version: "2"
sql:
  - engine: ydb
    schema: schema.sql
    queries: queries.sql
    gen:
      cpp:
        out: "cpp/ydb"
        namespace: "authors"
        runtime: "ydb"
```

userver

```
version: "2"
sql:
  - engine: ydb
    schema: schema.sql
    queries: queries.sql
    gen:
      cpp:
        out: "cpp/userver"
```

```
        namespace: "authors"
        runtime: "userver"
```

Go

YDB SDK

```
version: "2"
sql:
  - engine: ydb
    schema: schema.sql
    queries: queries.sql
    gen:
      go:
        out: "go/native"
        package: "authors"
        sql_package: "ydb"
```

database/sql

```
version: "2"
sql:
  - engine: ydb
    schema: schema.sql
    queries: queries.sql
    gen:
      go:
        out: "go/sql"
        package: "authors"
        sql_package: "database/sql"
```

Java

YDB SDK

```
version: "2"
sql:
  - engine: ydb
    schema: schema.sql
    queries: queries.sql
    gen:
      java:
        out: "java/ydb"
        package: "authors"
        runtime: "ydb"
```

JDBC

```
version: "2"
sql:
  - engine: ydb
    schema: schema.sql
    queries: queries.sql
    gen:
      java:
        out: "java/jdbc"
        package: "authors"
        runtime: "jdbc"
```

jOOQ

```
version: "2"
sql:
  - engine: ydb
    schema: schema.sql
    queries: queries.sql
    gen:
      java:
        out: "java/jooq"
        package: "authors"
        runtime: "jooq"
```

Python

YDB SDK

```
version: "2"
sql:
  - engine: ydb
    schema: schema.sql
    queries: queries.sql
    gen:
      python:
```

```
        out: "python/ydb"
        runtime: "ydb"
```

DB-API

```
version: "2"
sql:
  - engine: ydb
    schema: schema.sql
    queries: queries.sql
    gen:
      python:
        out: "python/dbapi"
        runtime: "dbapi"
```

SQLAlchemy

```
version: "2"
sql:
  - engine: ydb
    schema: schema.sql
    queries: queries.sql
    gen:
      python:
        out: "python/sqlalchemy"
        runtime: "sqlalchemy"
```

C#

ADO.NET

```
version: "2"
sql:
  - engine: ydb
    schema: schema.sql
    queries: queries.sql
    gen:
      csharp:
        out: "csharp/adonet"
        namespace: "Authors"
        runtime: "adonet"
```

Dapper

```
version: "2"
sql:
  - engine: ydb
    schema: schema.sql
    queries: queries.sql
    gen:
      csharp:
        out: "csharp/dapper"
        namespace: "Authors"
        runtime: "dapper"
```

TypeScript

YDB SDK

```
version: "2"
sql:
  - engine: ydb
    schema: schema.sql
    queries: queries.sql
    gen:
      typescript:
        out: "typescript"
        runtime: "ydb"
```

Rust

YDB SDK

```
version: "2"
sql:
  - engine: ydb
    schema: schema.sql
    queries: queries.sql
    gen:
      rust:
        out: "rust"
        runtime: "ydb"
```

PHP

YDB SDK

```
version: "2"
sql:
  - engine: ydb
    schema: schema.sql
    queries: queries.sql
    gen:
      php:
        out: "php"
        namespace: "Authors"
        runtime: "ydb"
```

Kotlin

YDB SDK

```
version: "2"
sql:
  - engine: ydb
    schema: schema.sql
    queries: queries.sql
    gen:
      kotlin:
        out: "kotlin/ydb"
        package: "authors"
        runtime: "ydb"
```

JDBC

```
version: "2"
sql:
  - engine: ydb
    schema: schema.sql
    queries: queries.sql
    gen:
      kotlin:
        out: "kotlin/jdbc"
        package: "authors"
        runtime: "jdbc"
```

Exposed

```
version: "2"
sql:
  - engine: ydb
    schema: schema.sql
    queries: queries.sql
    gen:
      kotlin:
        out: "kotlin/exposed"
        package: "authors"
        runtime: "exposed"
```

2. Генерация кода

В каталоге с `sqlc.yaml` выполните команды. `compile` проверяет входные данные, `generate` записывает код, а `diff` проверяет, что сохранённый результат совпадает с генерацией. Последнюю команду можно включить в CI: при различиях она возвращает код 1.

```
sqlc-ydb compile
sqlc-ydb generate
sqlc-ydb diff
```

Ниже — фрагменты реального результата генерации из общего запроса: тип строки или метод чтения. Импорты и окружающие объявления классов опущены. Используйте полные файлы из каталога `out`; не редактируйте их вручную.

C++

YDB SDK

```
struct GetAuthorRow final {
    std::uint64_t id;
    std::string name;
};

    void UpsertAuthor(std::uint64_t author_id, const std::string& author_name) const;
    std::optional<GetAuthorRow> GetAuthor(std::uint64_t author_id) const;
```

userver

```
struct GetAuthorRow final {
    std::uint64_t id;
    ::userver::ydb::Utf8 name;
};

    void UpsertAuthor(std::uint64_t author_id, const ::userver::ydb::Utf8& author_name) const;
    std::optional<GetAuthorRow> GetAuthor(std::uint64_t author_id) const;
```

Go

YDB SDK

```
// -- name: GetAuthor :one
func (q *Queries) GetAuthor(ctx context.Context, arg uint64, opts ...query.ExecuteOption) (GetAuthorRow, erro
r) {
    parameters := ydb.ParamsBuilder()
    parameters = parameters.Param("$author_id").Uint64(arg)

    callOptions := append([]query.ExecuteOption(nil), opts...)
    callOptions = append(callOptions, query.WithParameters(parameters.Build()))

    result, err := q.db.QueryRow(ctx, ""+
        "SELECT id, name FROM authors WHERE id = $author_id;",
        callOptions...,
    )
    if err != nil {
        return GetAuthorRow{}, err
    }

    var row GetAuthorRow
    if err := result.ScanNamed(
        query.Named("id", &row.ID),
        query.Named("name", &row.Name),
    ); err != nil {
        return GetAuthorRow{}, err
    }

    return row, nil
}
```

database/sql

```
// -- name: GetAuthor :one
func (q *Queries) GetAuthor(ctx context.Context, arg uint64) (GetAuthorRow, error) {
    var row GetAuthorRow
    err := q.db.QueryRowContext(ctx, ""+
        "SELECT id, name FROM authors WHERE id = $author_id;",
        sql.Named("author_id", arg),
    ).Scan(
        &row.ID,
        &row.Name,
    )

    return row, err
}
```

Java

YDB SDK

```
public record GetAuthorRow(long id, String name) {}

    // -- name: GetAuthor :one
    public java.util.Optional<GetAuthorRow> getAuthor(long authorId) {
        var _params = Params.create();
        _params.put("$author_id", PrimitiveValue.newUint64(authorId));
        var _query = QueryReader.readFrom(
                client.createQuery("""
                    SELECT id, name FROM authors WHERE id = $author_id;\
                    """, _params)).join().getValue();
        if (_query.getResultSetCount() != 1) throw new IllegalStateException("Expected one result set");
        var _rows = _query.getResultSet(0);
        if (!_rows.next()) return java.util.Optional.empty();
        long _value0 = _rows.getColumn(0).getUint64();
        String _value1 = _rows.getColumn(1).getText();
        return java.util.Optional.of(new GetAuthorRow(_value0, _value1));
    }
```

JDBC

```
public record GetAuthorRow(long id, String name) {}

    // -- name: GetAuthor :one
    public java.util.Optional<GetAuthorRow> getAuthor(long authorId) throws java.sql.SQLException {
        try (var _prepared = client.prepareStatement("""
```

```
            SELECT id, name FROM authors WHERE id = ?;\
            """)) {
            _prepared.setObject(1, PrimitiveValue.newUint64(authorId));
            try (var _rows = _prepared.executeQuery()) {
                if (!_rows.next()) return java.util.Optional.empty();
                long _value0 = _rows.getLong(1);
                String _value1 = _rows.getString(2);
                return java.util.Optional.of(new GetAuthorRow(_value0, _value1));
            }
        }
    }
```

jOOQ

```
public record GetAuthorRow(ULong id, String name) {}

    // -- name: GetAuthor :one
    public Optional<GetAuthorRow> getAuthor(ULong authorId) {
        return dsl.select(AUTHORS.ID, AUTHORS.NAME)
                .from(AUTHORS)
                .where(AUTHORS.ID.eq(val(authorId, YdbTypes.UINT64)))
                .fetchOptional(mapping(GetAuthorRow::new));
    }
```

Python

YDB SDK

```
@dataclass
class Authors:
    id: int
    name: str
```

DB-API

```
@dataclass
class Authors:
    id: int
    name: str
```

SQLAlchemy

```
@dataclass
class Authors:
    id: int
    name: str
```

C#

ADO.NET

```
public sealed record GetAuthorRow(
    ulong ID,
    string Name
);

    // -- name: GetAuthor :one
    public async Task<GetAuthorRow> GetAuthorAsync(ulong authorId, CancellationToken cancellationToken = defa
ult)
    {
        await using var command = new YdbCommand(
            "SELECT id, name FROM authors WHERE id = $author_id;", _connection) { Transaction = _transaction
};
        command.Parameters.Add(new YdbParameter("$author_id", DbType.UInt64, authorId));
        await using var reader = await command.ExecuteReaderAsync(cancellationToken).ConfigureAwait(false);
        if (!await reader.ReadAsync(cancellationToken).ConfigureAwait(false))
        {
            throw new InvalidOperationException("query returned no rows");
        }
        return GetAuthorRowFrom(reader);
    }

    private static GetAuthorRow GetAuthorRowFrom(DbDataReader reader) => new(
        reader.GetFieldValue<ulong>(0),
        reader.GetFieldValue<string>(1)
    );
```

Dapper

```
public sealed record GetAuthorRow(
    ulong ID,
    string Name
);

    // -- name: GetAuthor :one
```

```
    public async Task<GetAuthorRow> GetAuthorAsync(ulong authorId, CancellationToken cancellationToken = defa
ult, int? commandTimeout = null)
    {
        var parameters = new YdbParameters(
            new YdbParameter("$author_id", DbType.UInt64, authorId)
        );

        var command = new CommandDefinition(
            commandText: """
            SELECT id, name FROM authors WHERE id = $author_id;
            """,
            parameters: parameters,
            transaction: _transaction,
            commandTimeout: commandTimeout,
            cancellationToken: cancellationToken);

        return await _connection.QueryFirstAsync<GetAuthorRow>(command).ConfigureAwait(false);
    }
```

TypeScript

YDB SDK

```
// -- name: GetAuthor :one
  async getAuthor(authorId: bigint, configure?: ConfigureQuery): Promise<GetAuthorRow | null> {
    const stmt = this.#sql<[GetAuthorRow]>`SELECT id, name FROM authors WHERE id = $author_id;`
      .parameter("author_id", new Uint64(authorId));
    configure?.(stmt);
    const [rows] = await stmt;

    return rows[0] ?? null;
  }
```

Rust

YDB SDK

```
// -- name: GetAuthor :one
    #[builder(on(String, into))]
    pub async fn author(&mut self, author_id: u64) -> ydb::YdbResult<GetAuthorRow> {
        let mut row = self
            .client
            .query_row(r"SELECT id, name FROM authors WHERE id = $author_id;")
            .param("$author_id", author_id)
            .await?;
        Ok(GetAuthorRow {
            id: row.remove_field(0)?.try_into()?,
            name: row.remove_field(1)?.try_into()?,
        })
    }
```

PHP

YDB SDK

```
final class GetAuthorRow
{
    public function __construct(
        public readonly string $id,
        public readonly string $name
    ) {}
}
```

Kotlin

YDB SDK

```
data class GetAuthorRow(
    val id: Long,
    val name: String
)

    // -- name: GetAuthor :one
    fun getAuthor(authorId: Long): GetAuthorRow? {
        val _params = Params.create()
        _params.put("\$author_id", PrimitiveValue.newUint64(authorId))
        val _query = if (transaction != null) {
            QueryReader.readFrom(transaction.createQuery(
                "SELECT id, name FROM authors WHERE id = \$author_id;", _params)).join().getValue()
        } else {
            client!!.supplyResult { _session ->
                QueryReader.readFrom(_session.createQuery(
                "SELECT id, name FROM authors WHERE id = \$author_id;", TxMode.SERIALIZABLE_RW, _params))
            }.join().getValue()
        }
        kotlin.check(_query.getResultSetCount() == 1) { "Expected one result set" }
        val _rows = _query.getResultSet(0)
        if (!_rows.next()) return null
```

```
        val _value0: Long = _rows.getColumn(0).getUint64()
        val _value1: String = _rows.getColumn(1).getText()
        return GetAuthorRow(_value0, _value1)
    }
```

JDBC

```
data class GetAuthorRow(
    val id: Long,
    val name: String
)

    // -- name: GetAuthor :one
    fun getAuthor(authorId: Long): GetAuthorRow? {
        client.prepareStatement(
            "SELECT id, name FROM authors WHERE id = ?;").use { _prepared ->
            _prepared.setObject(1, PrimitiveValue.newUint64(authorId))
            _prepared.executeQuery().use { _rows ->
                if (!_rows.next()) return null
                val _value0: Long = _rows.getLong(1)
                val _value1: String = _rows.getString(2)
                return GetAuthorRow(_value0, _value1)
            }
        }
    }
```

Exposed

```
data class GetAuthorRow(
    val id: Long,
    val name: String
)

    // -- name: GetAuthor :one
    fun getAuthor(authorId: Long): GetAuthorRow? {
        val _connection = client.connection.connection as java.sql.Connection
        _connection.prepareStatement(
            "SELECT id, name FROM authors WHERE id = ?;").use { _prepared ->
            _prepared.setObject(1, PrimitiveValue.newUint64(authorId))
            _prepared.executeQuery().use { _rows ->
                if (!_rows.next()) return null
                val _value0: Long = _rows.getLong(1)
                val _value1: String = _rows.getString(2)
                return GetAuthorRow(_value0, _value1)
            }
        }
    }
```

Подготовка приложения

Генератор не создаёт таблицы в базе. Примените `schema.sql` через принятый в проекте механизм миграций или YDB CLI, а для примера одиночного чтения добавьте запись:

```
UPSERT INTO authors (id, name) VALUES (42, "Alice");
```

Подключите зависимости выбранного SDK/драйвера и настройте аутентификацию. Фрагменты ниже используют уже открытые соединения или клиенты и импортированный сгенерированный код. Полные проекты с зависимостями и настройкой подключения доступны в примерах проекта.

3. Выполнение отдельных запросов

Вызовите сгенерированный метод для автора с идентификатором 42:

C++

YDB SDK

```
authors::Queries queries{client};
auto author = queries.GetAuthor(42);
```

`client` — настроенный `NYdb::NQuery::TQueryClient`; нужны C++20 и YDB C++ SDK. Одиночный вызов использует SDK retry; `RetryQuerySync` повторяет всю транзакцию. `Queries` не завершает переданную транзакцию: пример явно делает commit или rollback. Полный пример.

userver

```
authors::Queries queries{table_client};
auto author = queries.GetAuthor(42);
```

`table_client` — `userver::ydb::TableClient` из компонента YDB; код выполняется в корутине userver. `RetryTx` повторяет весь callback и завершает транзакцию по `TxAction`; `kRollback` позволяет отменить изменения. `Utf8` — строковый тип userver для YQL Utf8. Полный пример.

Go

YDB SDK

```
q := authors.New(driver.Query())
author, err := q.GetAuthor(ctx, 42)
if err != nil {
    return err
}
fmt.Println(author.Name)
```

`driver` — открытый `*ydb.Driver`, `ctx` — контекст. Импортируйте сгенерированный пакет как `authors`, а для транзакции — `github.com/ydb-platform/ydb-go-sdk/v3/query`. Фрагменты находятся в функции, возвращающей `error`.

database/sql

```
q := authors.New(connection)
author, err := q.GetAuthor(ctx, 42)
if err != nil {
    return err
}
fmt.Println(author.Name)
```

`connection` — открытый `*sql.DB` с драйвером YDB, `ctx` — контекст. Импортируйте сгенерированный пакет как `authors`. Фрагменты находятся в функции, возвращающей `error`; при повторе выполните весь транзакционный блок заново.

Java

YDB SDK

```
GetAuthorRow author = retry.supplyResult(session -> {
    var tx = session.createNewTransaction(TxMode.SERIALIZABLE_RW);
    try {
        var row = new Queries(tx).getAuthor(42L).orElseThrow();
        return CompletableFuture.completedFuture(tx.commit().join().map(info -> row));
    } finally {
        if (tx.isActive()) {
            tx.rollback().join().expectSuccess();
        }
    }
}).join().getValue();
```

`retry` имеет тип `SessionRetryContext` и настроен приложением; импорты: `TxMode`, `CompletableFuture`, сгенерированные `Queries` и `GetAuthorRow`. Даже одиночный вызов Java native использует открытую транзакцию. `retry` повторяет всю функцию с новой сессией; завершение транзакции остаётся в вызывающем коде.

JDBC

```
GetAuthorRow author = new Queries(connection).getAuthor(42L).orElseThrow();
```

`connection` имеет тип `java.sql.Connection`, исходно работает в autocommit и не содержит другой транзакции. Импортированы сгенерированные `Queries` и `GetAuthorRow`. Код транзакции возвращает autocommit после commit или rollback; исключения передаются вызывающему коду. Повтор всей транзакции при необходимости организует приложение.

jOOQ

```
GetAuthorRow author = new Queries(dsl).getAuthor(ULong.valueOf(42)).orElseThrow();
```

`dsl` имеет тип `YdbDSLContext`; импорты: `YDB`, `ULong`, сгенерированные `Queries` и `GetAuthorRow`. Для отдельного вызова соединение работает в autocommit. `transactionResult` фиксирует изменения при успехе и откатывает при исключении; `YDB.using(configuration)` сохраняет соединение текущей транзакции. Повторы организует приложение вокруг всей транзакции.

Python

YDB SDK

```
import ydb
from python.ydb.queries import Querier

queries = Querier(pool, retry_settings=ydb.RetrySettings(idempotent=True))
author = queries.get_author(42)
```

`pool` — готовый `ydb.QuerySessionPool`. `retry_tx_sync` повторяет всю транзакцию и выполняет commit после callback; callback не должен иметь внешних побочных эффектов. В примере повторяемый UPSERT задаёт фиксированные значения. Запускаемый пример.

DB-API

```
from python.dbapi.queries import Querier

author = Querier(connection).get_author(42)
```

`connection` — открытое соединение `ydb_dbapi` без активной транзакции. По умолчанию драйвер использует AUTOCOMMIT: для общей транзакции нужно выбрать SERIALIZABLE до `begin()`. Повторы всей транзакции организует вызывающий код. Запускаемый пример.

SQLAlchemy

```
from python.sqlalchemy.queries import Querier

with engine.connect() as connection:
    author = Querier(connection).get_author(42)
```

`engine` — готовый SQLAlchemy Engine с диалектом `yql+ydb`. SERIALIZABLE задаётся до `begin()`: один transaction block не отменяет AUTOCOMMIT драйвера. Контекст выполняет commit или rollback; повторы всей транзакции организует вызывающий код. Запускаемый пример.

C#

ADO.NET

```
var author = await dataSource.ExecuteAsync(
    (connection, ct) => new Authors.Queries(connection).GetAuthorAsync(42UL, ct),
    cancellationToken);
```

Нужны .NET 8, Ydb.Sdk, настроенный `YdbDataSource dataSource` и `CancellationToken cancellationToken`. `ExecuteAsync` повторяет весь callback согласно политике data source. Для нескольких вызовов в одной транзакции используйте `ExecuteInTransactionAsync`: провайдер управляет коммитом, откатом и повтором всей транзакции. Полный пример. Если строка не найдена, `GetAuthorAsync` выбрасывает `InvalidOperationException`.

Dapper

```
var author = await dataSource.ExecuteAsync(
    (connection, ct) => new Authors.Queries(connection).GetAuthorAsync(42UL, ct),
    cancellationToken);
```

Нужны .NET 8, Ydb.Sdk и Dapper, настроенный `YdbDataSource dataSource` и `CancellationToken cancellationToken`. `ExecuteAsync` повторяет весь callback согласно политике data source. Для нескольких вызовов в одной транзакции используйте `ExecuteInTransactionAsync`: провайдер управляет коммитом, откатом и повтором всей транзакции. Полный пример. Если строка не найдена, `GetAuthorAsync` выбрасывает `InvalidOperationException`.

TypeScript

YDB SDK

```
import { Queries } from "./typescript/queries.js";

const queries = new Queries(sql);
const author = await queries.getAuthor(42n);
console.log(author?.name);
```

`sql` — функция `query(driver)` из `@ydbjs/query` для подготовленного драйвера. `transaction` фиксирует изменения после успешного callback и откатывает при ошибке.

Rust

YDB SDK

```
use generated::queries::Queries;

let mut queries = Queries::new(&mut query_client);
let author = queries.author().author_id(42).call().await?;
println!("{}", author.name);
```

`query_client` — изменяемый `ydb::QueryClient`. Подключите файлы из `rust/` как модуль `generated` и зависимости `ydb` и `bon`. SDK фиксирует транзакцию после успешного callback; повторяется весь callback.

PHP

YDB SDK

```
use Authors\Queries;

$queries = new Queries($table, idempotent: true);
$author = $queries->getAuthor('42');
```

`$table` — готовый `YdbPlatform\Ydb\Table`; сгенерированные классы подключены через autoload. `Uint64` передаётся десятичной строкой. Для этого SELECT безопасные повторы явно включены через `idempotent: true`. Запускаемый пример.

Kotlin

YDB SDK

```
val author = requireNotNull(Queries(retry).getAuthor(42L))
```

`retry` имеет тип `SessionRetryContext` и настроен приложением; импорты: `TxMode`, `CompletableFuture` и сгенерированный `Queries`. Одиночный вызов использует retry-контекст, а составная операция передаёт хелперам одну `QueryTransaction`. Внешний `retry` повторяет всю транзакцию.

JDBC

```
val author = requireNotNull(Queries(connection).getAuthor(42L))
```

`connection` имеет тип `java.sql.Connection`, исходно работает в autocommit и не содержит другой транзакции; импортирован сгенерированный `Queries`. Код транзакции возвращает autocommit после commit или rollback; исключения передаются вызывающему коду. Повтор всей транзакции при необходимости организует приложение.

Exposed

```
val author = ydbTransaction(database) {
    requireNotNull(Queries(this).getAuthor(42L))
}
```

`database` представляет настроенный Exposed `Database` с зарегистрированным YDB-диалектом; импорты: `tech.ydb.exposed.dialect.ydbTransaction` и сгенерированный `Queries`. Блок `ydbTransaction` передаёт одну `JdbcTransaction`, фиксирует её при успехе и откатывает при исключении. Политика повторов применяется к блоку целиком.

4. Выполнение нескольких запросов в транзакции

В каждом примере два сгенерированных метода выполняют два YQL-запроса в одной транзакции: `UpsertAuthor` записывает имя, затем `GetAuthor` читает его. Приложение управляет границами транзакции и повторами; при повторе выполняйте весь блок. В callback, который SDK может повторить, не помещайте внешние побочные эффекты.

C++

YDB SDK

```
std::optional<authors::GetAuthorRow> author;
const auto status = client.RetryQuerySync([&](NYdb::NQuery::TSession session) -> NYdb::TStatus {
    auto started = session.BeginTransaction(
        NYdb::NQuery::TTxSettings::SerializableRW()).GetValueSync();
    if (!started.IsSuccess()) {
        return started;
    }
    auto transaction = started.GetTransaction();
    try {
        authors::Queries queries{transaction};
        queries.UpsertAuthor(42, "Alice");
        author = queries.GetAuthor(42);
    } catch (const NYdb::NStatusHelpers::TYdbErrorException& error) {
        transaction.Rollback().GetValueSync();
        return error.GetStatus();
    } catch (...) {
        transaction.Rollback().GetValueSync();
        throw;
    }
    return transaction.Commit().GetValueSync();
});
NYdb::NStatusHelpers::ThrowOnError(status);
```

userver

```
std::optional<authors::GetAuthorRow> author;
table_client.RetryTx("upsert-and-read-author", {}, [&](::userver::ydb::TxActor& transaction) {
    authors::Queries queries{transaction};
    queries.UpsertAuthor(42, ::userver::ydb::Utf8{"Alice"});
    author = queries.GetAuthor(42);
    return ::userver::ydb::TxAction::kCommit;
});
```

Go

YDB SDK

```
return driver.Query().DoTx(ctx, func(ctx context.Context, tx query.TxActor) error {
    q := authors.New(tx)
    if err := q.UpsertAuthor(ctx, authors.UpsertAuthorParams{
        AuthorID: 42, AuthorName: "Alice",
    }); err != nil {
        return err
    }
    _, err := q.GetAuthor(ctx, 42)
    return err
})
```

database/sql

```
tx, err := connection.BeginTx(ctx, nil)
if err != nil {
    return err
```

```
}
defer tx.Rollback()

q := authors.New(tx)
if err := q.UpsertAuthor(ctx, authors.UpsertAuthorParams{
    AuthorID: 42, AuthorName: "Alice",
}); err != nil {
    return err
}
if _, err := q.GetAuthor(ctx, 42); err != nil {
    return err
}
return tx.Commit()
```

Java

YDB SDK

```
GetAuthorRow author = retry.supplyResult(session -> {
    var tx = session.createNewTransaction(TxMode.SERIALIZABLE_RW);
    try {
        var queries = new Queries(tx);
        queries.upsertAuthor(42L, "Alice");
        var row = queries.getAuthor(42L).orElseThrow();
        return CompletableFuture.completedFuture(tx.commit().join().map(info -> row));
    } finally {
        if (tx.isActive()) {
            tx.rollback().join().expectSuccess();
        }
    }
}).join().getValue();
```

JDBC

```
connection.setAutoCommit(false);
try {
    var queries = new Queries(connection);
    queries.upsertAuthor(42L, "Alice");
    GetAuthorRow author = queries.getAuthor(42L).orElseThrow();
    connection.commit();
} catch (Exception error) {
    connection.rollback();
    throw error;
} finally {
    connection.setAutoCommit(true);
}
```

jOOQ

```
GetAuthorRow author = dsl.transactionResult(configuration -> {
    var queries = new Queries(YDB.using(configuration));
    var id = ULong.valueOf(42);
    queries.upsertAuthor(id, "Alice");
    return queries.getAuthor(id).orElseThrow();
});
```

Python

YDB SDK

```
import ydb
from python.ydb.queries import Querier


def update_author(tx):
    queries = Querier(tx)
    queries.upsert_author(42, "Alice")
    return queries.get_author(42)


author = pool.retry_tx_sync(
    update_author,
    tx_mode=ydb.QuerySerializableReadWrite(),
    retry_settings=ydb.RetrySettings(idempotent=True),
)
```

DB-API

```
import ydb_dbapi
from python.dbapi.queries import Querier

connection.set_isolation_level(ydb_dbapi.IsolationLevel.SERIALIZABLE)
connection.begin()
try:
    queries = Querier(connection)
    queries.upsert_author(42, "Alice")
```

```
    author = queries.get_author(42)
    connection.commit()
except Exception:
    connection.rollback()
    raise
```

SQLAlchemy

```
from python.sqlalchemy.queries import Querier

with engine.connect().execution_options(isolation_level="SERIALIZABLE") as connection:
    with connection.begin():
        queries = Querier(connection)
        queries.upsert_author(42, "Alice")
        author = queries.get_author(42)
```

C#

ADO.NET

```
var author = await dataSource.ExecuteInTransactionAsync(async (connection, ct) =>
{
    var queries = new Authors.Queries(connection);
    await queries.UpsertAuthorAsync(new Authors.UpsertAuthorParams(42UL, "Alice"), ct);
    return await queries.GetAuthorAsync(42UL, ct);
}, cancellationToken: cancellationToken);
```

Dapper

```
var author = await dataSource.ExecuteInTransactionAsync(async (connection, ct) =>
{
    var queries = new Authors.Queries(connection);
    await queries.UpsertAuthorAsync(new Authors.UpsertAuthorParams(42UL, "Alice"), ct);
    return await queries.GetAuthorAsync(42UL, ct);
}, cancellationToken: cancellationToken);
```

TypeScript

YDB SDK

```
import { Queries } from "./typescript/queries.js";
import type { ConfigureQuery } from "./typescript/queries.js";

const author = await sql.transaction(async (tx, signal) => {
  const queries = new Queries(tx);
  const configure: ConfigureQuery = (stmt) => { stmt.signal(signal); };
  await queries.upsertAuthor({ authorId: 42n, authorName: "Alice" }, configure);
  return queries.getAuthor(42n, configure);
});
console.log(author?.name);
```

Rust

YDB SDK

```
use generated::queries::Queries;

let author = query_client.retry_tx(ydb::closure!(async |tx| {
    let mut queries = Queries::new(tx);
    queries.upsert_author()
        .author_id(42)
        .author_name("Alice")
        .call().await?;
    Ok(queries.author().author_id(42).call().await?)
})).await?;
println!("{}", author.name);
```

PHP

YDB SDK

```
use Authors\Queries;
use Authors\UpsertAuthorParams;

$session = $table->session();
$txId = $session->beginTransaction();
$queries = (new Queries($table))->withTx($session, $txId);

$queries->upsertAuthor(new UpsertAuthorParams('42', 'Alice'));
$author = $queries->getAuthor('42');

$session->commitTransaction();
```

Оба запроса выполняются в одной транзакции. `$table` — настроенный `YdbPlatform\Ydb\Table`, сгенерированные классы доступны через автозагрузку. Обработка ошибок опущена: при ошибке приложение должно вызвать `$session-`

`>rollbackTransaction()`. Связанный helper не выполняет повторные попытки отдельных запросов; повторять нужно всю транзакцию. После коммита или отката сессия возвращается в пул, а для следующей транзакции нужно получить сессию заново и создать новый связанный helper. Контракт PHP.

Kotlin

YDB SDK

```
val author = retry.supplyResult { session ->
    val tx = session.createNewTransaction(TxMode.SERIALIZABLE_RW)
    try {
        val queries = Queries(tx)
        queries.upsertAuthor(42L, "Alice")
        val row = requireNotNull(queries.getAuthor(42L))
        CompletableFuture.completedFuture(tx.commit().join().map { row })
    } finally {
        if (tx.isActive) {
            tx.rollback().join().expectSuccess()
        }
    }
}.join().getValue()
```

JDBC

```
connection.autoCommit = false
try {
    val queries = Queries(connection)
    queries.upsertAuthor(42L, "Alice")
    val author = requireNotNull(queries.getAuthor(42L))
    connection.commit()
} catch (error: Exception) {
    connection.rollback()
    throw error
} finally {
    connection.autoCommit = true
}
```

Exposed

```
val author = ydbTransaction(database) {
    val queries = Queries(this)
    queries.upsertAuthor(42L, "Alice")
    requireNotNull(queries.getAuthor(42L))
}
```

Актуальный состав поддерживаемых команд, конфигурации и конструкций YQL приведён в контракте совместимости sqlc-ydb. Если ваш запрос не поддерживается или сгенерированный код неудобен, создайте issue с минимальными схемой, запросом, конфигурацией и выводом `sqlc-ydb version --verbose`.

Туториал по YQL - Обзор

Из этого туториала вы узнаете, как выполнять базовые операции с данными в YDB, и познакомитесь с синтаксисом YQL. Подробное описание этого синтаксиса находится в справочной документации по YQL.

Туториал состоит из 15 шагов:

1. Создание таблиц
2. Добавление данных в таблицы
3. Выборка данных из всех колонок
4. Выборка данных из определенных колонок
5. Сортировка и фильтрация
6. Агрегирование данных
7. Дополнительные условия выборки
8. Объединение таблиц с помощью JOIN
9. Вставка и модификация данных с помощью REPLACE
10. Вставка и модификация данных с помощью UPSERT
11. Вставка данных с помощью INSERT
12. Обновление данных с помощью UPDATE
13. Удаление данных
14. Добавление и удаление колонок
15. Удаление таблиц

Создание таблиц

Создайте строковые таблицы для OLTP-операций или колоночные таблицы для OLAP-нагрузок и задайте для них схемы данных с помощью конструкции CREATE TABLE. Подробнее о типах таблиц в YDB и области их применения можно прочесть в статье Таблица

> **Примечание**
>
> Ключевые слова не чувствительны к регистру и написаны заглавными буквами для наглядности.

Создание строковых таблиц

```
CREATE TABLE series         -- series — имя таблицы.
(                           -- Должно быть уникальным в пределах каталога.
    series_id Uint64,
    title Utf8,
    series_info Utf8,
    release_date Uint64,
    PRIMARY KEY (series_id) -- Первичный ключ — это столбец или
                            -- комбинация столбцов, однозначно идентифицирующих
                            -- каждую строку в таблице (может содержать только
                            -- неповторяющиеся значения). Для таблицы может быть
                            -- указан только один первичный ключ. Для каждой таблицы
                            -- YDB указание первичного ключа является обязательным.
);

CREATE TABLE seasons
(
    series_id Uint64,
    season_id Uint64,
    title Utf8,
    first_aired Uint64,
    last_aired Uint64,
    PRIMARY KEY (series_id, season_id)
);

CREATE TABLE episodes
(
    series_id Uint64,
    season_id Uint64,
    episode_id Uint64,
    title Utf8,
    air_date Uint64,
    PRIMARY KEY (series_id, season_id, episode_id)
);
```

Создание колоночных таблиц

Колоночная таблица в YQL создаётся путём указания в конструкции `CREATE TABLE` блока `WITH` с параметром `STORE = COLUMN`.

```
CREATE TABLE series         -- series — имя таблицы.
(                           -- Должно быть уникальным в пределах каталога.
    series_id Uint64 NOT NULL, -- первичный ключ колоночной таблицы не может быть NULLABLE
    title Utf8,
    series_info Utf8,
    release_date Uint64,
    PRIMARY KEY (series_id) -- Первичный ключ — это столбец или
                            -- комбинация столбцов, однозначно идентифицирующих
                            -- каждую строку в таблице (может содержать только
                            -- неповторяющиеся значения). Для таблицы может быть
                            -- указан только один первичный ключ. Для каждой таблицы
                            -- YDB указание первичного ключа является обязательным.
)
PARTITION BY HASH(series_id) -- Партиционирование таблицы по хешу от набора колонок
WITH (
    STORE = COLUMN          -- STORE указывает на тип таблицы
                            -- если указан COLUMN, то будет создана колоночная таблица
);

CREATE TABLE seasons
(
    series_id Uint64 NOT NULL,
    season_id Uint64 NOT NULL,
    title Utf8,
    first_aired Uint64,
    last_aired Uint64,
    PRIMARY KEY (series_id, season_id)
)
WITH (
    STORE = COLUMN
);

CREATE TABLE episodes
(
    series_id Uint64 NOT NULL,
    season_id Uint64 NOT NULL,
    episode_id Uint64 NOT NULL,
```

```
    title Utf8,
    air_date Uint64,
    PRIMARY KEY (series_id, season_id, episode_id)
)
WITH (
    STORE = COLUMN
);
```

Добавление данных в таблицы

Наполните данными созданные ранее таблицы с помощью конструкции REPLACE INTO.

```
REPLACE INTO series (series_id, title, release_date, series_info)

VALUES

    -- По умолчанию числовые литералы имеют тип Int32,
    -- если значение соответствует диапазону.
    -- Иначе тип преобразуется в Int64.
    (
        1,
        "IT Crowd",
        CAST(Date("2006-02-03") AS Uint64),   -- CAST преобразует один тип данных в другой.
                                              -- Существует возможность преобразования строкового
                                              -- литерала в литерал простого типа.
                                              -- Функция Date() преобразует строковый
                                              -- литерал в формате ISO 8601 в дату.

        "The IT Crowd is a British sitcom produced by Channel 4, written by Graham Linehan, produced by Ash A
talla and starring Chris O'Dowd, Richard Ayoade, Katherine Parkinson, and Matt Berry."),
    (
        2,
        "Silicon Valley",
        CAST(Date("2014-04-06") AS Uint64),
        "Silicon Valley is an American comedy television series created by Mike Judge, John Altschuler and Da
ve Krinsky. The series focuses on five young men who founded a startup company in Silicon Valley."
    )
    ;

REPLACE INTO seasons (series_id, season_id, title, first_aired, last_aired)
VALUES
    (1, 1, "Season 1", CAST(Date("2006-02-03") AS Uint64), CAST(Date("2006-03-03") AS Uint64)),
    (1, 2, "Season 2", CAST(Date("2007-08-24") AS Uint64), CAST(Date("2007-09-28") AS Uint64)),
    (1, 3, "Season 3", CAST(Date("2008-11-21") AS Uint64), CAST(Date("2008-12-26") AS Uint64)),
    (1, 4, "Season 4", CAST(Date("2010-06-25") AS Uint64), CAST(Date("2010-07-30") AS Uint64)),
    (2, 1, "Season 1", CAST(Date("2014-04-06") AS Uint64), CAST(Date("2014-06-01") AS Uint64)),
    (2, 2, "Season 2", CAST(Date("2015-04-12") AS Uint64), CAST(Date("2015-06-14") AS Uint64)),
    (2, 3, "Season 3", CAST(Date("2016-04-24") AS Uint64), CAST(Date("2016-06-26") AS Uint64)),
    (2, 4, "Season 4", CAST(Date("2017-04-23") AS Uint64), CAST(Date("2017-06-25") AS Uint64)),
    (2, 5, "Season 5", CAST(Date("2018-03-25") AS Uint64), CAST(Date("2018-05-13") AS Uint64))
;

REPLACE INTO episodes (series_id, season_id, episode_id, title, air_date)
VALUES
    (1, 1, 1, "Yesterday's Jam", CAST(Date("2006-02-03") AS Uint64)),
    (1, 1, 2, "Calamity Jen", CAST(Date("2006-02-03") AS Uint64)),
    (1, 1, 3, "Fifty-Fifty", CAST(Date("2006-02-10") AS Uint64)),
    (1, 1, 4, "The Red Door", CAST(Date("2006-02-17") AS Uint64)),
    (1, 1, 5, "The Haunting of Bill Crouse", CAST(Date("2006-02-24") AS Uint64)),
    (1, 1, 6, "Aunt Irma Visits", CAST(Date("2006-03-03") AS Uint64)),
    (1, 2, 1, "The Work Outing", CAST(Date("2006-08-24") AS Uint64)),
    (1, 2, 2, "Return of the Golden Child", CAST(Date("2007-08-31") AS Uint64)),
    (1, 2, 3, "Moss and the German", CAST(Date("2007-09-07") AS Uint64)),
    (1, 2, 4, "The Dinner Party", CAST(Date("2007-09-14") AS Uint64)),
    (1, 2, 5, "Smoke and Mirrors", CAST(Date("2007-09-21") AS Uint64)),
    (1, 2, 6, "Men Without Women", CAST(Date("2007-09-28") AS Uint64)),
    (1, 3, 1, "From Hell", CAST(Date("2008-11-21") AS Uint64)),
    (1, 3, 2, "Are We Not Men?", CAST(Date("2008-11-28") AS Uint64)),
    (1, 3, 3, "Tramps Like Us", CAST(Date("2008-12-05") AS Uint64)),
    (1, 3, 4, "The Speech", CAST(Date("2008-12-12") AS Uint64)),
    (1, 3, 5, "Friendface", CAST(Date("2008-12-19") AS Uint64)),
    (1, 3, 6, "Calendar Geeks", CAST(Date("2008-12-26") AS Uint64)),
    (1, 4, 1, "Jen The Fredo", CAST(Date("2010-06-25") AS Uint64)),
    (1, 4, 2, "The Final Countdown", CAST(Date("2010-07-02") AS Uint64)),
    (1, 4, 3, "Something Happened", CAST(Date("2010-07-09") AS Uint64)),
    (1, 4, 4, "Italian For Beginners", CAST(Date("2010-07-16") AS Uint64)),
    (1, 4, 5, "Bad Boys", CAST(Date("2010-07-23") AS Uint64)),
    (1, 4, 6, "Reynholm vs Reynholm", CAST(Date("2010-07-30") AS Uint64)),
    (2, 1, 1, "Minimum Viable Product", CAST(Date("2014-04-06") AS Uint64)),
    (2, 1, 2, "The Cap Table", CAST(Date("2014-04-13") AS Uint64)),
    (2, 1, 3, "Articles of Incorporation", CAST(Date("2014-04-20") AS Uint64)),
    (2, 1, 4, "Fiduciary Duties", CAST(Date("2014-04-27") AS Uint64)),
    (2, 1, 5, "Signaling Risk", CAST(Date("2014-05-04") AS Uint64)),
    (2, 1, 6, "Third Party Insourcing", CAST(Date("2014-05-11") AS Uint64)),
    (2, 1, 7, "Proof of Concept", CAST(Date("2014-05-18") AS Uint64)),
    (2, 1, 8, "Optimal Tip-to-Tip Efficiency", CAST(Date("2014-06-01") AS Uint64)),
    (2, 2, 1, "Sand Hill Shuffle", CAST(Date("2015-04-12") AS Uint64)),
    (2, 2, 2, "Runaway Devaluation", CAST(Date("2015-04-19") AS Uint64)),
    (2, 2, 3, "Bad Money", CAST(Date("2015-04-26") AS Uint64)),
    (2, 2, 4, "The Lady", CAST(Date("2015-05-03") AS Uint64)),
    (2, 2, 5, "Server Space", CAST(Date("2015-05-10") AS Uint64)),
    (2, 2, 6, "Homicide", CAST(Date("2015-05-17") AS Uint64)),
    (2, 2, 7, "Adult Content", CAST(Date("2015-05-24") AS Uint64)),
    (2, 2, 8, "White Hat/Black Hat", CAST(Date("2015-05-31") AS Uint64)),
    (2, 2, 9, "Binding Arbitration", CAST(Date("2015-06-07") AS Uint64)),
    (2, 2, 10, "Two Days of the Condor", CAST(Date("2015-06-14") AS Uint64)),
    (2, 3, 1, "Founder Friendly", CAST(Date("2016-04-24") AS Uint64)),
    (2, 3, 2, "Two in the Box", CAST(Date("2016-05-01") AS Uint64)),
    (2, 3, 3, "Meinertzhagen's Haversack", CAST(Date("2016-05-08") AS Uint64)),
```

```
    (2, 3, 4, "Maleant Data Systems Solutions", CAST(Date("2016-05-15") AS Uint64)),
    (2, 3, 5, "The Empty Chair", CAST(Date("2016-05-22") AS Uint64)),
    (2, 3, 6, "Bachmanity Insanity", CAST(Date("2016-05-29") AS Uint64)),
    (2, 3, 7, "To Build a Better Beta", CAST(Date("2016-06-05") AS Uint64)),
    (2, 3, 8, "Bachman's Earnings Over-Ride", CAST(Date("2016-06-12") AS Uint64)),
    (2, 3, 9, "Daily Active Users", CAST(Date("2016-06-19") AS Uint64)),
    (2, 3, 10, "The Uptick", CAST(Date("2016-06-26") AS Uint64)),
    (2, 4, 1, "Success Failure", CAST(Date("2017-04-23") AS Uint64)),
    (2, 4, 2, "Terms of Service", CAST(Date("2017-04-30") AS Uint64)),
    (2, 4, 3, "Intellectual Property", CAST(Date("2017-05-07") AS Uint64)),
    (2, 4, 4, "Teambuilding Exercise", CAST(Date("2017-05-14") AS Uint64)),
    (2, 4, 5, "The Blood Boy", CAST(Date("2017-05-21") AS Uint64)),
    (2, 4, 6, "Customer Service", CAST(Date("2017-05-28") AS Uint64)),
    (2, 4, 7, "The Patent Troll", CAST(Date("2017-06-04") AS Uint64)),
    (2, 4, 8, "The Keenan Vortex", CAST(Date("2017-06-11") AS Uint64)),
    (2, 4, 9, "Hooli-Con", CAST(Date("2017-06-18") AS Uint64)),
    (2, 4, 10, "Server Error", CAST(Date("2017-06-25") AS Uint64)),
    (2, 5, 1, "Grow Fast or Die Slow", CAST(Date("2018-03-25") AS Uint64)),
    (2, 5, 2, "Reorientation", CAST(Date("2018-04-01") AS Uint64)),
    (2, 5, 3, "Chief Operating Officer", CAST(Date("2018-04-08") AS Uint64)),
    (2, 5, 4, "Tech Evangelist", CAST(Date("2018-04-15") AS Uint64)),
    (2, 5, 5, "Facial Recognition", CAST(Date("2018-04-22") AS Uint64)),
    (2, 5, 6, "Artificial Emotional Intelligence", CAST(Date("2018-04-29") AS Uint64)),
    (2, 5, 7, "Initial Coin Offering", CAST(Date("2018-05-06") AS Uint64)),
    (2, 5, 8, "Fifty-One Percent", CAST(Date("2018-05-13") AS Uint64));
```

Выборка данных из всех колонок

Выберите все колонки из таблиц с помощью оператора SELECT.

> **Примечание**
>
> Предполагается, что вы уже создали таблицы ранее на шаге Создание таблиц и заполнили их данными на шаге Добавление данных в таблицы.

```
SELECT         -- Оператор выбора данных.

    *          -- Выбор всех колонок из таблицы.

FROM `<table_name>`; -- Таблица, из которой нужно выбрать данные.
                     -- Можно выбрать данные из таблиц: series, seasons, episodes.
```

Выборка данных из определенных колонок

Выберите данные из колонок `series_id`, `release_date` и `title`. При этом переименуйте `title` в `series_title` и преобразуйте тип `release_date` из `Uint32` в `Date`.

> **Примечание**
>
> Предполагается, что вы уже создали таблицы ранее на шаге Создание таблиц и заполнили их данными на шаге Добавление данных в таблицы.

```
SELECT
    series_id,             -- Имена колонок (series_id, release_date, title)
                           -- перечисляются через запятую.

    title AS series_title, -- С помощью AS можно переименовать столбцы
                           -- или дать имя произвольному выражению

    CAST(release_date AS Date) AS release_date

FROM series;
```

Сортировка и фильтрация

Выберите первые три эпизода из всех сезонов IT Crowd за исключением первого.

> **Примечание**
>
> Предполагается, что вы уже создали таблицы ранее на шаге Создание таблиц и заполнили их данными на шаге Добавление данных в таблицы.

```
SELECT
   series_id,
   season_id,
   episode_id,
   CAST(air_date AS Date) AS air_date,
   title

FROM episodes
WHERE
   series_id = 1      -- Список условий для формирования результата.
   AND season_id > 1  -- Логическое И (AND) используется для написания сложных условий.

ORDER BY              -- Сортировка результатов.
   series_id,         -- ORDER BY сортирует значения по одному или нескольким
   season_id,         -- столбцам. Столбцы перечисляются через запятую.
   episode_id

LIMIT 3               -- LIMIT N после ORDER BY означает
                      -- «получить первые N» или «последние N» результатов
;                     -- в зависимости от порядка сортировки.
```

Агрегирование данных

Узнайте количество эпизодов с уникальными названиями для каждого сезона каждого сериала.

> **Примечание**
>
> Предполагается, что вы уже создали таблицы ранее на шаге Создание таблиц и заполнили их данными на шаге Добавление данных в таблицы.

```
SELECT
    series_id,
    season_id,
    COUNT (*) AS cnt -- Агрегатная функция COUNT возвращает количество строк,
                     -- полученных в результате выполнения запроса.
                     -- Звездочка (*) указывает, что функция COUNT
                     -- посчитает количество всех строк в таблице.
                     -- COUNT(*) возвращает количество строк в
                     -- указанной таблице с учетом повторяющихся строк.
                     -- Функция считает каждую строку отдельно.
                     -- В результат также входят строки, содержащие значения null.
FROM episodes

GROUP BY
    series_id,       -- Результат выполнения запроса будет выведен в порядке указанных колонок.
    season_id        -- Несколько колонок разделяются запятой.
                     -- Другие колонки можно указать после выполнения SELECT, только если
                     -- они передаются в функцию агрегации.
ORDER BY
    series_id,
    season_id
;
```

Дополнительные условия выборки

Выберите все названия эпизодов первого сезона каждого сериала и отсортируйте их по имени.

> **Примечание**
>
> Предполагается, что вы уже создали таблицы ранее на шаге Создание таблиц и заполнили их данными на шаге Добавление данных в таблицы.

```
SELECT
    series_title,               -- series_title будет определен ниже в GROUP BY

    String::JoinFromList(       -- вызов C++ UDF,
                                -- см. ниже

        AGGREGATE_LIST(title),  -- функция агрегации,
                                -- возвращающая все переданные значения в виде списка

        ", "                    -- String::JoinFromList преобразует
                                -- элементы указанного списка (первый аргумент)
                                -- в строку, используя разделитель (второй аргумент)
    ) AS episode_titles
FROM episodes
WHERE series_id IN (1,2)        -- IN определяет набор значений в условии WHERE,
                                -- которые должны попасть в результат.
                                -- Синтаксис:
                                -- test_expression (NOT) IN
                                -- ( subquery | expression ` ,...n ` )
                                -- Если значение test_expression равно
                                -- любому значению, возвращаемому подзапросом, или равно
                                -- любому выражению из списка, разделенного запятыми,
                                -- результатом будет TRUE; в противном случае – FALSE.
                                -- NOT IN означает, что значение НЕ присутствует в результате подзапроса
                                -- или в выражении.
                                -- Предупреждение: использование значений null вместе с
                                -- IN или NOT IN может привести к нежелательным результатам.
AND season_id = 1
GROUP BY
    CASE                        -- CASE проверяет список условий
                                -- и возвращает одно из нескольких возможных
                                -- выражений. CASE может быть использован в любом
                                -- выражении или с любым оператором,
                                -- поддерживающим данное выражение. Например, CASE можно использовать
                                -- выражениях SELECT, UPDATE, DELETE,
                                -- и с операторами IN, WHERE, ORDER BY.
        WHEN series_id = 1
        THEN "IT Crowd"
        ELSE "Other series"
    END AS series_title         -- GROUP BY можно выполнить
                                -- для произвольного выражения.
                                -- Результат будет доступен в SELECT
                                -- через алиас, указанный с помощью AS.
;
```

Объединение таблиц с помощью JOIN

Объедините колонки исходных таблиц `seasons` и `series` и выведите все сезоны сериала IT Crowd в результирующей таблице с помощью оператора JOIN.

> **Примечание**
>
> Предполагается, что вы уже создали таблицы ранее на шаге Создание таблиц и заполнили их данными на шаге Добавление данных в таблицы.

```
SELECT
    sa.title AS season_title,    -- sa и sr — это «связующие названия»,
    sr.title AS series_title,    -- алиасы таблиц, объявленные ниже с помощью AS.
    sr.series_id,                -- Они используются, чтобы избежать
    sa.season_id                 -- неоднозначности в именах указанных колонок.

FROM
    seasons AS sa
INNER JOIN
    series AS sr
ON sa.series_id = sr.series_id
WHERE sa.series_id = 1
ORDER BY                         -- Сортировка результатов.
    sr.series_id,
    sa.season_id                 -- ORDER BY сортирует значения по одному
;                                -- или нескольким столбцам.
                                 -- Столбцы перечисляются через запятую.
```

Вставка и модификация данных с помощью REPLACE

Важно

В настоящее время одновременное использование колоночных и строковых таблиц поддерживается в транзакциях, в которых данные только читаются, но не изменяются. Поддержка транзакций с возможностью модификации данных при одновременном использовании строковых и колоночных таблиц находится в разработке.

Если попытаться выполнить операцию записи в транзакции, в которой задействованы и колоночные, и строковые таблицы, транзакция завершится с ошибкой:

```
Write transactions that use both row-oriented and column-oriented tables are disabled at current t
ime
```

Добавьте новые данные в таблицы с одновременным обновлением уже существующих данных с помощью конструкции REPLACE INTO.

Примечание

Предполагается, что вы уже создали таблицы ранее на шаге Создание таблиц и заполнили их данными на шаге Добавление данных в таблицы.

```
REPLACE INTO episodes
(
    series_id,
    season_id,
    episode_id,
    title,
    air_date
)
VALUES
(
    2,
    5,
    12,
    "Test Episode !!!",
    CAST(Date("2018-08-27") AS Uint64)
)
;

-- Вызов COMMIT используется, чтобы следующей операции SELECT
-- были видны изменения, сделанные в рамках предыдущей транзакции.
COMMIT;

-- Посмотреть результат:
SELECT * FROM episodes WHERE series_id = 2 AND season_id = 5;
```

Вставка и модификация данных с помощью UPSERT

Добавьте данные в таблицу с помощью конструкции UPSERT INTO.

> **Примечание**
>
> Предполагается, что вы уже создали таблицы ранее на шаге Создание таблиц и заполнили их данными на шаге Добавление данных в таблицы.

```
UPSERT INTO episodes
(
    series_id,
    season_id,
    episode_id,
    title,
    air_date
)
VALUES
(
    2,
    5,
    13,
    "Test Episode",
    CAST(Date("2018-08-27") AS Uint64)
)
;

COMMIT;

-- Посмотреть результат:
SELECT * FROM episodes WHERE series_id = 2 AND season_id = 5;
```

Вставка данных с помощью INSERT

Важно

В настоящее время одновременное использование колоночных и строковых таблиц поддерживается в транзакциях, в которых данные только читаются, но не изменяются. Поддержка транзакций с возможностью модификации данных при одновременном использовании строковых и колоночных таблиц находится в разработке.

Если попытаться выполнить операцию записи в транзакции, в которой задействованы и колоночные, и строковые таблицы, транзакция завершится с ошибкой:

```
Write transactions that use both row-oriented and column-oriented tables are disabled at current t
ime
```

Добавьте данные в таблицу с помощью конструкции INSERT INTO.

Примечание

Предполагается, что вы уже создали таблицы ранее на шаге Создание таблиц и заполнили их данными на шаге Добавление данных в таблицы.

```
INSERT INTO episodes
(
    series_id,
    season_id,
    episode_id,
    title,
    air_date
)
VALUES
(
    2,
    5,
    21,
    "Test 21",
    CAST(Date("2018-08-27") AS Uint64)
),                                          -- Строки разделяются запятыми.
(
    2,
    5,
    22,
    "Test 22",
    CAST(Date("2018-08-27") AS Uint64)
)
;

COMMIT;

-- Посмотреть результат:
SELECT * FROM episodes WHERE series_id = 2 AND season_id = 5;
```

Обновление данных с помощью UPDATE

Обновите данные в таблице с помощью оператора UPDATE.

> **Примечание**
>
> Предполагается, что вы уже создали таблицы ранее на шаге Создание таблиц и заполнили их данными на шаге Добавление данных в таблицы.

```
UPDATE episodes
SET title="test Episode 2"
WHERE
    series_id = 2
    AND season_id = 5
    AND episode_id = 12
;

COMMIT;

-- Посмотреть результат:
SELECT * FROM episodes WHERE series_id = 2 AND season_id = 5;

-- YDB не знает об изменениях, имевших место в начале транзакции,
-- поэтому сначала выполняет чтение. Невозможно обновить или удалить таблицу,
-- которая уже была изменена в рамках текущей транзакции. UPDATE ON и
-- DELETE ON позволяют читать, обновлять и удалять несколько строк в таблице
-- в рамках одной транзакции.

$to_update = (
    SELECT series_id,
           season_id,
           episode_id,
           Utf8("Yesterday's Jam UPDATED") AS title
    FROM episodes
    WHERE series_id = 1 AND season_id = 1 AND episode_id = 1
);

SELECT * FROM episodes WHERE series_id = 1 AND season_id = 1;

UPDATE episodes ON
SELECT * FROM $to_update;

COMMIT;

-- Посмотреть результат:
SELECT * FROM episodes WHERE series_id = 1 AND season_id = 1;
```

Удаление данных

Удалите данные из таблицы с помощью оператора DELETE.

> **Примечание**
>
> Предполагается, что вы уже создали таблицы ранее на шаге Создание таблиц и заполнили их данными на шаге Добавление данных в таблицы.

```
DELETE
FROM episodes
WHERE
    series_id = 2
    AND season_id = 5
    AND episode_id = 12
;

-- Посмотреть результат:
SELECT * FROM episodes WHERE series_id = 2 AND season_id = 5;

-- YDB не знает об изменениях, имевших место в начале транзакции,
-- поэтому сначала выполняет чтение. Невозможно выполнить UPDATE или DELETE,
-- если таблица уже была изменена в рамках текущей транзакции. UPDATE ON и
-- DELETE ON позволяют читать, обновлять и удалять несколько строк из одной таблицы
-- в рамках одной транзакции.

$to_delete = (
    SELECT series_id, season_id, episode_id
    FROM episodes
    WHERE series_id = 1 AND season_id = 1 AND episode_id = 2
);

SELECT * FROM episodes WHERE series_id = 1 AND season_id = 1;

DELETE FROM episodes ON
SELECT * FROM $to_delete;

-- Посмотреть результат:
SELECT * FROM episodes WHERE series_id = 1 AND season_id = 1;
```

Добавление и удаление колонок

Добавьте новую колонку в таблицу, а затем удалите ее.

> **Примечание**
>
> Предполагается, что вы уже создали таблицы ранее на шаге Создание таблиц и заполнили их данными на шаге Добавление данных в таблицы.

Добавить колонку

Добавьте неключевую колонку в существующую таблицу:

```
ALTER TABLE episodes ADD COLUMN viewers Uint64;
```

Удалить колонку

Удалите добавленную колонку из таблицы:

```
ALTER TABLE episodes DROP COLUMN viewers;
```

Удаление таблиц

Удалите созданные таблицы с помощью конструкции DROP TABLE.

```
DROP TABLE episodes;
DROP TABLE seasons;
DROP TABLE series;
```

Выбор первичного ключа

Рекомендации по выбору правильного первичного ключа для таблицы зависят от ее типа:

- Строковые таблицы
- Колоночные таблицы

Выбор первичного ключа для максимальной производительности

Выбор колонок для первичного ключа строковой таблицы оказывает определяющее влияние на возможности YDB по масштабированию нагрузки и повышению производительности.

Общие рекомендации по выбору первичного ключа:

- Следует избегать ситуаций, когда основная часть нагрузки приходится на одну партицию строковой таблицы. Чем равномернее нагрузка распределяется по партициям, тем большая производительность может быть достигнута.
- Следует уменьшать количество партиций, которые могут быть затронуты в одном запросе. Более того, если запрос затрагивает не более одной партиции, то он выполняется по специальному упрощенному протоколу, что существенно увеличивает скорость и экономит ресурсы.

Все строковые таблицы в YDB отсортированы по возрастанию первичного ключа. Это означает, что запись в строковую таблицу данных с монотонно возрастающим первичным ключом приведет к добавлению новых данных в конец таблицы. Так как YDB разделяет строковые таблицы на партиции по диапазонам ключей, вставки будут обрабатываться одним конкретным сервером, отвечающим за "последнюю" партицию. Сосредоточение нагрузки на одном сервере приведет к медленной загрузке данных и неэффективному использованию распределенной системы.

То же поведение возникает, если монотонно возрастающее значение является **единственной** компонентой первичного ключа или **первой компонентой составного ключа**: диапазоны партиций задаются по префиксу ключа, поэтому монотонность «в начале» ключа даёт ту же «горячую точку». Автоинкрементные типы (`SERIAL`), будучи единственной или первой компонентой `PRIMARY KEY`, ведут себя так же.

В качестве примера рассмотрим запись лога пользовательских событий в строковую таблицу со схемой `( timestamp, userid, userevent, PRIMARY KEY (timestamp, userid) )`.

Значения колонки `timestamp` монотонно возрастают; как следствие, все новые записи будут добавляться в конец строковой таблицы, и "последняя" партиция, которая отвечает за данный диапазон ключей, будет обслуживать все операции вставки в таблицу. Это приведет к невозможности масштабирования нагрузки на вставку, производительность будет ограничена одним процессом обслуживания этой партиции, и не будет расти с добавлением серверов в кластер.

В YDB поддерживается автоматическое разделение партиции при достижении порогового размера или нагрузки. Но в рассматриваемом случае после разделения новая партиция начнет опять принимать всю нагрузку на вставку, и ситуация повторится.

Приемы, позволяющие равномерно распределить нагрузку по партициям строковой таблицы

Изменение порядка следования компонент ключа

Запись данных в строковую таблицу со схемой `( timestamp, userid, userevent, PRIMARY KEY (timestamp, userid) )` приводит к неравномерной нагрузке на партиции таблицы из-за монотонно возрастающего первичного ключа. Изменение порядка следования компонент ключа таким образом, чтобы монотонно возрастающая часть не была первой компонентой, может помочь более равномерно распределить нагрузку. Если изменить определение первичного ключа строковой таблицы на `PRIMARY KEY (userid, timestamp)`, то при достаточном количестве пользователей, генерирующих события, запись в БД будет распределена по партициям более равномерно.

Использование хеша от значений ключевых колонок в качестве первичного ключа

Для получения более равномерного распределения операций между партициями строковой таблицы, а также уменьшения размеров внутренних структур данных, необходимо увеличить разнообразие значений "префикса" (начальной части) первичного ключа. Для этого можно включить в первичный ключ значение хеш-кода от всего первичного ключа или его части.

Например, в рассматриваемую строковую таблицу со схемой `( timestamp, userid, userevent, PRIMARY KEY (userid, timestamp) )` можно включить дополнительное поле, рассчитываемое как хеш-код: `userhash = HASH(userid)`. В результате схема таблицы преобразуется к следующему виду:

```
( userhash, userid, timestamp, userevent, PRIMARY KEY (userhash, userid, timestamp) )
```

При правильном выборе функции хеширования строки будут распределены достаточно равномерно по всему пространству ключей, что приведет к более равномерной нагрузке на систему. При этом наличие полей `userid, timestamp` в составе ключа после поля `userhash` сохраняет локальность и сортировку данных по времени для конкретного пользователя.

Расчёт значения поля `userhash` в описанном выше примере должен осуществляться приложением, и явно указываться как при вставке новых записей таблицы, так и при доступе к записям по первичному ключу.

Один из распространённых частных случаев того же принципа — монотонно возрастающие идентификаторы (`SERIAL`, sequence или последовательный числовой id), а не только временная метка из примера с логом выше.

Монотонно возрастающий идентификатор (SERIAL, sequence, auto increment)

Не рекомендуется использовать монотонно возрастающее значение в качестве единственной или первой компоненты первичного ключа. Типичный пример — автоинкрементный идентификатор (`SERIAL`), значение из sequence или последовательный числовой id.

Строковые таблицы в YDB партиционируются по диапазонам значений первичного ключа, поэтому новые строки с монотонным ключом последовательно попадают в последнюю партицию и создают «горячую точку»: основная нагрузка на запись концентрируется на ограниченном числе партиций, остальные используются неравномерно.

Та же проблема возникает для составного ключа, если его первой компонентой является монотонно возрастающее значение, например `PRIMARY KEY (order_id, customer_id)`.

YDB поддерживает автоинкрементные типы (`SERIAL`), однако использование их в качестве единственной или первой компоненты первичного ключа не рекомендуется для высоконагруженных сценариев вставки.

По возможности замените монотонный ключ на случайный идентификатор (например, UUID) или на составной ключ из бизнес-полей без монотонного префикса. Если монотонно возрастающий идентификатор необходимо сохранить в ключе, используйте составной первичный ключ, где первая компонента — хеш от значения идентификатора, а вторая — сам идентификатор. Это позволяет добиться **более равномерного** распределения записей по партициям и сохранить уникальность (идеальный баланс партиций хеш не гарантирует).

Плохой пример

```
CREATE TABLE orders (
    order_id Uint64,
    customer_id Uint64,
    amount Double,
    PRIMARY KEY (order_id)
);

INSERT INTO orders (order_id, customer_id, amount) VALUES
    (1001, 10, 999.99),
    (1002, 11, 1499.00),
    (1003, 12, 499.50);

SELECT *
FROM orders
WHERE order_id = 1001;
```

Хороший пример

```
CREATE TABLE orders (
    order_hash Uint64,
    order_id Uint64,
    customer_id Uint64,
    amount Double,
    PRIMARY KEY (order_hash, order_id)
);

INSERT INTO orders (order_hash, order_id, customer_id, amount) VALUES
    (Digest::NumericHash(1001), 1001, 10, 999.99),
    (Digest::NumericHash(1002), 1002, 11, 1499.00),
    (Digest::NumericHash(1003), 1003, 12, 499.50);

SELECT *
FROM orders
WHERE order_hash = Digest::NumericHash(1001)
    AND order_id = 1001;
```

Значение поля `order_hash` нужно вычислять при записи и при точечном чтении по ключу — так же, как для `userhash` в примере выше.

Уменьшение количества партиций, затрагиваемых в одном запросе

Предположим, что основной сценарий работы с данными строковой таблицы — прочитать все события по конкретному `userid`. Тогда при использовании схемы строковой таблицы `( timestamp, userid, userevent, PRIMARY KEY (timestamp, userid) )` каждое чтение будет затрагивать все партиции таблицы. При этом, каждая партиция будет просканирована полностью, так как строки, относящиеся к конкретному `userid`, расположены в заранее неизвестном порядке. Изменение порядка следования компонент ключа `( timestamp, userid, userevent, PRIMARY KEY (userid, timestamp) )` приведет к тому, что все строки, относящиеся к конкретному `userid`, будут следовать друг за другом. Такое расположение строк положительно повлияет на скорость чтения информации по конкретному `userid`, и сократит нагрузку.

Значение NULL в ключевой колонке

В YDB все колонки, включая ключевые, могут содержать значение NULL. Использование NULL в качестве значений в ключевых колонках не рекомендуется. По стандарту языка SQL (ISO/IEC 9075) значение NULL нельзя сравнивать с другими значениями, вследствие чего использование лаконичных конструкции SQL с простыми операторами сравнения может приводить, например, к тому что, строки, содержащие значение NULL, могут быть пропущены при фильтрации.

Ограничение размера строки

Для достижения хорошей производительности не рекомендуется записывать в БД строки размером более 8 МБ и ключевые колонки размером более 2 КБ.

Выбор ключей для максимальной производительности колоночных таблиц

В отличие от строковых таблиц YDB, колоночные таблицы партиционируют данные не по первичным ключам, а по специально выделенным ключам — **ключам партицирования**. При этом внутри каждой партиции данные распределяются в порядке первичного ключа таблицы.

Ключ партиционирования

Ключ партиционирования должен являться непустым подмножеством столбцов первичного ключа. Хэш от ключа партиционирования определяет партицию, в которую попадёт строка. Ключ следует выбирать таким образом, чтобы данные распределялись равномерно по партициям. Обычно этого достигают путём включения в ключ партиционирования высококардинальных столбцов, например, временных меток с высоким разрешением (тип данных `Timestamp`). Если в качестве ключа партиционирования используется низкокардинальный ключ, данные могут распределиться по партициям неравномерно, что приведёт к перегрузке некоторых партиций. Перегрузка партиций вызывает неоптимальную производительность запросов и/или накладывает ограничения на максимальный поток вставляемых данных.

В настоящий момент колоночные таблицы не поддерживают автоматического репартицирования, поэтому важно указывать правильное число партиций при создании таблицы.

Для базовой оценки числа партиций можно использовать формулу `(количество узлов * 4`. Это позволит максимально утилизировать ресурсы кластера при выполнении параллельных запросов.

Первичный ключ

Первичный ключ определяет порядок хранения данных внутри партиции, поэтому при выборе первичного ключа необходимо учитывать не только эффективность сценария чтения данных из партиции, но и эффективность вставки данных в партицию. Оптимальный сценарий вставки — запись данных в начало или в конец таблицы с редкими локальными обновлениями уже вставленных данных. Например, эффективен сценарий хранения журналов работы приложений, когда данные хранятся по временным меткам, а новые записи добавляются в конец партиции за счет использования текущего времени в первичном ключе.

Пример

Рассмотрим следующий сценарий: необходимо спроектировать таблицу для хранения и анализа данных из высоконагруженной рекламной сети. Система генерирует события (показы, клики).

Основные сущности:

- `event_timestamp`: Время события (точность до микросекунд).
- `user_id` : Идентификатор пользователя.
- `campaign_id` : Идентификатор рекламной кампании.
- `ad_id` : Идентификатор конкретного рекламного объявления.
- `event_type` : Тип события (`'impression'` , `'click'`).
- `event_id` : Уникальный идентификатор события.

Типичные аналитические запросы:

1. Посчитать количество кликов по кампании `campaign_id = X` за последний час.
2. Найти все события для пользователя `user_id = Y` , отсортированные по времени.

Выбор ключа партиционирования

Для обеспечения максимальной производительности необходимо равномерно распределить входящие данные по всем партициям.
Рассмотрим использование различных колонок в качестве ключа партиционирования.

Неудовлетворительные варианты:

- `event_timestamp` или `event_date` . Все поступающие данные за текущий момент времени будут отправляться в одну и ту же партицию, создавая высокую нагрузку на запись в одну партицию на одном узле, в то время как остальные партиции будут простаивать.
- `campaign_id` . Рекламные кампании имеют разную популярность. Одна крупная кампания может генерировать 80% всех событий, что приведет к перекосу данных и записи преимущественно в несколько партиций.

Оптимальные варианты:

- `user_id` . Идентификаторов пользователей очень много (высокая кардинальность), их активность, скорее всего, распределена случайным образом.
- композитный ключ, например, (`event_timestamp` , `user_id`). Для вычисления партиции данные колонок конкатенируются, и хеш-функция вычисляется уже по итоговой суммарной строке, обеспечивая уникальное значение.

Итог: оптимальным вариантом ключа партиционирования является либо колонка `user_id` , либо композитный ключ вида (`event_timestamp` , `user_id`).
Например, `PARTITION BY HASH(event_timestamp, user_id)` .

Выбор первичного ключа

Выбор первичного ключа важен для следующих задач:

1. Обеспечение быстрой фильтрации по диапазону (особенно по времени).
2. Оптимизация записи, делая её максимально приближённой к последовательному добавлению в конец (append-only).
3. Обеспечение целостности данных.

Вернёмся к требованиям к системе:

1. Посчитать количество кликов по кампании `campaign_id = X` за последний час.
2. Найти все события для пользователя `user_id = Y` , отсортированные по времени.

Рассмотрим варианты выбора первичного ключа:

1. Для обеспечения выборок, связанных со временем, и для записи поступающих данных в конец таблиц, оптимально использовать `event_timestamp` как первую часть ключа.

2. В один и тот же момент времени могут поступать данные о действиях разных пользователей, поэтому в качестве второй части ключа используем `user_id`.
3. Пользователи могут совершать действия в разных кампаниях, поэтому в третью часть ключа добавим поле `campaign_id`.

Итоговая структура таблицы

```
CREATE TABLE ad_events (
    -- Ключевые колонки, участвующие в ключах
    user_id Utf8 NOT NULL,
    event_timestamp Timestamp NOT NULL,
    event_id Uint64,
    campaign_id Uint64 NOT NULL,

    -- Остальные колонки
    ad_id Uint64,
    event_type Utf8,
    PRIMARY KEY(event_timestamp, user_id, campaign_id)
)
PARTITION BY HASH(user_id, event_timestamp) -- Равномерно распределяем данные
WITH (
    STORE = COLUMN
);
```

Такая структура эффективно решает поставленные задачи: входящий поток данных равномерно распределяется по партициям благодаря `HASH(user_id, event_timestamp)`, а аналитические запросы по временным диапазонам работают быстро за счёт сортировки данных по `event_timestamp` внутри каждой партиции.

Партицирование

В этом подразделе собраны материалы по партицированию таблиц в приложениях:

- Строковые таблицы
- Колоночные таблицы

См. также:

- Выбор первичного ключа
- Вторичные индексы
- Таблица

Как работает партицирование

Строковая таблица в YDB может быть разбита на **партиции** — непересекающиеся диапазоны значений первичного ключа. Каждая партиция обслуживается отдельной таблеткой data shard; партиции могут находиться на разных узлах кластера и переноситься между ними при балансировке и отказах.

Подробное описание модели, параметров `WITH (...)` и поведения при росте данных и нагрузки приведено в статье Таблица — разделы Партиционирование строковой таблицы и перечень параметров `AUTO_PARTITIONING_*`.

Дальше в этом подразделе:

- Автоматическое партицирование — режимы **по объёму** и **по нагрузке**, а также ограничения автоматического партицирования;
- Как выбрать количество партиций — выбор минимального и максимального числа партиций;
- Анти-паттерны при партицировании — типичные ошибки схемы и конфигурации.

> **Примечание**
>
> Колоночные таблицы партицируются иначе и **не поддерживают** автоматического репартицирования по тем же правилам, что строковые. См. Партицирование колоночных таблиц.

Как работает партицирование

Строковая таблица в YDB может быть разбита на **партиции** — непересекающиеся диапазоны значений первичного ключа. Каждая партиция обслуживается отдельной таблеткой data shard; партиции могут находиться на разных узлах кластера и переноситься между ними при балансировке и отказах.

Подробное описание модели, параметров `WITH (...)` и поведения при росте данных и нагрузки приведено в статье Таблица — разделы Партиционирование строковой таблицы и перечень параметров `AUTO_PARTITIONING_*`.

Дальше в этом подразделе:

- Автоматическое партицирование — режимы **по объёму** и **по нагрузке**, а также ограничения автоматического партицирования;
- Как выбрать количество партиций — выбор минимального и максимального числа партиций;
- Анти-паттерны при партицировании — типичные ошибки схемы и конфигурации.

> **Примечание**
>
> Колоночные таблицы партицируются иначе и **не поддерживают** автоматического репартицирования по тем же правилам, что строковые. См. Партицирование колоночных таблиц.

Автоматическое партицирование

Для строковых таблиц YDB может автоматически **разделять** (split) и **объединять** (merge) партиции, чтобы подстроиться под объём данных и интенсивность запросов. Это задаётся параметрами `AUTO_PARTITIONING_*` в описании таблицы (см. Партиционирование строковой таблицы).

Типичное время операции разделения или слияния — порядка **500 мс**; в этот интервал данные по затронутой партиции могут быть кратковременно недоступны для чтения и записи (подробности — в Партиционирование строковой таблицы).

Ограничения автоматического партицирования (лимиты базы и таблицы, очередь split/merge) описаны ниже в разделе Ограничения автошардирования.

По объёму (size-based)

Режим `AUTO_PARTITIONING_BY_SIZE` включает разделение партиции, когда её размер превышает заданный порог `AUTO_PARTITIONING_PARTITION_SIZE_MB`, и объединение соседних партиций, когда суммарный размер достаточно мал (см. точные условия в описании параметров в `AUTO_PARTITIONING_BY_SIZE`).

Практические ориентиры:

- Рекомендуемый диапазон `AUTO_PARTITIONING_PARTITION_SIZE_MB` в документации по умолчанию ориентирован на значения порядка **от десятков мегабайт до 2000 MB**; конкретное значение зависит от профиля нагрузки и допустимой частоты split/merge.
- Слишком **большой** порог при неравномерном доступе к ключам приводит к «тяжёлым» партициям и сильнее нагружает один data shard; слишком **маленький** — к частым split/merge.

Независимо от пользовательского порога, в продукте также участвует внутренняя логика, завязанная на ориентир **~2000 MB** для части решений о разделении — см. Партиционирование строковой таблицы.

При изменении порога на **уже большой таблице** избегайте резкого скачка (например, с 2000 MB до 100 MB сразу). Хотя количество одновременных операций split ограничено, резкое изменение создаст **длинную очередь** из задач на разделение. Все операции из этой очереди будут выполняться последовательно с ограниченным параллелизмом, что может затянуть процесс стабилизации таблицы. Рекомендуем снижать порог постепенно и мониторить состояние; см. также Избыточные разделения и слияния партиций таблиц. Подробнее о лимитах на число одновременных операций — в Ограничениях автошардирования.

По нагрузке (load-based)

Режим `AUTO_PARTITIONING_BY_LOAD` позволяет разделять партицию, когда она **перегружена по CPU**, даже если размер партиции ещё не большой. По умолчанию для строковых таблиц этот режим **выключен** (`DISABLED`); для высокой конкурентной записи/чтения по узкому диапазону ключей его обычно имеет смысл **включить** вместе с `AUTO_PARTITIONING_MIN_PARTITIONS_COUNT`, выставленным исходя из ожидаемого числа партиций, чтобы избежать их излишнего сжатия при спаде нагрузки.

Каноническое описание порогов и алгоритма (сэмпл ключей, учёт реплик, merge при низкой загрузке) — в `AUTO_PARTITIONING_BY_LOAD`.

Типичные симптомы перегруженной партиции

- высокая загрузка CPU на отдельных партициях при умеренной общей загрузке кластера;
- рост задержек на «горячих» ключах;
- рост числа внутренних ответов `STATUS_OVERLOADED`. Начиная с версии 26.2, Query Processor автоматически повторяет такие запросы, поэтому пользователь обычно наблюдает увеличение задержки, а не саму ошибку.

Один data shard использует **не более одного ядра CPU** для операций изменения и чтения данных на партиции: простое добавление CPU на узел не снимает bottleneck с одной партиции — нужны split и/или ключ, лучше способствующий распределению нагрузки по шардам.

Ограничения автошардирования

«Автошардирование» здесь означает автоматические **split/merge** партиций строковой таблицы. На практику накладываются лимиты уровня **базы данных** и **таблицы**, а также внутренние ограничения алгоритма.

Лимиты уровня базы данных

- **Максимальное число таблеток в базе** (`MaxShards`) — при достижении лимита автоматическое разделение партиций прекращается. См. Ограничения объектов схемы.
- **Максимальное число шардов таблицы** (`MaxShardsInPath`) — верхняя граница числа партиций (data shard) на одном пути таблицы, включая индексы.
- **Жёсткий лимит размера партиции на уровне кластера** — по умолчанию **2 ГиБ** (`ForceShardSplitDataSize`). Если размер партиции превышает этот порог, она разделится, даже если пользовательский `AUTO_PARTITIONING_PARTITION_SIZE_MB` выше или достигнут `AUTO_PARTITIONING_MAX_PARTITIONS_COUNT`. Подробнее — в Партиционирование строковой таблицы.

Параметры таблицы

`AUTO_PARTITIONING_MIN_PARTITIONS_COUNT` задаёт нижнюю границу для автоматического объединения партиций, а `AUTO_PARTITIONING_MAX_PARTITIONS_COUNT` — верхнюю границу для обычного автоматического разделения. При достижении максимума разделение по размеру или нагрузке останавливается, но превышение жёсткого лимита размера 2 ГиБ всё равно запускает разделение.

Рекомендуемая разница между min и max: если разница между минимумом и максимумом **больше ~20%**, при сильно меняющейся во времени нагрузке компонента Hive возможны лишние циклы split/merge. Подробнее и что делать — в Избыточные разделения и слияния партиций таблиц.

Скорость адаптации по нагрузке

При включённом `AUTO_PARTITIONING_BY_LOAD`:

- минимальная продолжительность сэмплирования нагрузки перед решением о разделении составляет **две минуты**;
- за одну операцию партиция разделяется **не более чем на две** части.

При значительном или многократном росте нагрузки адаптация партицирования может занимать **единицы и десятки минут**. В течение этого времени перегруженные партиции могут возвращать внутренние ответы `OVERLOADED`, которые начиная с версии 26.2 автоматически обрабатываются повторными попытками в Query Processor. Для пользователя это проявляется прежде всего увеличением задержки.

Разделение и объединение партиций включают **компакшен** данных партиции — дополнительную нагрузку на диск.

После завершения сэмплирования обработка решения о разделении таблетки в scheme shard может занимать около **15 секунд** (см. Избыточные разделения и слияния партиций таблиц).

Очередь операций split/merge

Одновременно на **таблице** может выполняться ограниченное число операций split — остальные партиции ждут в очереди. Дополнительно Scheme shard ограничивает общее число параллельных split/merge в базе.

Вторичные индексы

Параметры партиционирования задаются для **таблицы**; индекс имеет свой набор партиций и свои правила по умолчанию — см. Партиционирование строковой таблицы.

> **Важно**
>
> Колоночные таблицы: автоматическое репартицирование по тем же правилам **не поддерживается** — число партиций задаётся при проектировании таблицы. См. Партицирование колоночных таблиц.

Как выбрать количество партиций

Количество партиций влияет на то, сколько таблеток data shard обрабатывают таблицу параллельно и как данные распределяются по узлам.

Рекомендации из модели данных YDB:

- Одна партиция размещается на одном сервере и для операций изменения использует **не более одного ядра CPU**. Поэтому для таблиц с ожидаемой высокой нагрузкой имеет смысл задавать `AUTO_PARTITIONING_MIN_PARTITIONS_COUNT` **не меньше числа узлов** (серверов) базы, а лучше **значительно больше суммарного числа ядер**, выделенных под нагрузку: нагрузка на таблицу почти всегда неравномерна, часть партиций простаивает, и в типичном случае распределение CPU по партициям ближе к нормальному, чем к равномерному.
- `AUTO_PARTITIONING_MAX_PARTITIONS_COUNT` ограничивает верхнюю границу роста числа партиций; при достижении потолка новые split по размеру или нагрузке прекращаются — следите за метриками и при необходимости увеличивайте лимит заранее.*
- При включённом автопартицировании по нагрузке **не оставляйте** минимальное число партиций равным **1**, если нагрузка «плавающая»: после спада нагрузки merge может свести таблицу к одной партиции и потребоваться заново делить при новом всплеске. Подробнее — `AUTO_PARTITIONING_MIN_PARTITIONS_COUNT`.

Применить рекомендации на практике можно инструкцией YQL:

```
ALTER TABLE `my_table` SET (
    AUTO_PARTITIONING_BY_LOAD = ENABLED,
    AUTO_PARTITIONING_MIN_PARTITIONS_COUNT = 10,
    AUTO_PARTITIONING_MAX_PARTITIONS_COUNT = 50
);
```

Начальное равномерное распределение по ключу задаётся опционально через `UNIFORM_PARTITIONS` или `PARTITION_AT_KEYS` — см. Партиционирование строковой таблицы.

---

* Независимо от значения `AUTO_PARTITIONING_PARTITION_SIZE_MB` партиция всегда разделяется при достижении жёсткого лимита размера, заданного на уровне кластера (**2 ГиБ** по умолчанию).

Анти-паттерны при партицировании

Типичные ошибки при проектировании ключа и настроек партиционирования:

Антипаттерны дизайна первичного ключа (монотонные ключи, горячие точки) рассмотрены в Выбор первичного ключа для максимальной производительности.

- **Жёсткий лимит размера партиции на уровне кластера** — по умолчанию **2 ГиБ**. Если размер партиции превышает этот лимит, она разделится независимо от `AUTO_PARTITIONING_MAX_PARTITIONS_COUNT`, если не достигнут лимит числа шардов таблицы в базе.
- **Использование `AUTO_PARTITIONING_MAX_PARTITIONS_COUNT` как жёсткого предела** — при достижении этого значения обычное разделение по размеру или нагрузке прекращается. Однако при превышении жёсткого лимита размера 2 ГиБ партиция всё равно разделяется, поэтому фактическое число партиций может превысить заданный максимум.

Настройки по умолчанию без явной конфигурации

Если при создании строковой таблицы **не задавать** параметры `AUTO_PARTITIONING_*` явно, действуют значения по умолчанию из модели данных (см. Партиционирование строковой таблицы).

- **`AUTO_PARTITIONING_BY_SIZE`** обычно **включён** — значение **`AUTO_PARTITIONING_PARTITION_SIZE_MB`** по умолчанию составляет ровно 2 ГиБ.
- **`AUTO_PARTITIONING_BY_LOAD`** по умолчанию **выключен** — перегрузка партиции по CPU **не приведёт** к автоматическому split, пока режим не включён.
- **`AUTO_PARTITIONING_MIN_PARTITIONS_COUNT`** по умолчанию **1** — при спаде нагрузки merge может оставить одну партицию; при следующем всплеске может потребоваться время на новые split.

При **высокой конкуренции за единичные ключи** (горячий ключ, низкая кардинальность первого компонента) производительность будет ограничена вне зависимости от настроек `AUTO_PARTITIONING_*` — партицирование по нагрузке здесь мало поможет; пересмотрите модель ключа (см. Выбор первичного ключа для максимальной производительности).

Если нагрузка распределена шире, но дефолтов недостаточно, задайте явно:

- `AUTO_PARTITIONING_BY_LOAD = ENABLED`;
- `AUTO_PARTITIONING_MIN_PARTITIONS_COUNT` и согласованный `AUTO_PARTITIONING_MAX_PARTITIONS_COUNT`: **не выставляйте** `AUTO_PARTITIONING_MIN_PARTITIONS_COUNT` меньше **80%** от `AUTO_PARTITIONING_MAX_PARTITIONS_COUNT`;
- для снижения нагрузки на отдельные партиции стоит уменьшить максимальный размер партиции — `AUTO_PARTITIONING_PARTITION_SIZE_MB`.

Так вы избегаете сценария «остались дефолты → одна партиция → перегрузка shard».

Партицирование колоночных таблиц

Информация по колоночным таблицам: Партицирование колоночной таблицы.

Вторичные индексы

Важно

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Индексы — вспомогательные структуры в базах данных, позволяющие находить подходящие под определенный критерий данные без необходимости их полного перебора, а также получать отсортированные выборки без выполнения фактической сортировки, требующей обработки полного набора всех сортируемых данных.

Данные в строковых таблицах YDB всегда проиндексированы по первичному ключу. Это означает, что получение любой записи из строковой таблицы с заданными значениями полей, составляющих первичный ключ, всегда будет занимать минимальное фиксированное время, вне зависимости от общего количества записей в строковой таблице. Также наличие индекса по первичному ключу позволяет получить любой последовательный диапазон записей по возрастанию или убыванию первичного ключа. Время исполнения такой операции будет зависеть только от количества получаемых записей, но не зависеть от общего количества записей в строковой таблице. Рекомендации по проектированию первичного ключа см. в разделе Выбор первичного ключа для максимальной производительности.

Для того, чтобы воспользоваться аналогичными возможностями для любых полей или комбинаций полей строковой таблицы, по ним могут быть построены дополнительные индексы, называемые **вторичными индексами**.

В транзакционных системах использование индексов позволяет сократить или исключить деградацию производительности и повышение стоимости выполнения запросов при росте объема хранимых данных.

В данной статье описаны основные операции по работе со вторичными индексами и даны ссылки на подробные материалы по каждой операции. Информация о разных типах вторичных индексов и особенностях их устройства находится в статье Вторичные индексы раздела «Концепции».

Создание вторичных индексов

Вторичный индекс является объектом схемы данных и может быть определен при создании таблицы командой YQL `CREATE TABLE`, или добавлен к ней позднее командой YQL `ALTER TABLE`.

Команда создания индекса `table index add` поддерживается в YDB CLI.

Без вторичного индекса запрос с условием только по неключевой колонке, как правило, выполняет полное сканирование таблицы (`FullScan`), при котором читается каждая строка. Например, рассмотрим таблицу:

```
CREATE TABLE users (
    id Uint64,
    email Utf8,
    name Utf8,
    PRIMARY KEY (id)
);
```

запрос:

```
DECLARE $email AS Utf8;
SELECT * FROM users WHERE email = $email;
```

вынужден читать все строки таблицы, так как `email` не входит в первичный ключ. Добавление индекса позволяет выполнять выборку по `email` без полного сканирования таблицы:

```
ALTER TABLE users ADD INDEX idx_email GLOBAL ON (email);
```

Так как индекс содержит собственные данные, являющиеся производными от данных в строковой таблице, то при создании индекса на существующей таблице с данными будет выполнена операция начального построения индекса, которая может занимать продолжительное время. Данная операция выполняется в фоновом режиме, не блокирует работу с таблицей, но до окончания построения новым индексом воспользоваться будет невозможно.

Использование индекса возможно только в порядке включенных в него полей. Если в индексе два поля `a` и `b`, то такой индекс может быть эффективно использован для запросов вида:

- `WHERE a = $var1 AND b = $var2`;
- `WHERE a = $var1`;
- `WHERE a > $var1`, а также другие операторы сравнения;
- `WHERE a = $var1 AND b > $var2`, а также любые другие операторы сравнения, но первое поле должно проверяться на равенство.

При этом, такой индекс не может быть применен для следующих запросов:

- `WHERE b = $var1`;
- `WHERE a > $var1 AND b > $var2`, точнее эта запись будет равнозначна `WHERE a > $var1` с точки зрения применения индекса;
- `WHERE b > $var1`.

Для эффективного ограничения результата запроса с помощью команды `LIMIT` необходимо использовать однонаправленную сортировку в команде `ORDER BY`:

- `WHERE a = $var1 AND b > $var2 ORDER BY a, b DESC LIMIT 1`, будут прочитаны все строки индекса, подходящие под условия фильтра;
- `WHERE a = $var1 AND b > $var2 ORDER BY a DESC, b DESC LIMIT 1`, будет прочитана всего одна строка.

С учетом описанной выше особенности, бесполезно пытаться заранее индексировать все возможные комбинации колонок в таблице в расчете на быстрое исполнение любых запросов. Индекс - это всегда компромисс между скоростью поиска и

записи, а также занимаемым данными местом в хранилище. Индексы создаются под конкретные выборки и условия поиска, которые будут делаться приложением в базе данных.

Применение вторичных индексов при выборке данных

При выборке данных из строковой таблицы YDB предоставляет два способа использования вторичных индексов:

1. **Явное указание индекса** — имя вторичного индекса указывается в секции `VIEW` после имени таблицы, как описано в статье про команду `SELECT` YQL. Например, для получения из строковой таблицы Заказов (`orders`) выборки заказов клиента с заданным ID (`id_customer`) запрос будет выглядеть следующим образом:

   ```
   DECLARE $customer_id AS Uint64;
   SELECT *
   FROM   orders VIEW idx_customer AS o
   WHERE  o.id_customer = $customer_id
   ```

   , где `idx_customer` — имя вторичного индекса на строковой таблице `orders`, первым в котором указано поле `id_customer`.
2. **Автоматический выбор индекса оптимизатором запросов** — если секция `VIEW` не указана, оптимизатор может самостоятельно принять решение об использовании того или иного вторичного индекса на основе текста запроса. Подробное описание критериев выбора приведено ниже.

> **Важно**
>
> Если у вас существуют запросы без явного указания индекса (без секции `VIEW`), то при добавлении нового индекса к таблице такие запросы могут начать использовать его автоматически. Это может привести к неожиданному изменению плана и времени выполнения запроса. Чтобы избежать подобных неожиданностей, рекомендуется:
>
> - Явно указывать в запросах, какой индекс должен использоваться, с помощью секции `VIEW`;
> - На основе критериев выбора вторичного индекса заранее представлять, может ли добавление нового вторичного индекса повлиять на существующие запросы.

Автоматическое использование индексов при выборке

Явное указание секции `VIEW` имеет приоритет над решением оптимизатора о использовании вторичных индексов. То есть запрос

```
SELECT * FROM `Table` VIEW Index
```

гарантированно будет производить выборку с использованием индекса `Index`.

Для явного указания чтения с использованием первичного ключа следует использовать следующую конструкцию:

```
SELECT * FROM `Table` VIEW PRIMARY KEY
```

Критерии выбора вторичного индекса

Выбор индекса используемого для чтения происходит во время оптимизации запроса при определении диапазонов строк, которые необходимо прочитать (predicate pushdown). Индексы, как и основная таблица представляют из себя набор строк упорядоченных по набору ключевых колонок.

Выбор между чтениями с использованием индекса и с использованием первичного ключа происходит на основе следующих метрик:

1. Необходимость дополнительного чтения из основной таблицы. Если в индексе присутствуют все нужные для запроса колонки, то дополнительные чтения не требуются.
2. Длина точечного префикса предиката для ключа соответствующей таблицы. То есть, предикат ограничивает некоторый набор колонок являющихся первыми компонентами ключа точечными условиями: `=`, `IN`, `IS NULL`. Здесь приоритет отдается индексам, для которых зафиксированы все индексируемые колонки или основной таблице если точечным является первичный ключ целиком.
3. Количество использованных колонок в границах диапазона для чтения. В следующем запросе к таблице Table с первичным ключом (Key1, Key2, Key3)

```
SELECT * FROM `Table` WHERE (Key1, Key2, Key3) < ($param1, $param2, $param3) AND (Key1, Key2) > ($param4, $pa
ram5)
```

чтение будет производиться в диапазоне `(($param4, $param5), ($param1, $param2, $param3))` и таким образом количество использованных колонок будет равно 3. Аналогично 2 здесь отдается предпочтение индексам для которых использованы все индексируемые колонки.

Способы чтения ранжируются между собой в соответствии с критерием 2, при равенстве с критерием 3 и дополнительно учитывается критерий 1.

Примеры автоматического выбора индексов

```
CREATE TABLE `Table` (
     Key Int32,
     SubKey1 Int32,
     SubKey2 String,
     Value1 String,
     Value2 String,
     PRIMARY KEY (Key, SubKey1, SubKey2),
     INDEX Index12 GLOBAL ON (SubKey1, SubKey2),
     INDEX Index21 GLOBAL ON (SubKey2, Value1),
```

```
    INDEX Index212 GLOBAL ON (SubKey2) COVER (Value2)
);
```

`SELECT * FROM Table WHERE SubKey1 = $p1 and SubKey2 > $p2` — будет использован `Index12`. Выражение для диапазона — `(($p1; $p2), ($p1)]`. Длина точечного префикса для `Index12` — 1, для остальных индексов — 0.

`SELECT * FROM Table WHERE Key = $p1 and SubKey1 = $p2 And SubKey2 = $p2` — индекс не будет использоваться. При выборе скана основной таблицы используется все 3 колонки `[Key, Fk1, Fk2]`, длина точечного префикса 3.`

`SELECT * FROM Table WHERE Key = $p1 and SubKey2 = $p2` — вторичные индексы не будут использоваться. При выборе любого вторичного индекса используется 1 колонка, точечный префикс также не более 1 при любом варианте выбора индекса.

`SELECT * FROM Table WHERE Key >= $p1 and SubKey1 = $p2 And SubKey2 = $p3` — должен быть выбран Index12, так как при его выборе в получающемся диапазоне `[[Fk1; Fk2; Key], [Fk1; Fk2])` получится длина точечного префикса — 2, и будут использоваться 3 колонки.

`SELECT * FROM Table WHERE Key = 2 and SubKey2 = 3` — вторичные индексы не должны быть использованы. В случае чтения по `PK` и при использовании любого из вторичных индексов точечный префикс состоит не более чем из одной колонки. Также используется не более одной колонки.

`SELECT * FROM Table WHERE SubKey1 > 2` — Должен быть выбран `Index12`. Только при использовании `Index12` будет нетривиальный диапазон для чтения.

`SELECT * FROM Table WHERE SubKey2 = 2` — Может быть выбран любой из `Index21` и `Index212`. При использовании вышеупомянутых индексов длина точечного префикса будет 1. Количество использованных колонок также максимизируется при выборе `Index21` и `Index212`

`SELECT Value2 FROM Table WHERE SubKey2 = 2` — Должен быть выбран Index212. При использовании Index21 и Index212 длина точечного префикса будет 1, но при использовании Index212 не нужно чтение основной таблицы.

Этот пример показывает общее правило: индекс избавляет от дополнительного чтения основной таблицы, если в нём есть все колонки, необходимые запросу. Глобальный вторичный индекс содержит свои ключевые колонки, колонки из `COVER` и неявно сохранённые колонки первичного ключа основной таблицы (см. Вторичные индексы). Если запрос выбирает другие колонки (например, `SELECT * FROM Table WHERE SubKey2 = 2`), их потребуется дополнительно прочитать из основной таблицы для найденных строк.

`SELECT * FROM Table WHERE SubKey2 > 2` — Будут использованы `Index21` или `Index212`, так как читаемый диапазон нетривиален только при их использовании.

`SELECT * FROM Table WHERE SubKey1 = 2` — Будет использован `Index12`, так как при его использовании длина точечного префикса будет 1, а в остальных случаях 0.

Проверка стоимости запроса

Любой запрос в транзакционном приложении необходимо проверять с точки зрения того, сколько он выполнил операций ввода-вывода в базе данных и сколько CPU было потрачено на его исполнение. Также необходимо убедиться, что эти цифры не растут бесконечно с ростом объема базы данных. В YDB после выполнения каждого запроса возвращается статистика, содержащая необходимую для анализа информацию.

При использовании YDB CLI вывод статистики после исполнения команды `yql` включается опцией `--stats`. Все YDB SDK также содержат структуры, содержащие статистику после исполнения запросов. При исполнении запросов в UI рядом с закладкой результатов также присутствует закладка со статистикой.

Обновление данных с использованием вторичного индекса

Команды YQL изменения записей (`UPDATE`, `UPSERT`, `REPLACE`) не позволяют указать на использование вторичного индекса для поиска данных, поэтому попытка выполнить `UPDATE ... WHERE indexed_field = $value` приведет к полному сканированию строковой таблицы. Чтобы избежать этого, можно предварительно выполнить `SELECT` по индексу с получением значения первичного ключа, а затем выполнить `UPDATE` по первичному ключу. Также можно воспользоваться инструкцией `UPDATE ON`.

Чтобы обновить данные в строковой таблице `table1`, выполните запрос:

```
$to_update = (
    SELECT pk_field, $f1 AS field1, $f2 AS field2, ...
    FROM   table1 VIEW idx_field3
    WHERE  field3 = $f3)

UPDATE table1 ON SELECT * FROM $to_update
```

Примечание

В настоящий момент обновление данных возможно только при использовании синхронного вторичного индекса. Это следствие того, что модификация данных возможна только в Serializable транзакциях, гарантии которых нарушаются при использовании асинхронных индексов.

Удаление данных с использованием вторичного индекса

Для удаления данных по вторичному индексу используется `SELECT` с предикатом по вторичному индексу, а затем вызывается инструкция `DELETE ON`.

Чтобы удалить все данные о сериалах с нулевым количеством просмотров в строковой таблице `series`, выполните запрос:

```
DELETE FROM series ON
SELECT series_id
FROM series VIEW views_index
WHERE views = 0;
```

Примечание

В настоящий момент удаление данных возможно только при использовании синхронного вторичного индекса. Это следствие того, что удаление возможно только в Serializable транзакциях, гарантии которых нарушаются при использовании асинхронных индексов.

Атомарная замена вторичного индекса

Существующий вторичный индекс можно заменить атомарно. Это может быть полезно, например, для замены индекса на покрывающий. Для работающих приложений эта операция прозрачна — в момент замены индекса произойдет инвалидация скомпилированных запросов.

Атомарно заменить существующий индекс можно с помощью команды YDB CLI ydb table index rename с параметром `--replace`.

Производительность записи в строковой таблицы со вторичными индексами

Для работы вторичных индексов необходимы дополнительные структуры данных. Поддержка этих структур приводит к повышению стоимости операций изменения данных в строковых таблицах.

При синхронном обновлении индексов транзакция подтверждается только после записи всех необходимых данных, как в строковой таблице, так и в синхронных индексах. Это приводит как к увеличению времени исполнения, так и к необходимости применения распределенных транзакций даже при добавлении или изменении записей в единственной партиции.

Асинхронно обновляемые индексы сохраняют возможность применения одношардовых транзакций, однако гарантируют только консистентность «в конечном счете», и все равно создают нагрузку на базу данных.

Векторные индексы

Векторные индексы — это специализированный тип вторичного индекса, который позволяет эффективно выполнять векторный поиск в многомерных пространствах. В отличие от традиционных вторичных индексов, оптимизированных для поиска по равенству или диапазону, векторные индексы позволяют выполнять приближенный поиск на основе функций схожести или расстояния.

Данные в таблице YDB хранятся и сортируются по первичному ключу, что обеспечивает эффективный поиск по точному совпадению и сканирование диапазонов. Векторные индексы предоставляют аналогичную эффективность для поиска ближайших соседей в векторных пространствах.

Виды векторных индексов

Векторный индекс может быть глобальным либо глобальным с фильтрацией. Также любой из этих видов индекса может быть покрывающим и включать копию данных дополнительных колонок из основной таблицы.

Глобальный векторный индекс

Глобальный векторный индекс по колонке `embedding` позволяет быстро искать приближённых ближайших соседей по всей таблице:

```
ALTER TABLE my_table
  ADD INDEX my_index
  GLOBAL USING vector_kmeans_tree
  ON (embedding)
  COVER (embedding, data)
  WITH (distance=cosine, vector_type="float", vector_dimension=512, overlap_clusters=3);
```

Пример запроса поиска к такому индексу:

```
PRAGMA ydb.KMeansTreeSearchTopSize = "10";

DECLARE $query_vector AS string;

$query_vector = Knn::ToBinaryStringFloat([1.0, 1.2, 0, ...]);

SELECT user, data
FROM my_table VIEW my_index
ORDER BY Knn::CosineSimilarity(embedding, $query_vector) DESC
LIMIT 10;
```

Обратите внимание, что:

- Как векторная колонка `embedding`, так и параметр `$query_vector` должны иметь строковый тип и содержать массив чисел в простом бинарном формате.
- Передавать параметр из SDK эффективнее в виде строки, сериализуя числа на стороне приложения (примеры). Альтернативно значение можно передавать из SDK как вектор чисел и конвертировать из списка функциями `Knn::ToBinaryString*`, но это медленнее.
- Конструкция `COVER (embedding, data)` необязательна и предназначена для создания покрывающего индекса. Это помогает дополнительно ускорить поиск.
- Поиск по векторному индексу всегда приближённый - его результаты отличаются от поиска полным перебором.
- Увеличение параметра `PRAGMA KMeansTreeSearchTopSize` повышает качество (полноту) поиска, но снижает его скорость. Параметр задаёт число ближайших к запросу сканируемых кластеров индекса. Значение по умолчанию равно 4 с перекрывающимися кластерами (`overlap_clusters > 1`) и 10 без перекрытия.
- Параметр `overlap_clusters=3` сильно повышает качество будущего поиска при индексации, задавая максимальное число листовых кластеров, в которые добавляется каждый вектор, но увеличивает размер индекса.
- Параметры `vector_type` и `vector_dimension` можно не указывать, если таблица не пуста — они будут автоматически определены из содержимого строк.

Векторный индекс с фильтрацией

Векторный индекс с фильтрацией позволяет искать ближайших соседей в рамках каждой категории, заданной уникальным значением дополнительных колонок.

Чтобы создать такой индекс, укажите несколько индексных колонок. Последняя колонка должна быть векторной, остальные (колонки категорий) - любыми:

```
ALTER TABLE my_table
  ADD INDEX my_index
  GLOBAL USING vector_kmeans_tree
  ON (user, embedding)
  COVER (embedding, data)
  WITH (distance=cosine);
```

Запросы поиска к такому индексу с фильтрацией могут включать условия по колонке `user`:

```
PRAGMA ydb.KMeansTreeSearchTopSize = "10";

DECLARE $query_vector AS string;

$query_vector = Knn::ToBinaryStringFloat([1.0, 1.2, 0, ...]);

SELECT user, data
FROM my_table VIEW my_index
WHERE user = 'john'
```

```
ORDER BY Knn::CosineSimilarity(embedding, $query_vector) DESC
LIMIT 10;
```

Для поиска по нескольким категориям используйте `IN` или `OR`. Если фильтруемых колонок несколько, поддерживается также условие по начальной части префикса. См. фильтрацию по префиксу векторного индекса.

Параметры индексации и поиска здесь работают аналогично глобальному индексу. Поскольку разные значения фильтруемых колонок часто содержат очень разное количество векторов, индекс с фильтрацией может дополнительно использовать адаптивное количество кластеров, чтобы выбирать количество кластеров для каждого значения автоматически.

Покрывающий векторный индекс

Покрывающий векторный индекс содержит копию данных дополнительных колонок, чтобы избежать чтения из основной таблицы и еще сильнее ускорить поиск.

Обратите внимание, что по умолчанию индекс не содержит копию векторной колонки (в примере - embedding), поэтому если её не добавить в список покрытых колонок явно, то избежать чтения из основной таблицы не удастся, так как вектора всегда используются для точной сортировки результатов на последнем этапе поиска.

```
ALTER TABLE my_table
  ADD INDEX my_index
  GLOBAL USING vector_kmeans_tree
  ON (embedding)
  COVER (embedding, data)
  WITH (distance=cosine);
```

Функции расстояния

Поддерживаются следующие функции схожести или расстояния:

- `distance=cosine` или `similarity=cosine` - косинусное расстояние, соответствует сортировке `ORDER BY Knn::CosineDistance(...) ASC` либо `ORDER BY Knn::CosineSimilarity(...) DESC`.
- `distance=manhattan` - Манхэттенское расстояние (L1-метрика), соответствует `ORDER BY Knn::ManhattanDistance(...) ASC`.
- `distance=euclidean` - Евклидово расстояние (L2-метрика), соответствует `ORDER BY Knn::EuclideanDistance(...) ASC`.
- `similarity=inner_product` - скалярное произведение, соответствует `ORDER BY Knn::InnerProductSimilarity(...) DESC`.

Полный синтаксис векторных индексов

Создание векторного индекса:

- При создании таблицы: CREATE TABLE.
- Добавление к существующей таблице: ALTER TABLE.

Полный синтаксис запроса к векторному индексу:

- VIEW VECTOR INDEX.

Алгоритм поиска

В текущей реализации доступен один тип индекса: `vector_kmeans_tree`.

Векторный индекс типа `vector_kmeans_tree`

Индекс `vector_kmeans_tree` реализует иерархическую кластеризацию данных. Структура индекса включает:

1. Иерархическая кластеризация:
   - индекс строит несколько уровней k-means кластеров;
   - на каждом уровне векторы распределяются по заданному количеству кластеров в степени уровня;
   - первый уровень кластеризует весь набор данных;
   - последующие уровни рекурсивно кластеризуют содержимое каждого родительского кластера.
2. Процесс поиска:
   - поиск идет рекурсивно от первого уровня к последующим;
   - при выполнении запросов индекс анализирует только наиболее перспективные кластеры;
   - такое усечение пространства поиска позволяет избежать полного перебора всех векторов.
3. Параметры:
   - `levels`: число уровней в дереве, задает глубину поиска (рекомендуется 1-3);
   - `clusters`: количество кластеров в k-means, определяющее ширину поиска (рекомендуется 64-512).
   - `overlap_clusters`: максимальное количество кластеров нижнего уровня, в которые добавляется каждый вектор (рекомендуется 3).

Внутри векторный индекс состоит из скрытых индексных таблиц вида `indexImpl*Table`. В запросах на выборку с использованием векторного индекса эти таблицы отображаются в статистике запросов. Подробнее об устройстве векторного индекса см. в отдельной статье Векторный индекс вида vector_kmeans_tree.

Перекрытие кластеров

Векторный индекс в YDB может добавлять каждый вектор в несколько кластеров с целью улучшения полноты и скорости поиска:

```
ALTER TABLE my_table
  ADD INDEX my_index
  GLOBAL USING vector_kmeans_tree
  ON (embedding)
  WITH (distance=cosine, overlap_clusters=3);
```

В данном примере каждый вектор будет добавлен не в 1, а максимум в 3 ближайших листовых кластера.

Параметр overlap_clusters рекомендуется использовать практически всегда, особенно, для векторных индексов с `levels > 1`, так как он позволяет значительно улучшить полноту поиска даже с небольшими значениями PRAGMA KMeansTreeSearchTopSize (например, 3).

Таким образом, можно снизить параметр PRAGMA и значительно ускорить поиск при сохранении той же полноты.

Партиционирование индексных таблиц

Основная нагруженная таблица векторного индекса - `indexImplLevelTable`, таблица структуры кластеров. Любой запрос поиска по индексу читает эту таблицу, поэтому нагрузка на её партиции может ограничивать производительность выборок.

Чтобы улучшить производительность, можно включить для неё автопартиционирование по нагрузке:

```
ALTER TABLE `my_table/my_index/indexImplLevelTable`
SET AUTO_PARTITIONING_BY_LOAD ENABLED;
```

Или по размеру:

```
ALTER TABLE `my_table/my_index/indexImplLevelTable`
SET AUTO_PARTITIONING_PARTITION_SIZE_MB 100;
```

Аналогичную настройку можно применять и к другим индексным таблицам (`indexImplPostingTable` и `indexImplPrefixTable`), но таблица Level - самая нагруженная и при этом небольшая по размеру, поэтому для неё настройки автопартиционирования наиболее актуальны.

Использование реплик индексных таблиц

Ещё один способ ускорения поиска - использовать реплики таблиц. Для этого нужно:

1. Создать покрывающий индекс, чтобы в запросе поиска участвовали только индексные таблицы.
2. Включить реплики на всех индексных таблицах:

   ```
   ALTER TABLE `my_table/my_index/indexImplLevelTable` SET READ_REPLICAS_SETTINGS 'PER_AZ:3';
   ALTER TABLE `my_table/my_index/indexImplPostingTable` SET READ_REPLICAS_SETTINGS 'PER_AZ:3';
   ```

   И, для индекса с фильтрацией, также:

   ```
   ALTER TABLE `my_table/my_index/indexImplPrefixTable` SET READ_REPLICAS_SETTINGS 'PER_AZ:3';
   ```

3. Использовать режим запроса Stale Read-Only.

Требования к данным и ограничения

Векторная колонка хранит сериализованные векторы в типе `String`. Для получения такого представления используйте функции преобразования Knn. Тип и размерность векторов должны соответствовать индексу. Если при построении встречается непустой вектор несовместимой размерности, построение завершается ошибкой `Vector dimension mismatch`; значения `NULL` и пустые эмбеддинги пропускаются.

Таблицы с векторными индексами в настоящее время не поддерживают TTL. Создание индекса на таблице с включённым TTL и включение TTL на таблице с векторным индексом завершаются ошибкой.

`BulkUpsert` не поддерживает таблицы с синхронными векторными индексами. Загружайте данные через `BulkUpsert` до создания индекса или используйте YQL `INSERT` и `UPSERT` для обновления таблицы с индексом.

Обновление векторных индексов

После завершения построения индекса операции `INSERT`, `UPSERT`, `UPDATE` и `DELETE` обновляют его синхронно с основной таблицей. Векторный поиск в той же транзакции видит выполненные ранее записи. При этом сохраняются следующие ограничения:

Кластеры не пересчитываются при обновлении

При обновлении таблицы с векторным индексом его внутренняя структура — дерево кластеров (групп схожих векторов) — не перестраивается. Новые или изменённые записи лишь распределяются по уже существующим кластерам.

Со временем это может приводить к деградации индекса, которая проявляется двумя способами:

- Снижение полноты — индекс может возвращать меньше релевантных результатов, так как кластеры перестают отражать истинную структуру данных;
- Снижение производительности — если кластеры становятся несбалансированными (например, один кластер содержит слишком много записей), поиск замедляется.

Степень деградации зависит от характера обновлений:

- Если индекс был построен на репрезентативной выборке (например, случайные 50 % данных), а затем добавлены оставшиеся записи, структура остаётся актуальной, и деградация минимальна;
- Если же изначально отсутствовали целые группы схожих векторов, кластеры могут разделять пространство некорректно, и качество поиска может существенно упасть.

Крайним случаем является индекс, созданный на пустой таблице: в этом случае он содержит только один кластер, и все новые записи попадают в него. Поиск по такому индексу эквивалентен полному сканированию всей таблицы.

Чтобы избежать деградации:

- Не создавайте векторный индекс на пустой таблице;
- Если в таблице накопилось много новых данных, перестройте индекс, когда снизились качество или скорость поиска.

Чтобы определить, когда требуется перестроение:

1. Выберите репрезентативный набор векторов запросов. Оцените полноту, сравнив результаты поиска по индексу с точными результатами полного сканирования той же таблицы. Пример такого сравнения с параметром `--recall` есть в команде векторной нагрузки.
2. Измерьте время поиска для тех же запросов. Повторяйте измерения с одинаковой функцией расстояния и настройками поиска, включая `KMeansTreeSearchTopSize`.
3. Перестройте индекс, если после изменения распределения данных полнота устойчиво снизилась или время поиска выросло. Сам по себе рост числа строк служит поводом для измерения, а не задаёт порог перестроения.

Неконсистентность при обновлении во время построения

Векторные индексы не поддерживают консистентные обновления во время построения. То есть, векторный индекс не обновляется, если данные в основной таблице меняются до завершения построения индекса.

Это означает, что если вы хотите, чтобы векторный индекс оставался на 100% консистентным, вы должны приостановить обновления данных в таблице во время его построения.

Обновления таблицы не блокируются автоматически, так как поиск по векторному индексу всегда приблизительный и поэтому отсутствие консистентности при построении часто не является проблемой.

Это временное ограничение планируется устранить в одной из следующих версий YDB.

Перестроение векторного индекса

При перестроении создаётся новое дерево кластеров, по которым заново распределяются векторы таблицы. Используйте `ALTER TABLE ... REBUILD INDEX`, если изменения в распределении данных привели к снижению полноты или скорости поиска:

```
ALTER TABLE `my_table` REBUILD INDEX `my_index`;
```

Команда сохраняет имя индекса, набор индексируемых и покрываемых колонок и настройки векторного индекса. Чтобы адаптировать дерево к изменившемуся объёму данных, явно задайте `clusters` и `levels`, которые определяют количество кластеров и уровней дерева:

```
ALTER TABLE `my_table` REBUILD INDEX `my_index`
WITH (clusters = 128, levels = 2);
```

Во время построения существующий индекс продолжает обслуживать запросы и получать обновления таблицы. Когда новая версия готова, YDB атомарно заменяет старый индекс. Приложения продолжают использовать прежнее имя индекса. На время операции требуется место для хранения обеих версий индекса и ресурсы для построения новой версии.

Чтобы ограничить число параллельных обработчиков партиций при перестроении, задайте параметр `parallel`. Например, чтобы одновременно работало не более восьми обработчиков:

```
ALTER TABLE `my_table` REBUILD INDEX `my_index`
WITH (parallel = 8);
```

Новая версия строится по снимку данных, поэтому ограничение консистентности при построении распространяется и на перестроение. Если нужен полностью консистентный индекс:

1. Остановите все приложения и процессы загрузки, записывающие данные в таблицу, и дождитесь завершения уже выполняющихся операций записи. YDB не приостанавливает запись автоматически.
2. Запустите перестроение. Найдите идентификатор операции командой `ydb operation list buildindex` и проверьте её состояние командой `ydb operation get`.
3. Возобновите запись, когда операция вернёт `ready: true` и `status: SUCCESS`.

Запросы остаются доступны во время перестроения, но в момент замены индекса может потребоваться повторная попытка.

Рецепты работы с векторным индексом

Для начала работы с векторным индексом можно воспользоваться следующими рецептами:

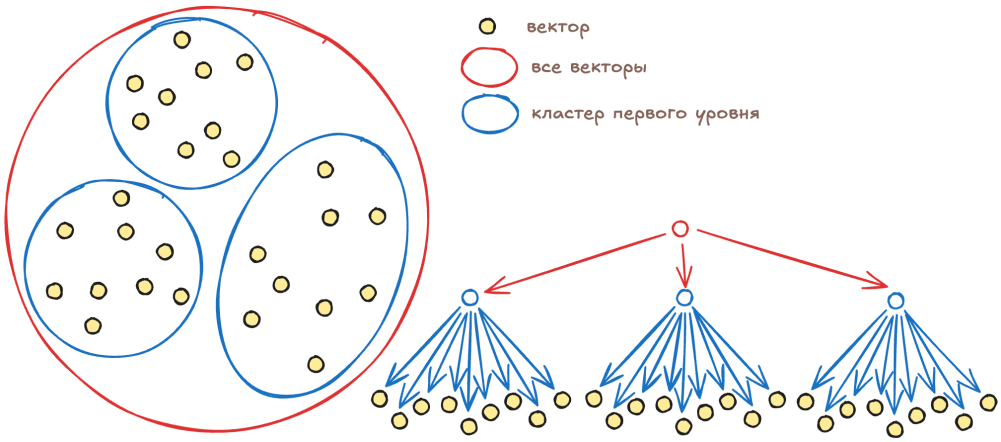
- YDB CLI & YQL
- YDB SDK: Python, C++

Векторный индекс вида vector_kmeans_tree

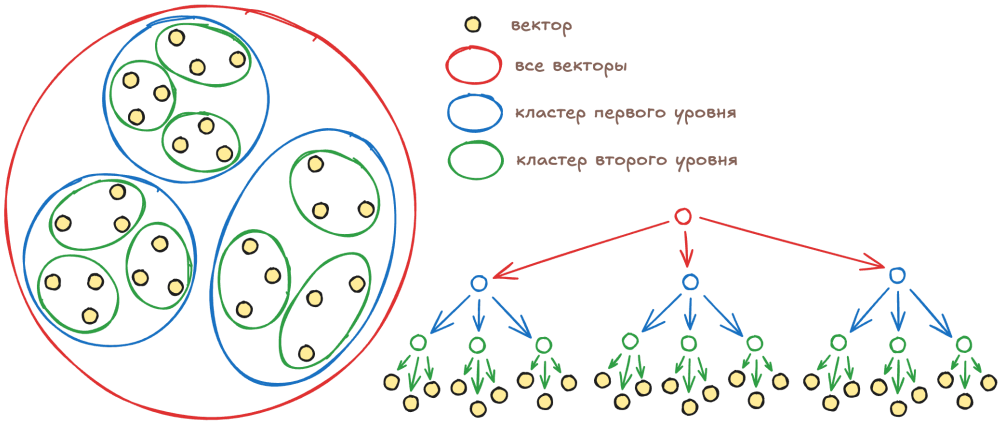
Векторные индексы — это специализированные структуры данных, которые позволяют эффективно выполнять векторный поиск в многомерных пространствах. Работа с ними подробно описана в обзорной статье, а эта статья подробно описывает работу векторного индекса вида `vector_kmeans_tree`.

Создание векторного индекса

Алгоритм `vector_kmeans_tree` рекурсивно делит все векторы, по которым нужно вести поиск, на кластеры с помощью алгоритма K-means. Сначала все векторы разделяются на количество кластеров, указанное при создании индекса с помощью параметра векторного индекса `clusters`. Например, кластеры первого уровня могут выглядеть вот так:



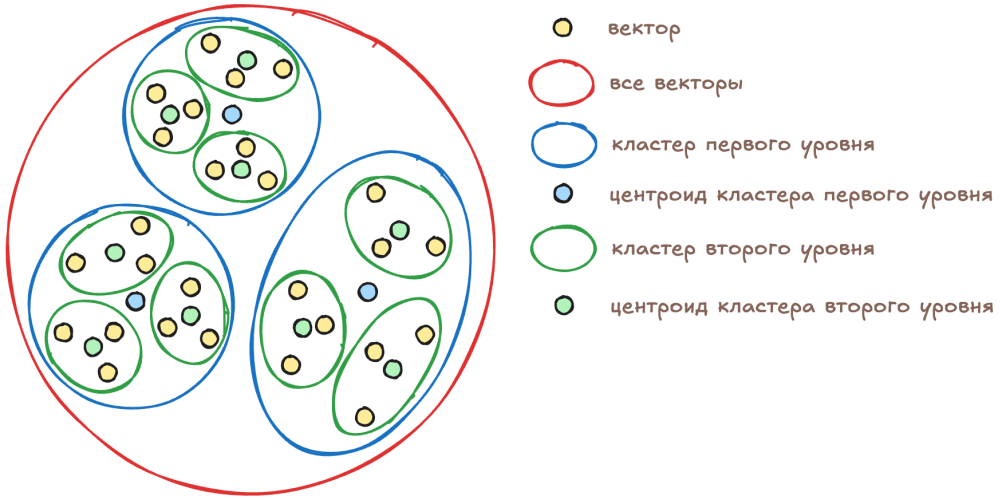
Затем каждый кластер, в свою очередь, разделяется на столько же кластеров. Так как каждый кластер первого уровня состоит из кластеров второго уровня, образуется дерево кластеров. Количество уровней в дереве задаётся с помощью параметра векторного индекса `levels`. Например, кластеры второго уровня будут выглядеть вот так:



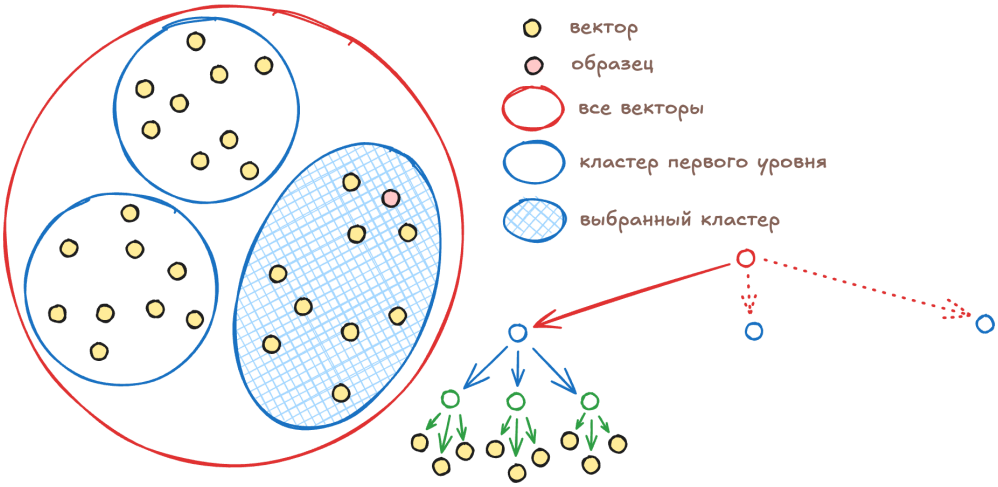
Дерево векторного индекса будет содержать три кластера первого уровня, каждый из которых будет содержать три кластера второго уровня, каждый из которых, в свою очередь, будет содержать три вектора.

Поиск в векторном индексе

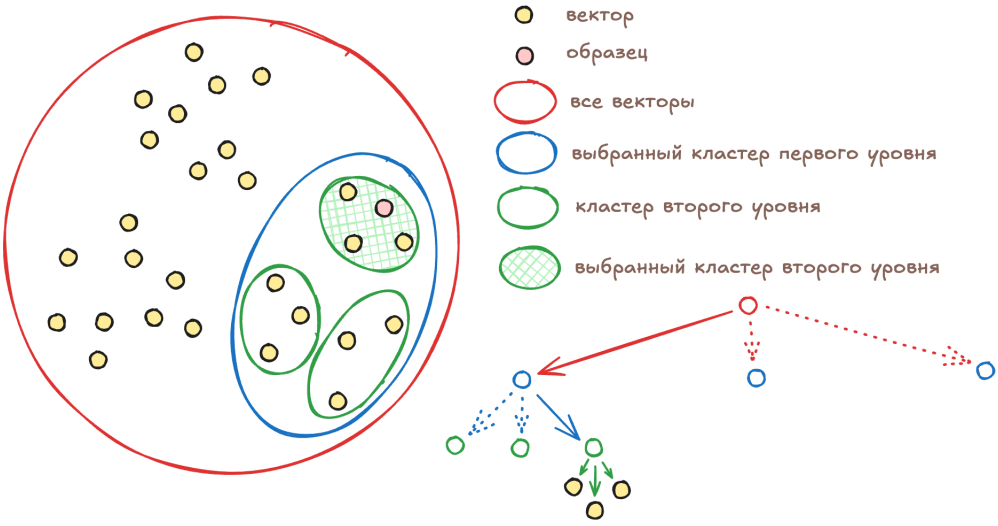
Для поиска в векторном индексе `vector_kmeans_tree` последовательно выбирается на каждом уровне один из нескольких кластеров, в котором содержатся векторы, похожие на образец. Чтобы можно было выбрать нужный кластер, для каждого кластера рассчитывается и сохраняется центроид — среднее арифметическое всех векторов в кластере.



Выбрав кластер на одном из уровней, YDB спускается на уровень ниже и перебирает кластеры, входящие в выбранный на предыдущем уровне кластер. YDB сравнивает расстояния между центроидами кластеров и образцом, выбирая тот кластер, для которого это расстояние минимально. Приближённый поиск исходит из предположения, что если центроид кластера ближе всего к образцу, то и векторы в таком кластере тоже будут ближе всего к образцу.



Каждый кластер первого уровня разделён на кластеры второго уровня, и так далее. Алгоритм последовательно «спускается» по дереву, на каждом шаге выбирая кластер меньшего размера, пока не достигнет кластера самого нижнего уровня, который содержит не список центроидов, а непосредственно список векторов. YDB перебирает эти векторы и выбирает заданное количество ближайших к образцу. Перебирая ограниченное количество кластеров на ограниченном количестве уровней, YDB выполняет поиск со сложностью `O(1)`.



А если в кластере самого нижнего уровня меньше векторов, чем указано в запросе, то YDB вернет все найденные векторы. Чтобы повысить полноту поиска в таких сценариях, нужно воспользоваться параметром `KMeansTreeSearchTopSize`, о котором рассказано далее.

Параметры индекса

YDB позволяет настроить количество уровней векторного индекса с помощью параметра `levels` и количество кластеров на каждом уровне — с помощью параметра `clusters`. Эти параметры следует подбирать с учётом ожидаемого объёма хранимых векторов, а также характеристик кластера: сетевой задержки между узлами, пропускной способности сети и производительности центральных процессоров.

Кластеры, из которых состоит дерево векторного индекса, хранятся в распределённом хранилище. На каждом переходе вниз по дереву YDB выполняет чтение из распределённого хранилища и получает по сети бинарные данные со списком векторов. Время выполнения такой операции зависит от оборудования датацентра, сетевой нагрузки и объёма данных, передаваемых между кластерами.

Выбор одного из кластеров на уровне требует перебора всех векторов-центроидов и вычисления расстояния между ними и образцом. Оценить скорость такого перебора на конкретном оборудовании можно с помощью запроса.

Если данных мало, то параметр `levels` можно установить в один уровень. А для миллиардов векторов может потребоваться увеличить значение этого параметра до четырёх уровней и подобрать оптимальное для вашего оборудования значение параметра `clusters`.

Оптимальное количество кластеров во многом зависит от производительности серверов — сколько векторов они способны перебрать в секунду. Поэтому, выбрав значение `levels` на основе ожидаемого объёма данных и сетевых задержек, значение `clusters` следует подбирать с учётом результатов тестов производительности.

Для большинства аппаратных конфигураций рекомендуется выбирать от 20 до 50 кластеров, это значение выбрано с запасом для типичных векторов длиной в 256 элементов и основано на экспериментах по балансу производительности и полноты результатов поиска.

Также рекомендуется выбирать конфигурацию, при которой в кластерах последнего уровня векторного индекса будет не более 512 векторов, чтобы их перебор укладывался в миллисекунды.

Например, если вы ожидаете, что в таблице будет 10 миллионов векторов, установили параметр `levels` в значение 3, а параметр `clusters` в значение 50, то векторный индекс будет иметь следующий вид:

- 1-й уровень дерева: 50 кластеров по 200 тысяч векторов в каждом;
- 2-й уровень дерева: каждый кластер 1-го уровня разделен на 50 кластеров по 4 тысячи векторов в каждом;

- 3-й уровень дерева: каждый кластер 2-го уровня разделен на 50 кластеров по 80 векторов в каждом.

Автоматический подбор параметров

Параметры `clusters` и `levels` можно не указывать при создании индекса. В этом случае YDB подбирает их автоматически, исходя из количества векторов в таблице. Явно заданные параметры YDB не переопределяет.

Настройка полноты поиска

Если центроид кластера находится ближе всего к образцу, то, как правило, векторы этого кластера также ближе к образцу, чем векторы из других кластеров. Однако при сложных топологиях данных это может не выполняться, поэтому YDB позволяет перебирать векторы более чем в одном кластере на каждом уровне дерева.

Количество кластеров, участвующих в поиске, задаётся параметром `KMeansTreeSearchTopSize`. По умолчанию он равен 4 с перекрывающимися кластерами (`overlap_clusters > 1`) и 10 без перекрытия. Этот параметр указывается не при создании индекса, а для поиска.

Если указать большее значение, например 20, то на каждом уровне система будет выбирать до 20 ближайших кластеров. Для перехода на следующий уровень потребуется получить по сети дочерние центроиды всех выбранных кластеров, из которых затем будут отобраны следующие 20. Такой подход повышает полноту поиска, но увеличивает объём передаваемых по сети данных и количество перебираемых векторов.

Покрывающий индекс

По умолчанию последний кластер хранит не сами векторы для поиска, а только список ключей этих векторов в таблице. Такой список называется Posting Table — это разновидность инвертированного индекса. Хранение ссылок на векторы вместо самих векторов позволяет минимизировать размер векторного индекса.

Однако такой подход означает, что после нахождения последнего кластера YDB должен выполнить `SELECT`-запрос к таблице, чтобы получить несколько сотен элементов (точное количество зависит от количества векторов в последнем кластере).

Запросы на получение векторов из последнего кластера могут быть медленными, если значение `clusters` слишком велико или индекс таблицы не включает колонку с векторами. В таких случаях поиск можно ускорить за счёт увеличения размера индекса: если сделать индекс покрывающим, векторы будут храниться непосредственно в индексе. Данные по-прежнему будут передаваться по сети из распределённого хранилища, но это значительно быстрее, чем выполнение полноценного запроса к индексируемой таблице.

В покрывающий индекс можно включить не только векторы, но и любые другие колонки из индексируемой таблицы. В этом случае `SELECT`-запросы к этим колонкам будут выполняться быстрее ценой хранения избыточной копии их данных в индексе.

Покрывающий индекс можно дополнительно ускорить, если включить чтение с реплик.

Векторный индекс с фильтрацией

Часто возникает необходимость фильтровать объекты по дополнительным атрибутам или метаданным. Векторный индекс с фильтрацией ускоряет запросы с условиями WHERE, позволяя мгновенно получать похожие объекты среди подходящих по фильтру. При использовании такого индекса поиск ближайших векторов производится только среди строк, соответствующих фильтру, что обеспечивает высокую производительность на больших объёмах данных.

Быстрый и точный фильтрованный поиск поддерживается непосредственно на уровне структуры индекса. Для этого фильтруемая колонка должна быть явно указана в SQL-запросе на создание индекса.

Добавление фильтрации непосредственно в векторный индекс связано с тем, что если фильтровать до поиска, то после применения фильтра останется набор строк, в которых нужно найти ближайшие к образцу векторы. Обычный векторный индекс позволяет искать ближайшие векторы только среди всех векторов в таблице, его нельзя применить к произвольному набору строк. Для нахождения ближайших векторов СУБД придётся перебрать все оставшиеся строки и для каждой из них вычислить дистанцию до образца.

Если же фильтровать после поиска, то подходящих результатов может остаться меньше, чем нужно, и качество поиска упадёт. В таком случае нужно будет повторить поиск с увеличенным лимитом, что тоже медленно. В YDB фильтрация интегрирована прямо в структуру и алгоритм векторного индекса, так что не подходящие по условиям объекты сразу исключаются из рассмотрения.

Адаптивное количество кластеров

Индекс с фильтрацией строит отдельное дерево кластеров для каждого уникального значения фильтруемых колонок, а разные значения часто содержат очень разное количество векторов — от миллионов до нескольких десятков. Единое фиксированное значение `clusters` не может быть оптимальным сразу для всех: при малом значении кластеры последнего уровня получаются слишком большими и перебираются медленно, а при большом — появляется множество почти пустых кластеров, что расходует место и может снизить полноту поиска.

Адаптивное количество кластеров решает эту проблему: когда режим включён, YDB выбирает количество кластеров *для каждого значения фильтруемой колонки* автоматически, исходя из того, сколько векторов это значение фактически содержит. Значения с большим количеством векторов получают больше кластеров (вплоть до значения `clusters`, которое выступает верхней границей), а значения с малым количеством векторов — меньше.

Адаптивное количество кластеров управляется параметром индекса `adaptive_clusters`:

- `adaptive_clusters` применяется только к индексам с фильтрацией (индексам с более чем одной индексируемой колонкой). Для индексов без фильтрации этот параметр не действует.
- Если при создании индекса с фильтрацией параметры не заданы явно и YDB подбирает их автоматически, адаптивное количество кластеров включается по умолчанию.
- Если параметры индекса заданы явно, то `adaptive_clusters` по умолчанию равен `false`; чтобы включить адаптивный режим, задайте `adaptive_clusters` равным `true`.

В адаптивном режиме параметр `clusters` интерпретируется как *максимальное* количество кластеров для одного значения фильтруемой колонки. Фактическое количество для каждого значения вычисляется на основе количества его векторов, параметров `levels` и `overlap_clusters` и никогда не бывает меньше 2. Значения с очень малым количеством векторов (менее 100) всегда используют минимум — 2 кластера.

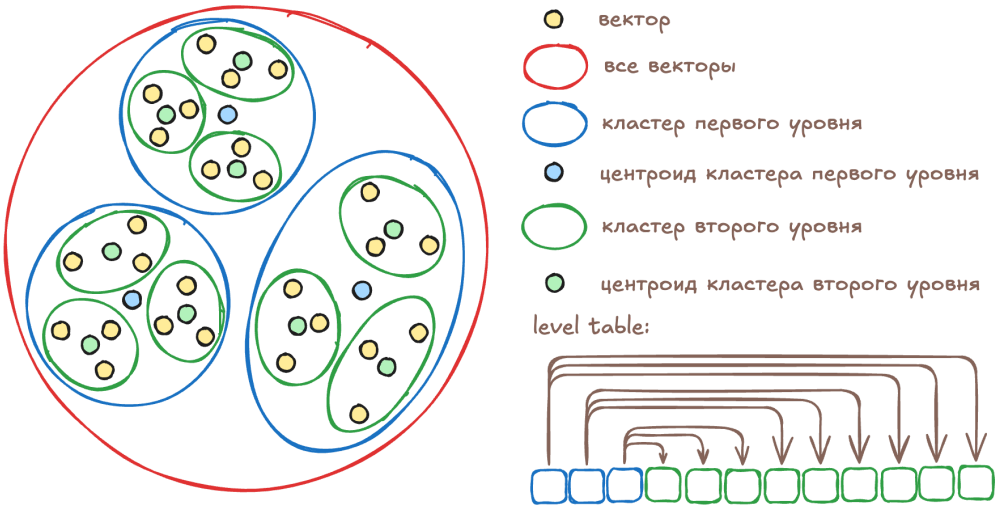
Пример создания индекса с фильтрацией и адаптивным количеством кластеров:

```
ALTER TABLE my_table
  ADD INDEX my_index
  GLOBAL USING vector_kmeans_tree
  ON (user, embedding)
  COVER (embedding, data)
  WITH (
    distance=cosine,
    vector_type="float",
    vector_dimension=512,
    clusters=512,
    levels=2,
    adaptive_clusters=true
  );
```

Здесь `clusters=512` — верхняя граница: значение `user` с миллионами векторов может получить количество кластеров, близкое к 512, а значение `user` с несколькими десятками векторов получит только 2.

Внутреннее устройство индекса

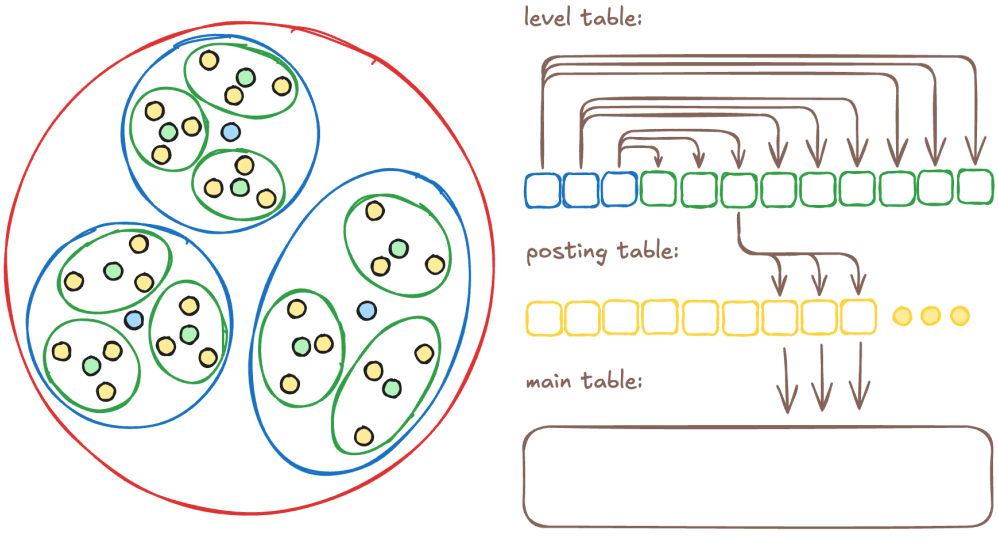
Индекс состоит из двух дополнительных таблиц: level table (`indexImplLevelTable`) и posting table (`indexImplPostingTable`). В таблице level table хранятся центроиды для дерева кластеров. К этой таблице YDB делает запросы при переходе с уровня на уровень во время векторного поиска. Например, если векторный индекс построен для 27 векторов, параметр `levels` установлен в значение 2, а параметр `clusters` — в значение 3, то level table будет выглядеть следующим образом:



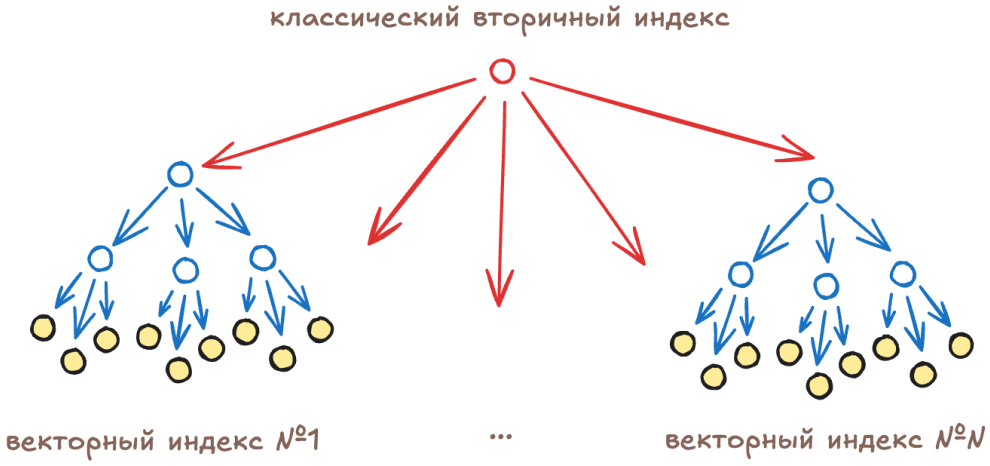
Для каждого центроида в level table также хранится информация о принадлежности к кластеру на вышестоящем уровне. YDB использует эту информацию, чтобы при переходе на нижележащий уровень запрашивать только те центроиды, которые принадлежат кластерам, выбранным на предыдущем уровне.

Чтобы не перечитывать центроиды из распределённого хранилища при каждом запросе, содержимое level table можно кэшировать в памяти на каждом узле — см. параметр `resource_manager.kqp_level_cache_max_size_bytes` в `table_service_config`.

Вторая таблица — posting table — используется для связи центроидов последнего уровня из level table с векторами в исходной таблице, для которой построен индекс. Если используется покрывающий индекс, в эту же таблицу сохраняются векторы и дополнительные колонки, указанные разработчиком при создании индекса:



При использовании индекса с фильтрацией перед векторным индексом добавляется классический вторичный индекс для указанных колонок. А для каждого уникального элемента вторичного индекса строится отдельный векторный индекс. Технически вторичный индекс помещается в дополнительную реляционную таблицу prefix table (`indexImplPrefixTable`), а все деревья векторных индексов остаются в таблице level table:



Структура таблицы `indexImplLevelTable`

В первичный ключ этой таблицы входят колонки `__ydb_parent` и `__ydb_id`.

Колонка	Тип данных	Описание
`__ydb_parent`	`Uint64`	Идентификатор родительского кластера
`__ydb_id`	`Uint64`	Идентификатор текущего кластера
`__ydb_centroid`	`String`	Вектор центроида текущего кластера

Структура таблицы `indexImplPostingTable`

В первичный ключ этой таблицы входят колонка `__ydb_parent` и колонки, соответствующие первичному ключу индексируемой таблицы (`id` в примере ниже).

Колонка	Тип данных	Описание
`__ydb_parent`	`Uint64`	Идентификатор кластера, содержащего текущий вектор
`id`	`Uint64`	Первичный ключ индексируемой таблицы с векторами. Названия и типы колонок копируются из индексируемой таблицы.

Пример структуры таблицы `indexImplPostingTable` с покрывающим индексом

При использовании покрывающего индекса в таблицу `indexImplPostingTable` копируются указанные колонки из индексируемой таблицы. В первичный ключ входят те же колонки, что и для `indexImplPostingTable`. Вот так будет выглядеть таблица, если при создании покрывающего индекса указать колонку `embedding`:

Колонка	Тип данных	Описание
`__ydb_parent`	`Uint64`	Идентификатор кластера, содержащего текущий вектор
`id`	`Uint64`	Первичный ключ индексируемой таблицы с векторами. Названия и типы колонок копируются из индексируемой таблицы.
`embedding`	`String`	Копия вектора из индексируемой таблицы

Пример структуры таблицы `indexImplPrefixTable` для индекса с фильтрацией

При использовании индекса с фильтрацией в таблицу `indexImplPrefixTable` копируются названия и типы колонок из индексируемой таблицы. Вот так будет выглядеть таблица, если при создании индекса с фильтрацией указать колонку `category_id`:

Колонка	Тип данных	Описание
`category_id`	`Uint64`	Названия и типы колонок копируются из индексируемой таблицы
`__ydb_id`	`Uint64`	Идентификатор корневого кластера векторного индекса

Полнотекстовые индексы

Полнотекстовые индексы — это специализированный тип вторичного индекса, который позволяет эффективно выполнять поиск по текстовому содержимому в колонках таблицы: по словам и (с N-граммами) по подстрокам. В отличие от традиционных вторичных индексов, оптимизированных для поиска по равенству или диапазону, полнотекстовые индексы обеспечивают поиск по текстовому содержимому.

Общее описание полнотекстового поиска см. в разделе Полнотекстовый поиск.

Характеристики полнотекстовых индексов

Полнотекстовые индексы в YDB строятся путём токенизации текста и создания инвертированного индекса. Это позволяет:

- быстро фильтровать строки через FulltextMatch;
- ранжировать результаты по релевантности (BM25) через FulltextScore при использовании fulltext_relevance;
- применять нормализацию регистра, лемматизацию и N-граммы с помощью фильтров индекса.

В текущей реализации доступны два варианта индекса:

- fulltext_plain — базовый полнотекстовый индекс;
- fulltext_relevance — полнотекстовый индекс со статистикой BM25 для расчёта релевантности.

Кроме этого, полнотекстовый индекс может быть **покрывающим** (через `COVER`) и включать копию данных дополнительных колонок из основной таблицы.

Виды полнотекстовых индексов

YDB поддерживает два типа полнотекстовых индексов, которые различаются составом хранимой статистики:

- fulltext_plain — хранит только инвертированный индекс. Поддерживает фильтрацию через FulltextMatch, но не позволяет ранжировать результаты по релевантности.
- fulltext_relevance — дополнительно хранит частотную статистику (TF-IDF / BM25), необходимую для работы FulltextScore.

Индекс любого из этих типов также может быть с фильтрацией для ограничения поиска конкретным логическим разделом.

Базовый полнотекстовый индекс (`fulltext_plain`)

Используйте `fulltext_plain`, когда достаточно проверить наличие термов в тексте без ранжирования по релевантности. Такой индекс компактнее `fulltext_relevance` и подходит для большинства задач фильтрации.

Пример создания глобального полнотекстового индекса по колонке `body`:

```
ALTER TABLE articles
  ADD INDEX ft_index
  GLOBAL USING fulltext_plain
  ON (body)
  WITH (tokenizer=standard, use_filter_lowercase=true);
```

Здесь `tokenizer=standard` разбивает текст на слова по пробелам и знакам препинания, а `use_filter_lowercase=true` нормализует все токены к нижнему регистру — это делает поиск регистронезависимым. Также можно использовать готовый анализатор.

Пример запроса к индексу:

```
SELECT id, title
FROM articles VIEW ft_index
WHERE FulltextMatch(body, "поисковые термы")
LIMIT 20;
```

Полнотекстовый индекс для ранжирования (`fulltext_relevance`)

`fulltext_relevance` хранит инвертированный индекс вместе с частотной статистикой (BM25), которая позволяет функции FulltextScore вычислять оценку релевантности документа запросу. Используйте этот тип, когда нужно не просто найти документы с нужными словами, но и упорядочить их по степени соответствия.

Пример создания индекса:

```
ALTER TABLE articles
  ADD INDEX ft_index
  GLOBAL USING fulltext_relevance
  ON (body)
  WITH (tokenizer=standard, use_filter_lowercase=true);
```

Пример запроса с ранжированием:

```
SELECT id, title, FulltextScore(body, "поисковые термы") AS relevance
FROM articles VIEW ft_index
WHERE FulltextScore(body, "поисковые термы") > 0
ORDER BY relevance DESC
LIMIT 10;
```

Поиск по подстроке (N-граммы)

Если вам нужен поиск по подстроке, создайте индекс с N-граммами. Доступны два типа N-грамм:

- **Обычные N-граммы** (`use_filter_ngram`) — разбивают слова на все возможные подстроки заданной длины, что позволяет находить совпадения в любой части слова. Например, слово "search" будет разбито на "sea", "ear", "arc", "rch" и т.д.

- **Краевые N-граммы** (`use_filter_edge_ngram`) — создают подстроки только от начала слова, что идеально подходит для автодополнения. Например, слово "search" будет разбито на "se", "sea", "sear", "searc", "search".

При использовании N-грамм станут доступны:

- FulltextMatch(..., "Wildcard" AS Mode) — поиск с шаблонами `%` и `_` (аналогично `LIKE`);
- предикаты `LIKE` / `ILIKE` по индексируемой текстовой колонке — YDB автоматически использует N-граммовый индекс при обращении к нему через `VIEW IndexName` .

Пример индекса с N-граммами:

```
ALTER TABLE articles
  ADD INDEX ngram_index
  GLOBAL USING fulltext_plain
  ON (body)
  WITH (
    tokenizer=standard,
    use_filter_lowercase=true,
    use_filter_ngram=true,
    filter_ngram_min_length=3,
    filter_ngram_max_length=5
  );
```

Пример запроса с `FulltextMatch` :

```
SELECT id, title
FROM articles VIEW ngram_index
WHERE FulltextMatch(body, "%обуч%", "Wildcard" AS Mode)
LIMIT 20;
```

Пример запроса с `LIKE` :

```
SELECT id, title
FROM articles VIEW ngram_index
WHERE body LIKE "%обуч%ние%"
LIMIT 20;
```

`LIKE` / `ILIKE` используют N-граммный индекс для отбора кандидатов, затем проверяют исходный текст. `LIKE` учитывает регистр, `ILIKE` регистронезависим. См. ограничения поиска по шаблонам.

Полнотекстовый индекс с фильтрацией

Полнотекстовый индекс с фильтрацией позволяет выполнять полнотекстовый поиск в рамках логического раздела, заданного значениями колонок фильтрации. Чтобы создать такой индекс, укажите одну или несколько колонок фильтрации перед текстовой колонкой в секции `ON` . Последней колонкой должна быть текстовая, остальные должны иметь типы, допустимые в первичном ключе:

```
ALTER TABLE articles
  ADD INDEX ft_index
  GLOBAL USING fulltext_plain
  ON (user_id, body)
  WITH (tokenizer=standard, use_filter_lowercase=true);
```

Запросы к такому индексу должны включать предикат равенства для каждой колонки фильтрации:

```
SELECT id, title
FROM articles VIEW ft_index
WHERE user_id = 42 AND FulltextMatch(body, "поисковые термы")
LIMIT 20;
```

Поддерживается несколько колонок фильтрации. Предикаты равенства в `WHERE` могут быть указаны в любом порядке — YDB автоматически упорядочит их в соответствии с порядком колонок в индексе.

Типы первичного ключа

Внутри инвертированного индекса каждый проиндексированный документ идентифицируется числовым идентификатором документа (`doc_id`). Способ получения `doc_id` зависит от первичного ключа основной таблицы:

- **Первичный ключ из одной целочисленной колонки** (`Uint64` , `Int64` , `Uint32` или `Int32`) — значение первичного ключа используется напрямую в качестве `doc_id` . Это самая компактная форма, дополнительные структуры не создаются.
- **Любой другой первичный ключ** (например, `Utf8` , `String` , другие нецелочисленные типы или составной ключ из нескольких колонок) — YDB поддерживает системную колонку `__ydb_row_id` типа `Uint64` и использует её в качестве `doc_id` .

При создании полнотекстового индекса на таблице, первичный ключ которой не является одной целочисленной колонкой, YDB автоматически:

- добавляет в таблицу колонку `__ydb_row_id` — существующие строки заполняются во время построения индекса;
- генерирует значение `__ydb_row_id` для каждой строки, в которой эта колонка не указана в `INSERT` / `UPSERT` ;
- создаёт уникальный вторичный индекс с именем `__ydb_unique_row_id` по колонке `__ydb_row_id` . Во время запроса этот индекс сопоставляет найденный `__ydb_row_id` с первичным ключом таблицы перед чтением строки из основной таблицы.

Системная колонка не входит в результат `SELECT *`, но её можно выбрать явно по имени. Изменение существующей строки через `UPSERT` или `UPDATE` сохраняет её `__ydb_row_id`.

Если на таблице создаётся несколько полнотекстовых индексов, все они **переиспользуют** одну и ту же колонку `__ydb_row_id` и индекс `__ydb_unique_row_id` — эти структуры создаются для таблицы только один раз.

Важно

Колонкой `__ydb_row_id` и индексом `__ydb_unique_row_id` управляет YDB:

- Позвольте YDB заполнять `__ydb_row_id` — не указывайте её в `INSERT` / `UPSERT`, и значение проставится автоматически. Модификация колонки `__ydb_row_id` пользователем запрещена — любые попытки задать или изменить значение этой колонки будут отклонены.
- Индекс `__ydb_unique_row_id` нельзя удалить, пока от него зависит хотя бы один полнотекстовый индекс. Сначала удалите зависимые полнотекстовые индексы.

Полный синтаксис полнотекстовых индексов

Создание полнотекстового индекса:

- при создании таблицы: CREATE TABLE;
- добавление к существующей таблице: ALTER TABLE.

Полный синтаксис запроса к полнотекстовому индексу:

- VIEW (Полнотекстовый индекс).

Функции и выражения для полнотекстового поиска:

- Базовые функции полнотекстового поиска;
- LIKE / ILIKE с полнотекстовым индексом.

Примечание

Полнотекстовый индекс не будет автоматически выбран оптимизатором, поэтому его нужно указывать явно с помощью `VIEW IndexName`.

Без `VIEW` запросы с `FulltextMatch` / `FulltextScore` завершаются с ошибкой, кроме `FulltextScore` внутри `HybridRank`.

Это ограничение может быть снято в будущих версиях YDB.

Обновление полнотекстовых индексов

Полнотекстовые индексы автоматически поддерживаются при модификации данных. Таблицы с полнотекстовыми индексами поддерживают:

- `INSERT`
- `UPSERT`
- `REPLACE`
- `UPDATE`
- `DELETE`

Удаление полнотекстовых индексов

```
ALTER TABLE articles DROP INDEX ft_index;
```

Ограничения

- Для таблиц с нецелочисленным или составным первичным ключом автоматически создаются системная колонка `__ydb_row_id` и уникальный индекс `__ydb_unique_row_id` (см. Типы первичного ключа).
- `BulkUpsert` не поддерживается для таблиц с полнотекстовыми индексами.
- Использование полнотекстового индекса необходимо задавать явно с помощью `VIEW IndexName`.
- В одном полнотекстовом индексе индексируется одна текстовая колонка.
- `FulltextMatch` / `FulltextScore` нельзя использовать с `OR` или `NOT`. Допускается комбинация с другими предикатами через `AND`.
- Несколько предикатов `FulltextMatch` с настройками поиска по ключевым словам по умолчанию можно объединять через `AND`. В одном чтении через `VIEW` нельзя смешивать `FulltextMatch` и `FulltextScore` или использовать несколько различных предикатов `FulltextScore`.
- Для доступа к индексу по релевантности требуется ограничение `FulltextScore(...) > 0` в `WHERE` (иначе запрос завершится с ошибкой).
- Полнотекстовые индексы с фильтрацией: для каждой колонки фильтрации требуется предикат равенства в `WHERE`.
- Полнотекстовые индексы с фильтрацией: колонки фильтрации могут включать часть первичного ключа, но не все его колонки.

Гибридный поиск

Гибридный поиск объединяет полнотекстовый поиск и векторный поиск в едином ранжированном результате: каждый документ оценивается одновременно по релевантности текста и по близости эмбеддингов, а два ранжирования объединяются в одно. Это сочетает точность лексического сопоставления и полноту семантической близости и является распространённым строительным блоком для этапа извлечения данных в подходе Retrieval-Augmented Generation (RAG).

Общее описание гибридного поиска см. в разделе Понятие гибридного поиска.

Гибридный поиск — это не отдельный тип индекса. Он переиспользует два существующих индекса одной и той же таблицы:

- fulltext_relevance по текстовой колонке — даёт сигнал релевантности BM25;
- vector_kmeans_tree по колонке с эмбеддингами — даёт сигнал близости ближайших соседей (семантический).

Подготовка индексов

Создайте таблицу с текстовой колонкой и колонкой эмбеддингов, затем добавьте оба индекса.

```
CREATE TABLE documents (
    id Uint64,
    text Utf8,
    embedding String,
    PRIMARY KEY (id)
);
```

Добавьте индекс `fulltext_relevance` по текстовой колонке (для ранжирования по релевантности требуется именно этот тип, а не `fulltext_plain`):

```
ALTER TABLE documents
  ADD INDEX ft_idx
  GLOBAL USING fulltext_relevance
  ON (text)
  WITH (tokenizer=standard, use_filter_lowercase=true);
```

Добавьте индекс `vector_kmeans_tree` по колонке эмбеддингов:

```
ALTER TABLE documents
  ADD INDEX vec_idx
  GLOBAL USING vector_kmeans_tree
  ON (embedding)
  WITH (distance=cosine);
```

Подробности о каждом типе индекса — в разделах Полнотекстовые индексы и Векторные индексы.

Выполнение гибридного запроса

Гибридный запрос — это обычный `SELECT` по основной таблице (без `VIEW`), у которого ключ `ORDER BY` представляет собой единственный вызов `HybridRank`. `HybridRank` принимает по одному оценивающему выражению на каждую ветвь: FullTextScore для текстовой ветви и Knn-расстояние или сходство для векторной ветви.

```
PRAGMA ydb.KMeansTreeSearchTopSize = "10";

$queryText = "машинное обучение";
$queryVector = Knn::ToBinaryStringFloat([0.1, 0.2, 0.3, 0.4]);

SELECT id, text
FROM documents
ORDER BY HybridRank(
    FullTextScore(text, $queryText),
    Knn::CosineDistance(embedding, $queryVector))
LIMIT 10;
```

Оба входа происходят из одного пользовательского запроса: `$queryText` — это текст поиска (сопоставляется лексически), а `$queryVector` — его эмбеддинг, вычисленный приложением (сопоставляется семантически). YDB не вычисляет эмбеддинги самостоятельно, поэтому вектор передаётся извне — здесь он построен из литерала с помощью Knn::ToBinaryStringFloat, но в приложении это результат работы модели эмбеддингов.

YDB автоматически сопоставляет каждую ветвь с её индексом по оцениваемой колонке (`text` → полнотекстовый индекс релевантности, `embedding` → векторный индекс), извлекает пул кандидатов из каждой ветви и объединяет два ранжирования. Результат — топ `LIMIT` документов по объединённой оценке.

Обратите внимание, что, в отличие от полнотекстового и векторного поиска, гибридный запрос **не** использует `VIEW IndexName`: индексы выбираются по аргументам `HybridRank`, а чтение идёт по основной таблице.

`PRAGMA ydb.KMeansTreeSearchTopSize` управляет полнотой векторной ветви — см. KMeansTreeSearchTopSize. Как и при обычном векторном поиске, его следует задавать явно.

Настройка объединения

Способ объединения и его параметры передаются как именованные аргументы `HybridRank`. Наиболее употребительные:

- `Mode` — `"rrf"` (по умолчанию, Reciprocal Rank Fusion) или `"linear"` (взвешенная сумма нормализованных оценок);
- `Weights` — кортеж весов ветвей, по одному значению на оценивающий аргумент, для смещения ранжирования в сторону одного из сигналов;
- `K` — константа RRF (по умолчанию `60.0`);
- `Indexes` / `Limits` — явные имена индексов и размеры пулов кандидатов по ветвям.

Для встроенных режимов вклад каждой ветви фиксирован формулой. Для полного контроля над объединением передайте вместо `Mode` пользовательскую лямбду:

- `RankLambda` — лямбда, получающая ранги документа по ветвям (1-based позиция внутри каждой ветви) и возвращающая итоговую оценку;
- `ScoreLambda` — лямбда, получающая исходные оценки документа по ветвям (релевантность полнотекстового поиска или векторное расстояние/сходство) и возвращающая итоговую оценку.

Пользовательская лямбда заменяет встроенное объединение, поэтому её нельзя комбинировать с `Mode`, `Weights`, `K` или `Normalize` — перенесите все веса и константы в тело лямбды. Подробное описание и примеры — в разделе Синтаксис гибридного запроса (HybridRank).

Например, чтобы дать векторной ветви вдвое больший вес, чем текстовой, в режиме RRF:

```
$queryText = "машинное обучение";
$queryVector = Knn::ToBinaryStringFloat([0.1, 0.2, 0.3, 0.4]);

SELECT id, text
FROM documents
ORDER BY HybridRank(
    FullTextScore(text, $queryText),
    Knn::CosineDistance(embedding, $queryVector),
    (1, 2) AS Weights)
LIMIT 10;
```

Полный список параметров и их семантика — в разделе Гибридный поиск (HybridRank).

По умолчанию каждая ветвь извлекает не более `LIMIT * 10` кандидатов. Для настройки пулов используйте `PRAGMA ydb.HybridSearchFactor` или явный `AS Limits`. Кандидаты, найденные только одной ветвью, также могут попасть в результат. Фильтры, не задающие префикс, применяются после объединения и могут сократить результат ниже `LIMIT`; см. пулы кандидатов и фильтрация.

Ограничения

- Для таблицы должны существовать готовые индексы `fulltext_relevance` и `vector_kmeans_tree` по соответствующим оцениваемым колонкам, с совместимыми векторными метриками; иначе запрос завершится с ошибкой.
- Текущая реализация поддерживает только таблицы с одноколоночным первичным ключом.
- Префиксные полнотекстовые и векторные индексы требуют равенств для всех колонок префикса в `WHERE`; см. выбор индексов и фильтрация.
- Если подходит несколько полнотекстовых индексов, используйте явный `AS Indexes`. Векторные ветви выбирают совместимый индекс с самым длинным полностью заданным префиксом; при одинаковой максимальной длине префикса также требуется `AS Indexes`.
- `LIMIT` должен быть литералом, так как он задаёт размеры пулов кандидатов по ветвям. Чтобы использовать параметризованный `LIMIT`, передайте явный `AS Limits`.
- `HybridRank(...)` должен быть единственным ключом `ORDER BY` — его нельзя отрицать, оборачивать в другое выражение или комбинировать с другими ключами сортировки.
- Пользовательская лямбда объединения (`RankLambda` или `ScoreLambda`) заменяет встроенное объединение и не может комбинироваться с `Mode`, `Weights`, `K` или `Normalize`. Можно указать не более одного из `RankLambda` или `ScoreLambda`.

JSON-индексы

JSON-индексы — это разновидность вторичного индекса, реализованная поверх инвертированного индекса, которая ускоряет фильтрацию строк таблицы по условиям, накладываемым на содержимое колонок типа `Json` и `JsonDocument`. Индекс задействуется, если в предикате `WHERE` используются функции JSON_EXISTS и JSON_VALUE с выражениями JsonPath. В отличие от традиционных вторичных индексов, оптимизированных для поиска по равенству или диапазону отдельных колонок таблицы, JSON-индекс работает с произвольными путями внутри JSON-документа.

Общее описание поиска по JSON и устройства инвертированного индекса по путям JSON-документа см. в разделе Поиск по содержимому JSON-документов.

Характеристики JSON-индексов

JSON-индексы в YDB позволяют:

- быстро фильтровать строки по JSON_EXISTS и JSON_VALUE с выражениями JsonPath;
- комбинировать индексируемые условия операторами `AND` и `OR`;
- использовать значения параметров запроса, переданные приложением, при обработке проверяемых предикатов.

JSON-индекс является глобальным синхронным индексом — его данные всегда согласованы с основной таблицей.

При выполнении запроса JSON-индекс может быть применён:

- явно — через оператор `<имя_таблицы> VIEW <имя_индекса>`;
- автоматически — оптимизатором, если предикат подходит под формальные правила.

Синтаксис JSON-индексов

Создание JSON-индекса:

- при создании таблицы: INDEX (CREATE TABLE);
- добавление к существующей таблице: ALTER TABLE.

Удаление JSON-индексов выполняется через ALTER TABLE:

```
ALTER TABLE documents DROP INDEX json_idx
```

Синтаксис запроса с явным указанием JSON-индекса:

- VIEW (JSON-индекс).

Функции и выражения для работы с JSON в предикатах:

- Функции для работы с JSON — `JSON_EXISTS` и `JSON_VALUE`;
- JsonPath — язык запросов для обращения к значениям внутри JSON.

Готовые сценарии использования собраны в разделе рецептов поиска по JSON-документам.

Обновление JSON-индексов

JSON-индексы автоматически поддерживаются при модификации данных и обновляются синхронно вместе с основной таблицей. Таблицы с JSON-индексами поддерживают:

- `INSERT`
- `UPSERT`
- `REPLACE`
- `UPDATE`
- `DELETE`

Пакетные операции (`BATCH UPDATE` и `BATCH DELETE`) для таблиц с JSON-индексами не поддерживаются. При попытке выполнения такого запроса над таблицей, для которой создан JSON-индекс, этот запрос будет отклонен, в приложение будет возвращена соответствующая ошибка, а данные останутся в неизменном состоянии.

Кроме того, для таблиц с JSON-индексами не поддерживается:

- массовая загрузка данных через вызов `BulkUpsert` — запрошенная операция будет отклонена с соответствующим сообщением об ошибке;
- автоматическое удаление строк по TTL — будут возвращены ошибки при попытке создать таблицу одновременно с TTL-политикой и JSON-индексом, а также при попытке получить такое сочетание свойств через команды `ALTER TABLE`.

Поддерживаемые предикаты

Для выполнения через JSON-индексы поддерживаются только выражения на основе функций `JSON_EXISTS` и `JSON_VALUE` в блоке `WHERE`, объединённые операторами `AND` / `OR` по правилам ниже.

JSON_EXISTS

Проверка существования пути или значения внутри фильтра JsonPath.

Разрешено:

```
-- Корень документа (значение не NULL)
WHERE JSON_EXISTS(doc, '$')

-- Цепочка ключей; индексы массивов «прозрачны»
WHERE JSON_EXISTS(doc, '$.user.name')
WHERE JSON_EXISTS(doc, '$.items[*].sku')
WHERE JSON_EXISTS(doc, '$.items[0 to last].active')

-- Фильтр ? (...) — предикаты внутри фильтра допустимы
WHERE JSON_EXISTS(doc, '$.items ? (@.price == 100)')
```

```
WHERE JSON_EXISTS(doc, '$.items ? (@.qty >= 1 && @.qty <= 10)')
WHERE JSON_EXISTS(doc, '$.items ? (@.tag == $t)' PASSING "sale" AS t)

-- Методы JsonPath (путь индексируется до метода; точную проверку выполняет пост-фильтр)
WHERE JSON_EXISTS(doc, '$.value.type()')
WHERE JSON_EXISTS(doc, '$.arr.size()')

-- Комбинации на одной колонке
WHERE JSON_EXISTS(doc, '$.a') AND JSON_EXISTS(doc, '$.b')
WHERE JSON_EXISTS(doc, '$.a') OR JSON_EXISTS(doc, '$.b')
```

Запрещено (ошибка при использовании оператора `VIEW` или отказ автовыбора индекса):

```
-- Предикаты сравнения на верхнем уровне пути (вне ? (...))
WHERE JSON_EXISTS(doc, '$.key == 10')
WHERE JSON_EXISTS(doc, 'exists($.key)')
WHERE JSON_EXISTS(doc, '$.key starts with "a"')

-- Отрицание в JsonPath
WHERE JSON_EXISTS(doc, '!($.key == 10)')

-- ON ERROR TRUE
WHERE JSON_EXISTS(doc, '$.key' TRUE ON ERROR)

-- Путь без оператора контекста ($) — переданный документ не используется
WHERE JSON_EXISTS(doc, '1')
```

> **Примечание**
>
> Функция `JSON_EXISTS` возвращает `true` для любого непустого результата JsonPath. Предикат `$.key == 10`, указанный на верхнем уровне, дал бы «существование пути» даже когда сравнение ложно, что не соответствует ожидаемой семантике. Сравнения нужно выносить в вызовы `JSON_VALUE` или в фильтр вида `? (...)`.

JSON_VALUE

Извлечение скалярного значения с обязательным `RETURNING <тип>`.

Для сравнения значения через `JSON_VALUE` всегда нужно указывать `RETURNING` с нужным типом. По умолчанию `JSON_VALUE` возвращает тип `Utf8`, что во время выполнения запроса приводит к некорректному сравнению — значения разных типов сравниваются как строки:

```
$tmp = Json(@@["1", 1]@@);
SELECT JSON_VALUE($tmp, '$[0]') == "1"; -- true: корректно, строка сравнивается со строкой
SELECT JSON_VALUE($tmp, '$[1]') == "1"; -- true: некорректно, число сравнивается со строкой
```

Поддерживаемые типы для секции `RETURNING`: `Int8` … `Int64`, `Uint8` … `Uint64`, `Float`, `Double`, `Bytes` (`String`), `Text` (`Utf8`), `Bool`.

Примеры предикатов, применяемых через JSON-индекс:

```
-- Равенство (путь + значение попадают в индекс)
WHERE JSON_VALUE(doc, '$.user.age' RETURNING Int32) = 25
WHERE JSON_VALUE(doc, '$.flag' RETURNING Bool) = true
WHERE JSON_VALUE(doc, '$.name' RETURNING Utf8) = "Alice"u

-- Неявное сравнение с true для Bool
WHERE JSON_VALUE(doc, '$.active' RETURNING Bool)

-- Параметры
WHERE JSON_VALUE(doc, '$.user.id' RETURNING Int64) = $id
WHERE JSON_VALUE(doc, '$.tag' RETURNING Utf8) = $tag

-- Сравнения (в индексе задействован только путь, сравнение выполняет пост-фильтр)
WHERE JSON_VALUE(doc, '$.score' RETURNING Int64) > 0
WHERE JSON_VALUE(doc, '$.score' RETURNING Int64) != 100
WHERE JSON_VALUE(doc, '$.score' RETURNING Int64) BETWEEN 1 AND 10
WHERE JSON_VALUE(doc, '$.score' RETURNING Int64) NOT BETWEEN 0 AND 5

-- IN: список литералов
WHERE JSON_VALUE(doc, '$.status' RETURNING Utf8) IN ("open"u, "pending"u)

-- IN: заданный параметр типа List<Utf8>
WHERE JSON_VALUE(doc, '$.status' RETURNING Utf8) IN $status_list

-- PASSING для переменных JsonPath
WHERE JSON_VALUE(doc, '$.x ? (@.y == $v)' RETURNING Int64 PASSING 42 AS v) = 10

-- Предикаты JsonPath внутри пути.
-- В отличие от JSON_EXISTS, разрешены предикаты на верхнем уровне.
WHERE JSON_VALUE(doc, '$.user ? (@.role == "admin")' RETURNING Utf8) = "ok"u
WHERE JSON_VALUE(doc, '$.code starts with "A"' RETURNING String) != ""
WHERE JSON_VALUE(doc, 'exists($.meta)' RETURNING Bool)

-- Комбинации AND / OR на одной колонке
WHERE JSON_VALUE(doc, '$.a' RETURNING Int32) = 1
```

```
   OR JSON_VALUE(doc, '$.b' RETURNING Int32) = 2
WHERE JSON_EXISTS(doc, '$.a') AND JSON_VALUE(doc, '$.a' RETURNING Int32) = 10
```

Примеры предикатов, которые не могут быть применены через JSON-индекс:

```
-- Вызов JSON_VALUE без RETURNING
WHERE JSON_VALUE(doc, '$.key') = "x"

-- DEFAULT при ON EMPTY / ON ERROR (кроме NULL)
WHERE JSON_VALUE(doc, '$.k' RETURNING Utf8 DEFAULT "x" ON ERROR) = "y"

-- Неподдерживаемый тип данных в RETURNING
WHERE JSON_VALUE(doc, '$.ts' RETURNING Timestamp) = ...

-- RETURNING Bool с операторами сравнения
WHERE JSON_VALUE(doc, '$.flag' RETURNING Bool) >= true

-- IS NULL / IS NOT NULL — семантически противоречат индексу «существования пути»
WHERE JSON_VALUE(doc, '$.k' RETURNING Utf8) IS NULL

-- Сравнение двух JSON_VALUE с разных колонок
WHERE JSON_VALUE(doc1, '$.k' RETURNING Utf8) = JSON_VALUE(doc2, '$.k' RETURNING Utf8)

-- Вложенные JSON_* в аргументах
WHERE JSON_VALUE(JSON_QUERY(doc, '$.a'), '$.b' RETURNING Utf8) = "x"
```

Примечание

Для проверки «значение равно `false`» или «значение равно `null`» используйте фильтр JsonPath внутри `JSON_EXISTS`, например `JSON_EXISTS(doc, '$.k ? (@ == false)')` или `JSON_EXISTS(doc, '$.k ? (@ == null)')`, а не `JSON_VALUE(...) IS NULL`.

Ограничения

- JSON-индексы поддерживаются только для строковых таблиц.
- Первичный ключ таблицы должен состоять из единственной колонки целочисленного типа (`Uint64`, `Uint32`, `Int64` или `Int32`). Это временное ограничение, которое будет снято в ходе дальнейшего развития.
- В одном JSON-индексе индексируется ровно одна колонка типа `Json` или `JsonDocument`.
- Выражение `COVER` для JSON-индексов не поддерживается.
- Для таблиц с JSON-индексами не поддерживается ряд операций и механизмов модификации данных.
- Тип параметра запроса чтения из индекса не может быть обёрнут в `Optional<T>` — опциональные параметры не поддерживаются.
- Сравнение по равенству с целочисленным литералом, абсолютное значение которого превышает 2^{53}, не ускоряется индексом по значению (такие числа не помещаются в численный тип, используемый в `Json` и `JsonDocument`) и сводится к проверке существования пути.
- Приведение вещественных литералов (`Float`, `Double`) к целочисленным типам при сравнении не выполняется — такое сравнение не ускоряется индексом.

Рецепты

Готовые сценарии работы с JSON-индексом:

- JSON-индекс — быстрый старт — быстрый старт.
- Каталог со вложенными атрибутами — каталог товаров со вложенными атрибутами.
- Параметризованные запросы и переменные JsonPath — параметризованные запросы и переменные JsonPath.
- Проверка типа поля и наличия пути — проверка типа поля и наличия пути.

Связанные материалы

- Функции для работы с JSON — `JSON_EXISTS`, `JSON_VALUE`, `JSON_QUERY`, синтаксис JsonPath.
- Вторичные индексы — общие сведения о глобальных индексах и `VIEW`.
- Полнотекстовые индексы — соседний механизм, построенный поверх инвертированного индекса по словам и фразам.
- INDEX (CREATE TABLE) и VIEW (JSON-индекс) — справка по синтаксису.

Локальные индексы

Локальные индексы — вспомогательные структуры, которые хранятся вместе с данными таблицы и применяются при чтении на стороне хранилища. В отличие от глобальных индексов, они не материализуют отдельную индексную таблицу и не подключаются в запросе через `VIEW`.

В этом разделе — практические материалы по видам локальных индексов:

- Блум-индексы — Блум-индексы для ускорения селективных запросов за счёт пропуска фрагментов данных.
- min_max-индекс — min_max-индекс, пропускающий фрагменты данных по сохранённым диапазонам значений.

Блум-индексы

Блум-индексы — локальные вспомогательные структуры на основе фильтра Блума, которые ускоряют селективные запросы за счёт пропуска фрагментов данных, в которых искомое значение гарантированно отсутствует. В отличие от глобальных вторичных индексов, они работают как фильтры чтения основной таблицы и уменьшают объём данных, которые нужно фактически читать.

Типы

Поддерживаются два типа:

- `bloom_filter` — фильтр по точным значениям индексируемой колонки; подходит для условий равенства и `IN` (см. когда применять).
- `bloom_ngram_filter` — фильтр по n-граммам строковой колонки (`String` , `Utf8`); подходит для поиска подстрок и шаблонов `LIKE` в колоночных таблицах.

Локальные блум-индексы

Тип `bloom_filter` поддерживается как в строковых (OLTP), так и в колоночных (OLAP) таблицах, но реализация различается:

- В строковых таблицах фильтр строится как префиксный фильтр Блума по левому префиксу первичного ключа. Индексируемые колонки должны образовывать непрерывное ведущее подмножество колонок первичного ключа. Это ускоряет точечные чтения и сканы по диапазону, ограничивающие ведущие колонки ключа. Для создания префиксного фильтра Блума используйте ALTER TABLE ... ADD INDEX, для удаления — ALTER TABLE ... DROP INDEX.
- В колоночных таблицах фильтр строится по значениям индексируемой колонки в каждом фрагменте данных (порции). Он применяется при аналитических сканах для пропуска фрагментов, не содержащих искомое значение.

Тип `bloom_ngram_filter` поддерживается только в колоночных таблицах.

Параметры и значения по умолчанию

Полный перечень параметров `WITH (...)` и значений по умолчанию:

- `bloom_filter`
  - `false_positive_probability` — целевая вероятность ложноположительного срабатывания фильтра: доля фрагментов, которые фильтр не отсечёт, хотя искомого значения в них нет (диапазон `(0, 1)`). Меньшее значение уменьшает число лишних чтений, но увеличивает размер индекса.
    - Если параметр не указан, для строковых таблиц используется `0.0001` , для колоночных — `0.1` . Более строгий порог по умолчанию в строковых таблицах соответствует OLTP-сценарию точечных чтений; в колоночных — аналитическим сканам, где допустим больший компромисс между размером индекса и отсечением фрагментов.
  - Индексируемые колонки — типы YQL, для которых определено сравнение на равенство, кроме `Yson` , `Json` и `JsonDocument` (см. операторы сравнения). В колоночной таблице допускается одна колонка; в строковой — несколько. На строковых таблицах индексируемые колонки должны образовывать левый префикс первичного ключа.
- `bloom_ngram_filter` (колонки `String` , `Utf8` ; только колоночные таблицы)
  - `ngram_size` — размер n-граммы, целое число от `3` до `8` (по умолчанию `3`).
  - `false_positive_probability` — целевая вероятность ложноположительного срабатывания (диапазон `(0, 1)` ; по умолчанию `0.1`).
  - `case_sensitive` — учёт регистра при построении n-грамм: `true` или `false` (по умолчанию `true`).

Синтаксис создания: CREATE TABLE, ALTER TABLE ADD INDEX.

Изменение параметров после создания: ALTER INDEX.

Примеры

Создание таблицы с индексом `bloom_filter` :

```
CREATE TABLE events (
    id Uint64,
    resource_id Utf8,
    message Utf8,
    PRIMARY KEY (id),
    INDEX idx_bloom LOCAL USING bloom_filter
        ON (resource_id)
        WITH (false_positive_probability = 0.01)
);
```

Добавление `bloom_ngram_filter` к существующей таблице:

```
ALTER TABLE events
  ADD INDEX idx_ngram LOCAL USING bloom_ngram_filter
  ON (message)
  WITH (
    ngram_size = 3,
    false_positive_probability = 0.01,
    case_sensitive = true
  );
```

Создание строковой (OLTP) таблицы с префиксными фильтрами Блума по префиксам первичного ключа. В строковых таблицах индексируемые колонки должны образовывать левый префикс первичного ключа:

```
CREATE TABLE orders (
    customer_id Utf8,
    order_id Utf8,
    amount Decimal(10,2),
    PRIMARY KEY (customer_id, order_id),
```

```
    -- Префиксный фильтр Блума по первой колонке ключа
    INDEX idx_customer LOCAL USING bloom_filter
        ON (customer_id)
        WITH (false_positive_probability = 0.001),
    -- Префиксный фильтр Блума по всему первичному ключу
    INDEX idx_full_key LOCAL USING bloom_filter
        ON (customer_id, order_id)
);
```

Изменение параметров:

```
ALTER TABLE events ALTER INDEX idx_ngram SET (
    ngram_size = 4,
    false_positive_probability = 0.005,
    case_sensitive = false
);
```

Настройка параметров

Если после развёртывания индекса запросы всё ещё читают слишком много данных, попробуйте уменьшить `false_positive_probability`. Если индекс занимает слишком много места — попробуйте увеличить его.

Значения меняются через `ALTER INDEX`, см. пример.

Рекомендации:

- Начните с `false_positive_probability = 0.01`, затем корректируйте по фактическим метрикам чтения и размеру индекса.
- `ngram_size` для `bloom_ngram_filter` обычно начинают с `3`; увеличение значения может сделать фильтрацию строже для более длинных подстрок.
- Меняйте параметры по одному и сравнивайте результат на одной и той же нагрузке.

Особенности и ограничения

- Индекс всегда локальный (`LOCAL`); глобального варианта нет.
- В запросах не используется синтаксис `VIEW <index>` (в отличие, например, от полнотекстовых индексов).
- Фильтр применяется при чтении только к тем фрагментам данных, для которых при записи или слиянии порций уже сохранён блок индекса, для остальных фрагментов пропуск по этому индексу не выполняется.
- На строковых таблицах `bloom_filter` реализован как **префиксный фильтр Блума**: индексируемые колонки должны образовывать левый префикс первичного ключа. Фильтр строится по этому префиксу ключа и ускоряет точечные чтения и сканы по диапазону, ограничивающие префикс. Два индекса `bloom_filter` с одинаковой длиной префикса не допускаются.

Ограничения

- Секции `COVER (...)` и дополнительные колонки индекса не поддерживаются.
- Для колоночных таблиц индексируемая колонка должна быть одна. Для строковых таблиц допускается несколько индексируемых колонок.
- Для строковых таблиц тип индекса `bloom_ngram_filter` не поддерживается.
- На строковых таблицах индексируемые колонки `bloom_filter` должны образовывать левый префикс первичного ключа. Наборы колонок, не являющиеся префиксом, отклоняются.
- На строковых таблицах два индекса `bloom_filter` не могут иметь одинаковую длину префикса (один и тот же набор ведущих колонок первичного ключа).

Дополнительные материалы

- Вторичные индексы
- Справочник YQL: CREATE TABLE
- Справочник YQL: SELECT
- Справочник YQL: ALTER TABLE
- Быстрый старт

min_max-индекс

min_max-индекс — локальный индекс, который ускоряет сканирующие запросы с высокоселективным фильтром за счёт пропуска фрагментов. В отличие от глобальных вторичных индексов, он работает как фильтр чтения основной таблицы и уменьшает объём данных, которые нужно фактически прочитать.

Для каждого индексируемого фрагмента данных min_max-индекс хранит минимальное и максимальное значение одной колонки. Во время выполнения запроса YDB вычисляет предикат на этих двух значениях. Если из результатов вычисления следует, что предикат отфильтрует все кортежи фрагмента, фрагмент пропускается.

Примеры

Синтаксис создания: CREATE TABLE, ALTER TABLE ADD INDEX.

Создание колоночной таблицы с min_max-индексом:

```
CREATE TABLE events (
    id Uint64,
    created_at Timestamp,
    level Int32,
    resource_id Utf8,
    PRIMARY KEY (id),
    INDEX idx_created_at LOCAL USING min_max
        ON (created_at),
    INDEX idx_level LOCAL USING min_max
        ON (level)
)
WITH (
    STORE = COLUMN
);
```

Добавление min_max-индекса к существующей колоночной таблице:

```
ALTER TABLE events
  ADD INDEX idx_resource_id LOCAL USING min_max
  ON (resource_id);
```

Запросы с диапазонными предикатами могут использовать индекс для пропуска неподходящих фрагментов:

```
SELECT id, resource_id
FROM events
WHERE created_at BETWEEN Timestamp("2024-01-01T00:00:00.000000Z")
                     AND Timestamp("2024-01-02T00:00:00.000000Z");
```

Когда применять

min_max-индекс полезен, когда значения индексируемой колонки слабо меняются между соседними по первичному ключу строками таблицы: например, это временные метки, монотонно растущие идентификаторы или другие значения, коррелированные с первичным ключом.

Также min_max-индекс может быть полезен, когда предикат фильтра выбирает очень малую долю данных (порядка одной строки на миллион). Например, при запросах над таблицей логов сервиса, оставляющих только записи с уровнем `ERROR`: если сервис пишет одну ошибку на 1 000 000 записей, min_max-индекс, вероятно, существенно сократит объём чтения.

Особенности и ограничения

- Индекс всегда локальный (`LOCAL`); глобального варианта нет.
- В запросах не используется синтаксис `VIEW <index>` (в отличие, например, от полнотекстовых индексов).
- Фильтр применяется при чтении только к тем фрагментам данных, для которых при записи или слиянии уже вычислены и сохранены вместе с данными таблицы минимальное и максимальное значения индексируемой колонки; для остальных фрагментов пропуск по этому индексу не выполняется.

Ограничения

- Поддерживаются только для колоночных таблиц.
- В `ON (...)` должна быть указана ровно одна колонка.
- `COVER (...)` и дополнительные колонки данных не поддерживаются.
- Специфичные параметры `WITH (...)` не поддерживаются.
- `ALTER INDEX` для min_max-индекса не поддерживается.
- Колонки типов `Json` и `JsonDocument` не поддерживаются.

Дополнительные материалы

- Вторичные индексы
- Локальные индексы
- Справочник YQL: CREATE TABLE
- Справочник YQL: SELECT
- Справочник YQL: ALTER TABLE
- Быстрый старт

Оптимизация запросов

При выполнении запросов к базе данных текст на языке SQL преобразуется в план запроса, пригодный для распределённого выполнения на кластере. Один и тот же запрос зачастую можно выполнить множеством различных способов, и оптимизатор YDB выбирает из них наиболее эффективный.

С помощью инструментов оптимизации запросов в YDB можно:

- понять, как база данных планирует выполнять запрос
- проанализировать, как именно запрос был выполнен по факту
- выявить и использовать возможности для ускорения выполнения

> **Примечание**
>
> В примерах этого раздела используется набор данных TPC-H. Схема таблиц и связи между ними описана в спецификации TPC-H. Некоторые примеры являются запросами из бенчмарка с адаптацией под YDB, другие специально подготовлены для иллюстрации излагаемого материала. Исходный текст SQL всех примеров позволяет самостоятельно повторить все описанные примеры.

Как база данных планирует выполнять запрос

После компиляции и оптимизации строится план выполнения запроса — последовательность операторов, которые необходимо выполнить для получения результата. Этот план можно получить при помощи параметра `--explain` команды CLI `ydb sql` без реальной обработки данных. Этот план будет отражать структуру выполнения запроса и содержать оценки стоимостного оптимизатора, полученные на основании существующей **статистики источников данных**, такие как количество строк в таблице и её размер в байтах.

Этот план выдаётся в формате JSON, однако для изучения его удобнее представлять в виде визуализации. Доступные варианты зависят от того, как выполнялся запрос (SDK, CLI, UI) — см. План выполнения запроса.

Как на самом деле был выполнен запрос

Для более детального и предметного анализа выполнения запроса YDB собирает **статистику выполнения**:

- длительность отдельных шагов (операторов)
- объёмы переданных данных
- ожидания, возникшие при взаимодействии отдельных узлов распределённой системы

Каждый шаг плана выполняется как набор задач, параллельно на множестве узлов. Детальная информация по каждой отдельной задаче занимала бы слишком большой объём, вместо этого она обрабатывается сервером и для каждого набора одинаковых задач публикуется в агрегированном виде, обогащая план в формате JSON.

Но даже в агрегированном виде **статистика выполнения** всё ещё является достаточно объёмной для самостоятельного анализа. Чтобы решить эту проблему, разработан специальный способ визуализации этой статистики вместе с планом в формате SVG. Как получить такой план читайте в статье План выполнения запроса; как его интерпретировать, читайте в разделе «Графический план запроса»: расположение информации, структура и визуализация метрик.

Что можно сделать для ускорения работы запроса

В этом разделе основной фокус сделан на анализе и интерпретации планов выполнения запросов: рассматриваются способы понять, как сервер планирует и реализует выполнение, а также как выявить узкие места с помощью изучения плана и сопутствующих метрик. Практические аспекты ускорения работы разбираются через нахождение потенциальных точек оптимизации на основе анализа этих данных.

Структура раздела

- План выполнения запроса
  - Обзор режимов
  - Получение EXPLAIN плана через CLI и SDK
  - Получение EXPLAIN плана в UI
  - Получение плана при выполнении запроса через CLI
  - Получение плана при выполнении запроса в UI
  - Получение графического плана при выполнении запроса через CLI
  - Получение графического плана при выполнении запроса в UI
- Графический план запроса
  - Расположение информации в плане запроса
    - Отдельные регионы (колонки)
    - Колонка ResultSet, Sink и Precompute
    - Операторная статистика
    - Число задач стадии и прогресс их выполнения
    - Метрики по стадиям
    - Временные графики
    - Интерактивные возможности
  - Структура фактического плана запроса
    - Стадии
    - Каналы связи
    - Объединение стадий
    - Множественные выходы
    - Составные структуры
  - Визуализация метрик запроса
    - Общий вид визуализации
    - Параллельность
    - Агрегаты
    - Масштаб метрик

        - Перекос данных
        - Перекос времени
        - Потребление CPU
        - Потребление памяти
    - Подсказки оптимизатора
    - Параметризованные запросы и повторная компиляция

План выполнения запроса

План выполнения запроса — подробное описание того, как сервер планирует выполнить пользовательский запрос после компиляции и оптимизаций. План запроса помогает заранее оценить порядок операторов и ожидаемые объёмы данных. План запроса можно получить без фактического выполнения запроса через CLI или UI.

Когда запрос выполняется, план дополнительно обогащается фактической статистикой выполнения и даёт больше данных для анализа.

Обзор режимов

Цель	Режим	CLI	UI	Раздел
Оценка плана без выполнения запроса	EXPLAIN	`ydb sql --explain`	кнопка **Explain**	EXPLAIN через CLI и SDK, EXPLAIN в UI
План с фактическими метриками выполнения	EXPLAIN ANALYZE	`ydb sql --explain-analyze`	кнопка **Run** с настройкой Statistics collection mode=Full или кнопка **Explain Analyze**	ANALYZE через CLI, ANALYZE в UI
Графический план в формате SVG	SVG	`ydb sql --explain-analyze --format svg`	кнопка **Run** с настройкой Statistics collection mode=Full или кнопка **Explain Analyze**, затем **Open Execution Plan**	SVG через CLI, SVG в UI

Получение плана выполнения запроса в режиме EXPLAIN через CLI и SDK

Рассмотрим простой запрос:

```
SELECT n_name
  FROM nation
  JOIN region ON nation.n_regionkey == region.r_regionkey
  WHERE r_name = "AMERICA"
```

Чтобы узнать план этого запроса, не выполняя его, необходимо выполнить команду `ydb sql` с указанием параметра `--explain` (предполагается, что текст запроса находится в файле `file1.sql`):

```
ydb sql -f file1.sql --explain
```

В терминале будет выведено следующее:

```
┌─────────┬─────────┬─────────┬──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┐
│ E-Cost  │ E-Rows  │ E-Size  │ Operation                                                                                                            │
├─────────┼─────────┼─────────┼──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┤
│         │         │         │ ┌> ResultSet                                                                                                         │
│ 2003    │ 3       │ 457     │ └─┬> InnerJoin (Map) (nation.n_regionkey = region.r_regionkey)                                                       │
│ 0       │ 25      │ 1212    │   ├──> TableFullScan (Table: nation, ReadColumns: ["n_nationkey (-∞, +∞)","n_name","n_regionkey"])                   │
│ 0       │ 1       │ 67      │   └─┬> Filter (Blocks: True, r_name == "AMERICA", Pushdown: True)                                                    │
│ 0       │ 5       │ 672     │     └──> TableFullScan (Table: region, ReadColumns: ["r_regionkey (-∞, +∞)","r_name"])                             │
└─────────┴─────────┴─────────┴──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┘
```

В колонке `Operation` показана структура запроса в виде дерева операторов. Здесь видно, что сервер планирует следующую последовательность выполнения (снизу вверх):

- прочитать таблицу `region`
- применить предикат (фильтр) к полю `r_name` со значением `AMERICA`
- соединить результат с содержимым таблицы `nation`

В других колонках — прогнозные (estimated, отсюда E-Cost, E-Rows и т.п.) оценки оптимизатора:

Колонка	Назначение
`E-Cost`	Оценка стоимости фрагмента плана
`E-Rows`	Ожидаемое число строк
`E-Size`	Ожидаемый объём данных

Оценки стоимостной оптимизатор выводит из **статистики источников данных** и структуры запроса.

В примере выше показан человекочитаемый табличный вывод. Тот же план в JSON можно получить, вызывая запрос с параметром `--format json-unicode` (или при вызове через SDK). Пример:

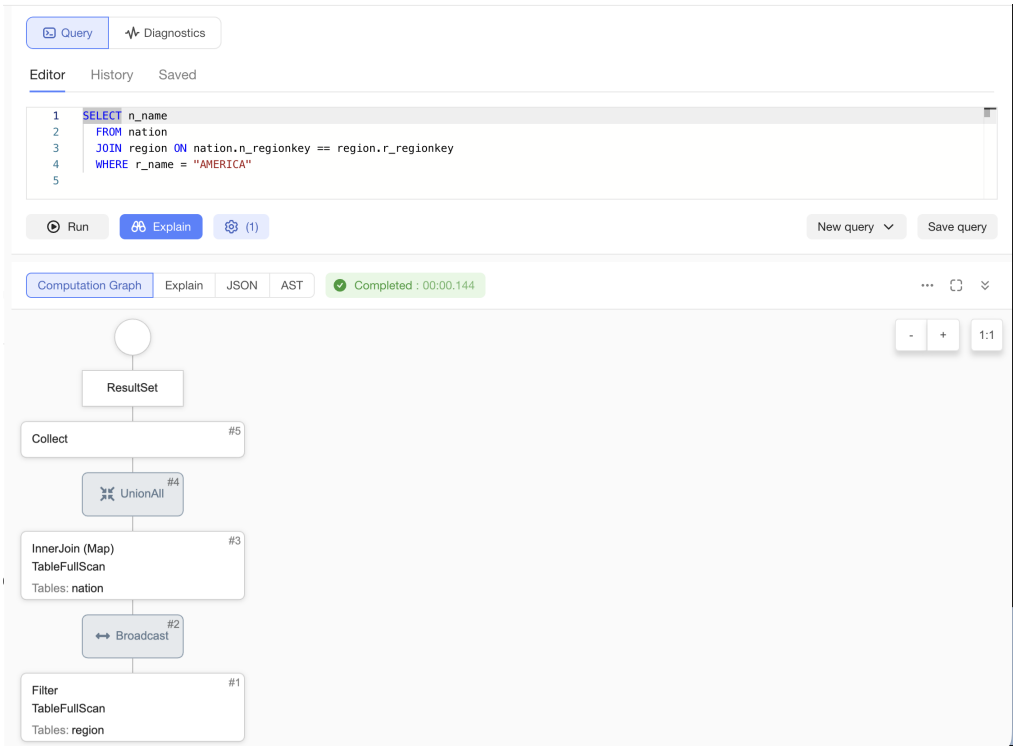
```
ydb sql -f file1.sql --explain --format json-unicode
```

Вывод достаточно объёмный и здесь приведён лишь частично:

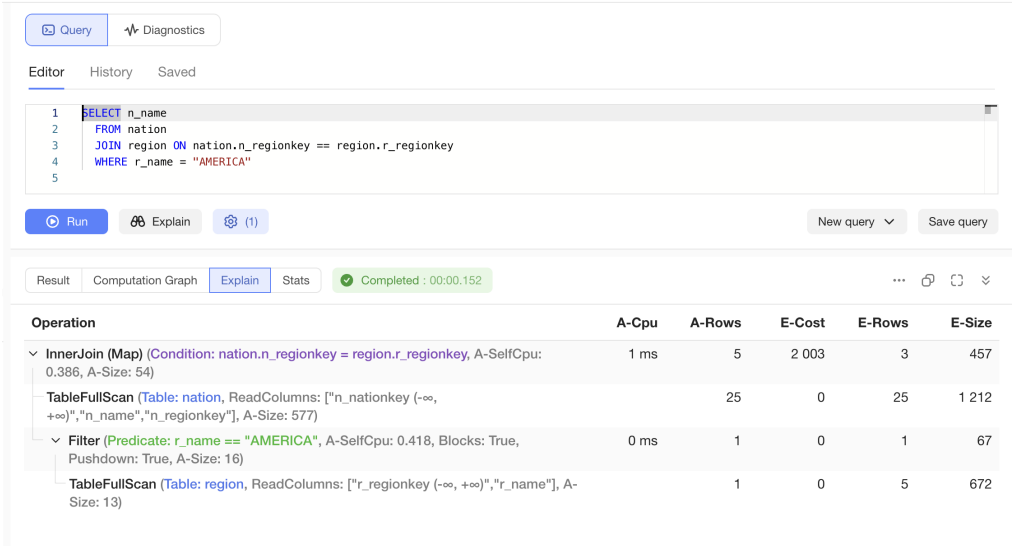
```
{
    "Plan" : {
        "Plans" : [
            {
                "PlanNodeId" : 6,
                "Plans" : [
                    {
"..."
                    }
                ],
                "Node Type" : "ResultSet",
                "PlanNodeType" : "ResultSet"
            }
        ],
        "Node Type" : "Query",
        "PlanNodeType" : "Query"
    }
}
```

Получение плана выполнения запроса в режиме EXPLAIN в UI

В графическом интерфейсе YDB также можно получить план выполнения, для этого надо использовать кнопку `[Explain]`. В UI доступно больше вариантов плана. По умолчанию показывается закладка `Computation Graph`, граф вычислений в виде операторов и связей между ними:



Если переключиться на вкладку `Explain`, вы увидите план, похожий на вывод CLI:



Оставшиеся вкладки позволяют также исследовать исходную структуру плана в формате `JSON` на одноимённой вкладке, а также ознакомиться с низкоуровневым форматом, в который компилируется программа на вкладке `AST`.

Получение фактического плана выполнения запроса через CLI

Чтобы получить фактический план выполнения запроса, следует запустить команду `ydb sql` с опцией `--explain-analyze`. Вывод в терминале при этом несколько изменится — в дополнение к оценкам оптимизатора появятся актуальные (actual, отсюда A-CPU, A-Rows) метрики, собранные во время работы запроса:

```
┌───────┬────────┬────────┬────────┬────────┬───────────────────────────────────────────────────
──────────────────────────────────────────────────┐
│ A-Cpu │ A-Rows │ E-Cost │ E-Rows │ E-Size │ Operation
│
├───────┼────────┼────────┼────────┼────────┼───────────────────────────────────────────────────
──────────────────────────────────────────────────┤
│       │        │        │        │        │ ┌> ResultSet
│
│ 1     │ 5      │ 2003   │ 3      │ 457    │ └─┬> InnerJoin (Map) (A-SelfCpu: 0.367, nation.n_regionkey = re
gion.r_regionkey, A-Size: 54)                    │
│       │ 25     │ 0      │ 25     │ 1212   │   ├─> TableFullScan (Table: nation, ReadColumns: ["n_nationkey
(-∞, +∞)","n_name","n_regionkey"], A-Size: 577) │
│ 0     │ 1      │ 0      │ 1      │ 67     │   └─┬> Filter (r_name == "AMERICA", A-SelfCpu: 0.333, Blocks: T
rue, Pushdown: True, A-Size: 16)                 │
│       │ 1      │ 0      │ 5      │ 672    │     └─> TableFullScan (Table: region, ReadColumns: ["r_regionk
ey (-∞, +∞)","r_name"], A-Size: 13)              │
└───────┴────────┴────────┴────────┴────────┴───────────────────────────────────────────────────
──────────────────────────────────────────────────┘
```

Получение фактического плана выполнения запроса в UI

Аналогично, фактический план выполнения запроса доступен в UI при обычном выполнении кнопкой `[Run]` с включенными настройками Statistics collection mode = Full или при нажатии на кнопку `[Explain Analyze]`. Как и в случае с CLI, добавляются актуальные метрики — вывод похож на рисунок в разделе Получение EXPLAIN плана в UI.

Получение графического плана запроса через CLI

Фактический план выполнения запроса в формате SVG можно получить посредством утилиты командной строки. Для этого надо изменить формат вывода плана, добавив опцию `--format svg`:

```
ydb sql -f file1.sql --explain-analyze --format svg > plan1.svg
```

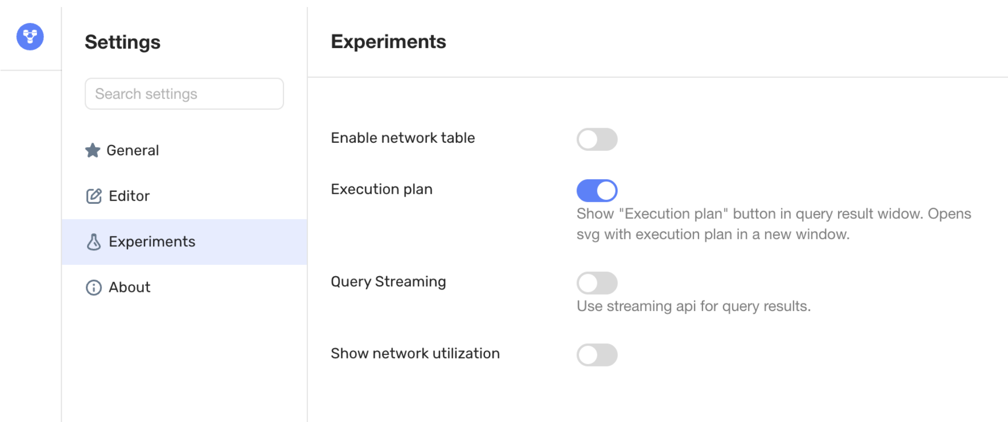
Формат SVG сам по себе текстовый, но предназначен для отображения специальными средствами просмотра, поэтому обычно имеет смысл сохранить полученный план в файл, как в показанном примере, а потом открыть его, например, в браузере.

Опция `--format svg` может быть использована и вместе с опцией `--explain` тоже. Так как в этом случае запрос реально не выполняется, то никакой статистики не будет, поэтому в плане будет показана только часть, связанная со структурой запроса.

Получение графического плана запроса в UI

Чтобы в YDB UI отображался графический план запроса, включите настройку `Experiments | Execution plan`.

> **Примечание**
>
> Графический план запроса в UI пока относится к экспериментальным возможностям (раздел `Experiments`). Со временем он должен перейти в обычные настройки интерфейса.



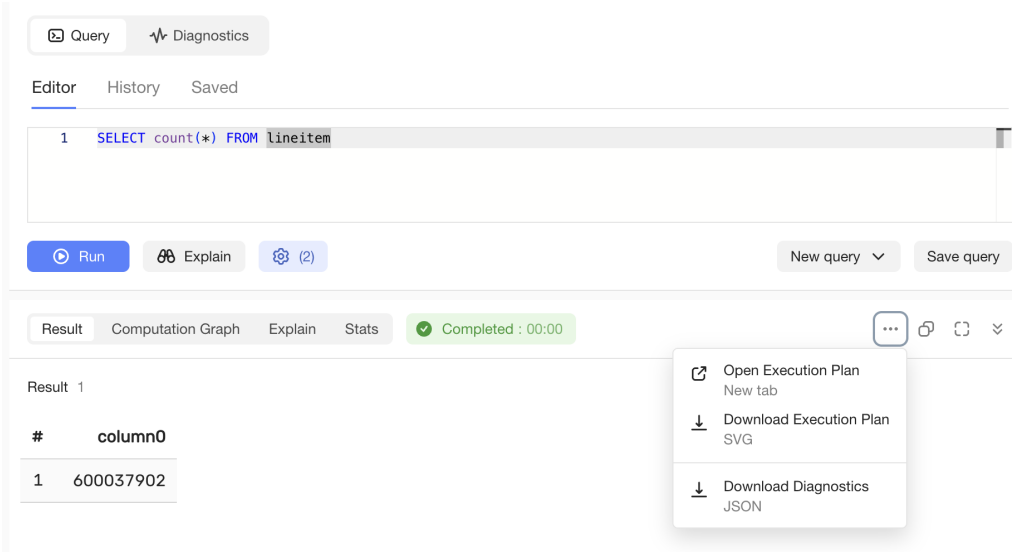
После этого выполните запрос, например:

```
SELECT count(*) FROM lineitem
```

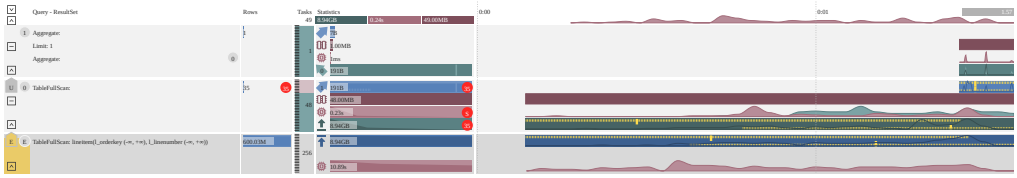
В дополнительном меню справа (кнопка [...], отображается на всех вкладках интерфейса) появятся пункты:

- `Open Execution Plan` — открыть фактический план запроса в новой вкладке браузера;
- `Download Execution Plan` — сохранить фактический план запроса в файл.

Там же есть пункт `Download Diagnostics`.



Для этого запроса он будет выглядеть примерно так (конкретные значения метрик зависят от конфигурации и нагрузки кластера):



Об элементах диаграммы подробнее — в разделе «Графический план запроса»: Расположение информации в плане запроса, Структура фактического плана запроса и Визуализация метрик запроса.

Расположение информации в плане запроса

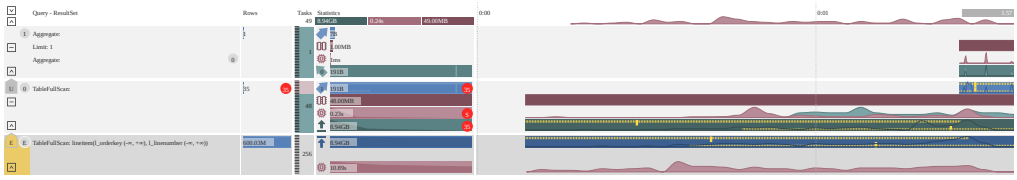
Графический SVG-план запроса с метриками получают после фактического выполнения запроса — см. Получение графического плана при выполнении запроса через CLI и в UI. Ниже разобраны колонки диаграммы.

Во встроенных в документацию изображениях нет интерактивных элементов — откройте план в отдельной вкладке браузера, чтобы пользоваться сворачиванием стадий и всплывающими подсказками.

Отдельные регионы (колонки)

Ниже показан план простого запроса:

```
SELECT count(*) FROM lineitem
```



План выводится в виде таблицы: слева — структура стадий, справа — колонки с метриками и графиками.

Колонка	Назначение	Подробнее
`Query - ...`	Структура стадий и операторов	Структура плана
`Rows`	Статистика по операторам стадии	Операторная статистика
`Tasks`	Число задач и прогресс выполнения	Число задач стадии и прогресс их выполнения
`Statistics`	Суммарные метрики по стадии	Метрики по стадиям, Визуализация метрик запроса
Временная шкала	Графики метрик во времени	Временные графики, Визуализация метрик запроса

`Query - ...` -- структура плана

Самая левая колонка — структура стадий плана. Каждая строка таблицы соответствует одной стадии; операторы внутри стадии — шаги обработки данных (подробнее — в Структуре фактического плана запроса).

Заголовок начинается с общего префикса `Query`, вторая часть заголовка зависит от типа выполняемого запроса:

- `ResultSet` (как в примере выше) — запросы с выборкой (`SELECT`);
- `Sink` — модифицирующие запросы (`INSERT`, `UPDATE`, `DELETE`);
- `Precompute` — отдельно выполняемые фрагменты плана, см. Составные структуры.

`Rows` -- операторная статистика

Стадия состоит из одного или нескольких операторов. Если хотя бы один физический оператор отдаёт статистику, колонка `Operators` заполнена; у стадий без отчётности по операторам ячейка пуста. Содержимое колонки дополняет картину по структуре стадии и агрегированным метрикам в `Stages`.

`Tasks` -- число задач стадии и прогресс их выполнения

В YDB запрос выполняется распределённо: одна и та же стадия может параллельно обрабатываться несколькими задачами на одном или нескольких узлах кластера.

В колонке `Tasks` указано число задач, запланированных для стадии. При наведении курсора во всплывающей подсказке видно, сколько из них уже завершилось. Слева в той же колонке прогресс показан пунктирной полосой.

Цвет фона колонки отражает интенсивность потребления CPU стадией; см. Потребление CPU.

`Statistics` -- метрики по стадиям

Колонка `Statistics` — агрегированные метрики по всем задачам стадии. Часть метрик описана в Визуализация метрик запроса. Общее свойство: это рассчитанные значения в некоторый момент времени (как правило, в конце выполнения запроса).

Одинаковые метрики по разным стадиям приводятся к одному масштабу, чтобы их можно было быстро сравнивать между собой, например, потребление CPU или памяти. Подробнее — Масштаб метрик.

Временная шкала

Колонка с условным названием `Timeline` (это название не выводится в плане, на этом месте отображаются отметки времени) показывает те же самые величины, что и `Statistics`, но на всём интервале выполнения запроса. Шаг временной сетки подбирается автоматически. Длительность запроса указана в сером прямоугольнике в правом верхнем углу колонки.

Интерактивные возможности

Откройте план запроса в отдельной вкладке браузера. У крупного запроса план занимает много строк и его удобно упрощать на экране:

- У каждой стадии кнопка сворачивает поддерево, начиная с этой стадии (кнопка меняется на для разворота).
- Кнопка переводит стадию в режим `slim`: по высоте остаётся одна метрика (`Output`). Обычный вид возвращается при любом действии на стадии.
- В левом верхнем углу плана кнопка включает режим `slim` для всех стадий, кнопка возвращает полный вид и разворачивает все поддеревья.

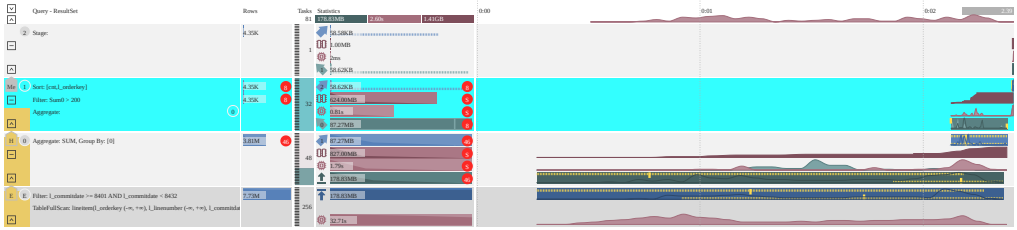
Структура фактического плана запроса

На диаграмме левая колонка показывает структуру стадий и операторов; ниже — их семантика, каналы связи и нетривиальные топологии.

Стадии

Ниже представлен запрос и его графическое представление:

```
SELECT count(*) cnt, l_orderkey, sum(l_quantity)
  FROM lineitem
  WHERE l_commitdate >= date("1993-01-01") and l_commitdate < date("1993-02-01")
  GROUP BY l_orderkey
  HAVING sum(l_quantity) > 200
  ORDER BY cnt DESC, l_orderkey
```



В примере три вычислительных (compute) стадии, пронумерованные как `0`, `1` и `2`, а также отдельная стадия у таблицы `lineitem` (стадии чтения не нумеруются). Любую стадию можно временно выделить на схеме; здесь выделена стадия `1`.

Интерактивные элементы на встроенных изображениях недоступны — см. Расположение информации в плане запроса.

На схеме — структура выполнения запроса, полученная после его компиляции: физические операторы, сгруппированные в стадии выполнения.

Стадия — основная единица плана; задачи стадии выполняются параллельно, часто на разных узлах. Данные передаются от стадии к стадии снизу вверх: от хранилища к результату.

Стадии бывают двух типов:

1. Стадии чтения (scan), которые выполняются в хранилище (колоночные таблицы или строковые таблицы) имеют тёмный фон.
2. Вычислительные стадии (compute) отображаются на светлом фоне и нумеруются последовательно, начиная с `0`.

> **Примечание**
>
> Нумерованный список ниже на странице идёт **сверху вниз**, на графике стадии и операторы расположены **снизу вверх**. Сопоставлять пункты со схемой удобно, начиная с **нижней** части плана.

Из плана мы можем понять следующее:

1. Таблица `lineitem` сканируется целиком, потому что условие `WHERE` не использует первичный ключ.
2. Фильтр по полю `l_commitdate` выполняется непосредственно в хранилище (pushdown предиката), это позволяет на раннем этапе отбросить ненужные данные и не пересылать их далее.
3. В стадии `0` выполняется предварительная агрегация с группировкой по `l_orderkey` (конструкция `GROUP BY` в тексте запроса).
4. Данные перераспределяются (hash shuffle) в стадию `1`; подробнее — в разделе Каналы связи.
5. В стадии `1` находится финальная агрегация, фильтр обозначенный в запросе в части `HAVING` и локальная (для каждого узла) сортировка.
6. Стадия `2` объединяет результат обработки на каждом из узлов в глобально упорядоченную последовательность строк и возвращает клиенту в качестве результата работы запроса.

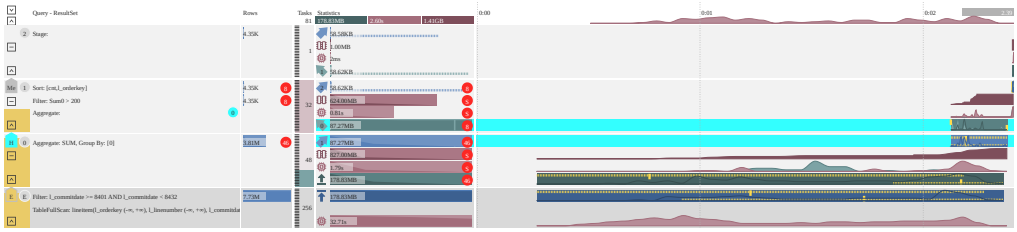
> **Совет**
>
> Неочевидный элемент схемы можно подсветить курсором: у большинства узлов есть всплывающая подсказка. Это не замена разделам документации, но удобно для быстрой ориентации и метрик.

Каналы связи

Канал связи на диаграмме показан пятиугольником с направлением потока. В этом примере выделен канал ⬟ между стадиями `0` и `1`.

У линейного плана из четырёх стадий три соединения:

1. `E` — получение данных из хранилища (scan).
2. `H` — hash shuffle, разбиение по одному или нескольким полям (список полей — во всплывающей подсказке по пятиугольнику).
3. `Me` — слияние (merge) уже отсортированных потоков; для этого типа указываются поля сортировки.



Соединение состоит из двух частей:

1. **Выходной канал** — исходящий трафик из стадии `0`. Синяя стрелка вправо-вверх с номером стадии-получателя (здесь `1`).
2. **Входной канал** — приём в стадию `1`. Зелёная стрелка влево-вверх с номером стадии-отправителя (`0`).

На графике **успешно** завершённого запроса объём данных на выходе канала и на входе должен совпадать: YDB гарантирует доставку между узлами без потерь и дубликатов.

Пока запрос выполняется, показатели входа и выхода могут расходиться из-за задержки буферов и асинхронного обновления статистики. На графике ошибочного запроса расхождения тоже возможны.

Входной канал сопоставляется оператору, который принимает данные. В простом случае у стадии один вход: поток из канала подаётся в первый по порядку обработки (нижний в списке) оператор. В сложных структурах выполнения запроса зачастую несколько входов; у оператора справа дублируется номер стадии-источника исходящего канала (здесь `0`), он подсвечивается при выборе канала. Разбор на примере с `JOIN` — в разделе ниже.

Что можно понять для выбранного канала между стадиями `0` и `1`:

1. Начинается в стадии `0`, направлен в стадию `1`.
2. Проходит через hash shuffle.
3. Входит в стадию `1` со стадии `0`.
4. Подаётся в оператор агрегации.

Стадии чтения (scans) и вычисления (compute) связаны аналогичными каналы связи, имеющими немного другой набор метрик, чем каналы, связывающие две вычислительных (compute) стадии (см. выше). Планировщик YDB по возможности размещает связанные задачи на одном и том же узле, чтобы сократить пересылаемый по сети объём данных и ускорить выполнения запроса.

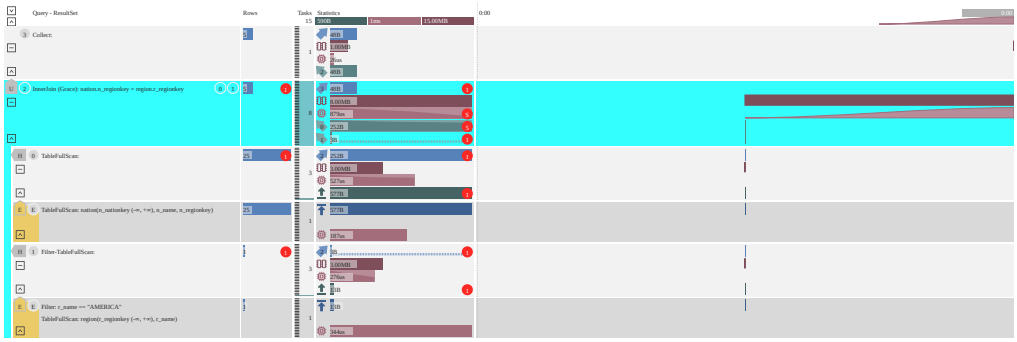
Объединение стадий

Ниже — запрос с `JOIN` двух таблиц.

```
PRAGMA ydb.OptimizerHints = 'JoinType(nation region shuffle)';

SELECT n_name
  FROM nation
  JOIN region ON nation.n_regionkey == region.r_regionkey
  WHERE r_name = "AMERICA"
```

Таблицы `region` и `nation` небольшие (5 и 25 строк), запрос быстрый, трафик небольшой, но структура плана показательна. Для примера задан **Shuffle Join** через подсказку оптимизатора. Без этой прагмы план будет другим: при малом объёме данных стоимостной оптимизатор выберет иной план (его можно сравнить, убрав `PRAGMA`).



Стадия `2` (на схеме изначально выделена) получает данные от двух стадий с номерами `0` и `1`. Стадии расположены по вертикали, и направление движения данных — снизу вверх, однако связи между ними не обязательно последовательные. Если у стадии есть несколько предшественников, слева появляется отступ, а входные `H`-каналы обозначаются пятиугольником ⯀ с направлением влево; в этом примере таких соединений два.

Данные объединяются в физическом операторе InnerJoin (эквивалент `JOIN` в тексте запроса). Указаны ключи `n_regionkey` и `r_regionkey`; справа — номера стадий `0` и `1`.

Красные круги с числами `1` и `5` — напоминание из раздела Агрегаты: число ненулевых метрик (здесь — по трафику в каналах) меньше числа задач стадии. Часть задач не получила данных, частая причина этого — перекос данных или избыточный параллелизм.

На этой схеме следует обратить внимание на два типичных случая, которые вызывают перекос по данным (`data skew`) и, таким образом, замедляют выполнение запроса, потому что те задачи, которые получили меньший объём трафика, завершатся быстрее и будут ждать другие, которым досталось больше данных:

1. Неравномерное распределение данных в хранилище

В столбце `Tasks` видно, что у стадий чтения указан параллелизм 1 (по одному шарду на таблицы `region` и `nation`). У связанных с ними вычислительных (`compute`) стадий с номерами `0` и `1` планировщиком создано по 3 задачи. Все

прочитанные из каждой таблицы строки были сгруппированы в один пакет (`batch`) и отправлены из колоночного шарда в **одну** из трёх задач (случайную). Оставшимся задачам ничего не досталось, поэтому они не отчитались в статистике по этому каналу.

Когда в таблице данных больше, и пересылка использует более одного сообщения, то обычно данные распределяются равномерно между вычислительными задачами. Однако, если при хранении данных возникает перекос (одни шарды содержат больше данных чем другие), то похожая неравномерность также может проявляться и плохо влиять на производительность, поэтому она выделяется на схеме, чтобы обратить на это внимание.

2. Неравномерное распределение ключей

Даже если данные в хранилище распределены равномерно, они перераспределяются в процессе последующей обработки, что также может вызывать перекос, вызванный алгоритмическими особенностями реализации.

Соединение **hash shuffle** используется для корректной обработки больших объёмов данных несколькими задачами одновременно. Смысл этого действия заключается в том, чтобы разделить всё множество строк на несколько непересекающихся групп таким образом, чтобы строки с одинаковыми значениями определённых колонок оказались бы в одной группе. Тогда каждая задача может независимо от других обрабатывать свою группу и получить правильный результат. Такие колонки ещё называют ключевыми, потому что они используются в качестве ключей для **hash shuffle** и последующих операторов.

В данном примере **hash shuffle** используется для того, чтобы сгруппировать необходимые строки в задаче, реализующей оператор `JOIN` . Он реализован как вычисление значения хеш-функции от всех ключевых колонок и далее остатка от деления этого значения на общее количество таких задач.

Но общее количество различных значений (кардинальность) хеш-функции не может превышать общего количества различных ключей в наших данных, поэтому даже если данных изначально много и все они распределены в хранилище равномерно, но количество ключей невелико и сильно отличается для разных значений ключей, у нас снова возможен перекос, который мы наблюдаем на этом плане.

После применения условия `WHERE` из таблицы `region` остаётся всего одна строка, которая естественным образом попадает только в одну из созданных задач, о чём сигнализирует красный круг с отметкой `1` на входе стадии `2` (мы наблюдаем низкую кардинальность по ключу `r_regionkey`).

Из таблицы `nation` выбираются все 25 строк. Однако для ключа `n_regionkey` у нас есть всего 5 разных значений. Поэтому из всех 8 задач стадии `2` данные из таблицы `nation` приходят только в 5 (остальным задачам ничего не досталось). Все 8 задач выполняют оператор `JOIN` , однако непустой результат будет только у одной из них — именно той, в которую попала строка из таблицы `region` . Эта задача вернёт 5 выбранных строк таблицы `nation` , связанных с данным регионом. Поэтому на выходе стадии `2` тоже появляется красная отметка `1` .

Множественные выходы

> **Примечание**
>
> В примерах этого раздела используется набор данных TPC-H. Схема таблиц и связи между ними описана в спецификации TPC-H. Некоторые примеры являются запросами из бенчмарка с адаптацией под YDB, другие специально подготовлены для иллюстрации излагаемого материала. Исходный текст SQL всех примеров позволяет самостоятельно повторить все описанные примеры.

Следующий случай выполнения запроса: одна таблица читается один раз, а результат расходится к двум (или более) потребителям, выше потоки снова сходятся.

Такую структуру удобно показывать **клонами** стадии: один экземпляр — основной, остальные — сокращённое представление и отдельный выход. Пример — TPC-H Q17, адаптированный под синтаксис YDB:

```
SELECT SUM(l_extendedprice) / 7.0 AS avg_yearly
  FROM lineitem
  CROSS JOIN part
  CROSS JOIN (
    SELECT l_partkey, 0.2 * AVG(l_quantity) AS quantity_threshold
      FROM lineitem
      GROUP BY l_partkey
    ) AS threshold
    WHERE part.p_partkey = lineitem.l_partkey
      AND p_brand = 'Brand#35'
      AND p_container = 'LG DRUM'
      AND l_quantity < quantity_threshold
      AND part.p_partkey = threshold.l_partkey
```

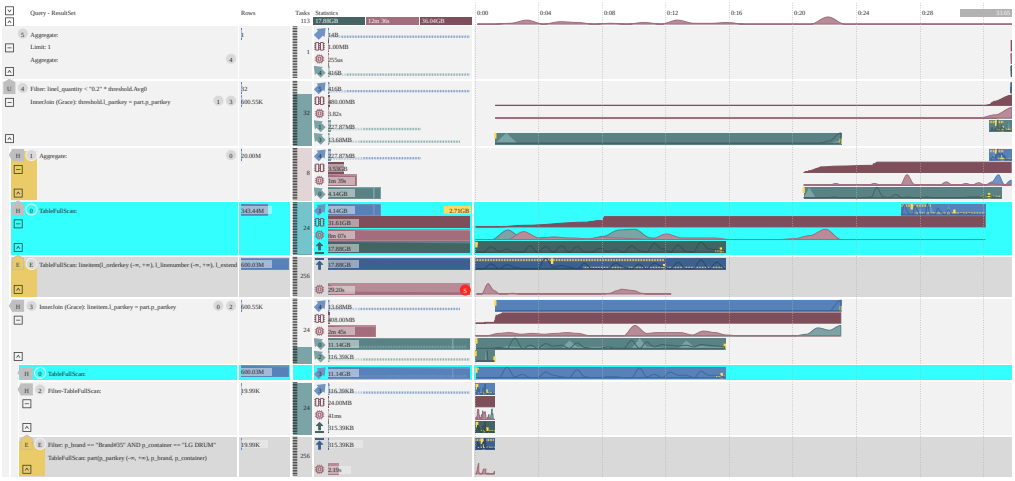


Таблица `lineitem` читается один раз в стадии `0` и используется в стадиях `1` и `3`. На плане стадия `0` показана дважды:

- **Основной экземпляр** — полный набор метрик: входы, CPU, память.
- **Клоны** — тот же цвет, что у стадий чтения, и одна метрика: выход, ведущий в нужную нижестоящую стадию.

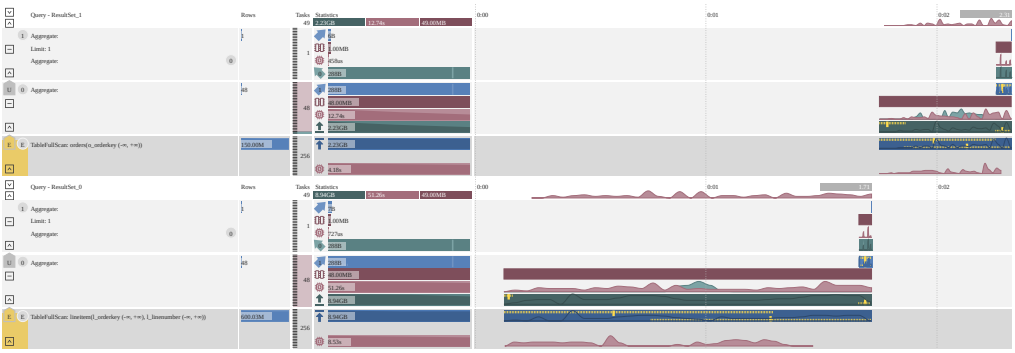
Выбор любого экземпляра стадии `0` подсвечивает все её копии; но выходы выбираются независимо друг от друга, чтобы можно было понять, куда идёт каждый из них.

Стоимостной оптимизатор может объединять и дедуплицировать не только чтение из хранилища, но и целые подструктуры — это хорошо видно, например, на TPC-H Q21.

Составные структуры

В примерах выше была одна структура выполнения. Если в одном вызове можно передать несколько выражений SQL, то сервер выполнит их по очереди, и на диаграмме появятся **несколько несвязанных** структур, каждая из которых представляет отдельное выполнение запроса.

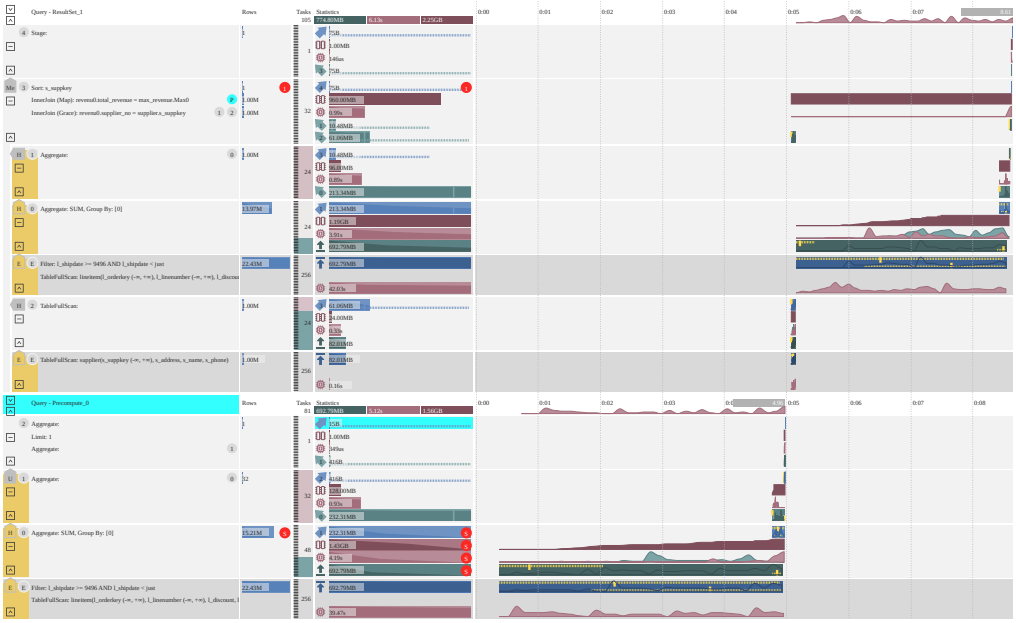
```
SELECT count(*) FROM lineitem;
SELECT count(*) FROM orders;
```



Бывают случаи, когда **одно** SQL-выражение приводит к появлению сразу нескольких ветвей вычислений: стоимостной оптимизатор может вынести часть операций в отдельный прекомпьют — самостоятельную цепочку, создающую промежуточный результат для последующего использования в основном процессе выполнения запроса. Например, так устроена TPC-H Q15 в синтаксисе YDB:

```
$revenue0 = (
  SELECT l_suppkey AS supplier_no, sum(l_extendedprice * (1 - l_discount)) AS total_revenue
    FROM lineitem
    WHERE l_shipdate >= date('1996-01-01') AND l_shipdate < date('1996-01-01') + interval('P90D')
    GROUP BY l_suppkey
);

SELECT s_suppkey, s_name, s_address, s_phone, total_revenue
  FROM supplier
  CROSS JOIN $revenue0 AS revenu0
  CROSS JOIN (
    SELECT max(total_revenue) AS max_total_revenue
      FROM $revenue0
    ) as max_revenue
  WHERE s_suppkey = supplier_no AND total_revenue = max_total_revenue
  ORDER BY s_suppkey;
```



Связь прекомпьюта с основной структурой выполнения становится очевидной при выделении имени прекомпьюта или его выходного канала с результатом: одновременно подсвечивается и вход оператора, который использует этот результат. Такой вход отмечен символом `P`. В данном примере это стадия `3` в верхней структуре; данные из прекомпьюта поступают в правую часть Map Join.

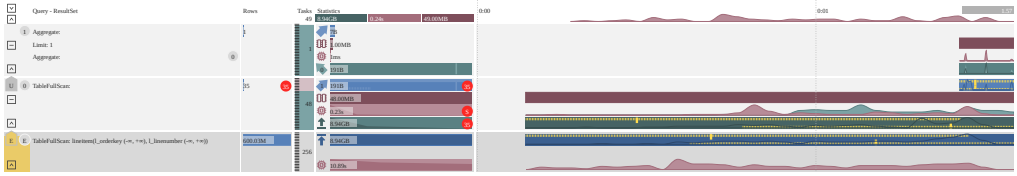
В тяжёлых запросах (в том числе из TPC-DS) прекомпьютов несколько; они могут выполняться последовательно или параллельно, результаты подмешиваются в следующие стадии или порождают новые прекомпьюты. Формат плана рассчитан на то, чтобы по схеме можно было восстановить реальную структуру выполнения и соответствие задач на кластере YDB.

Визуализация метрик запроса

YDB — распределённая СУБД для больших объёмов данных. Кластер может включать много узлов; структура запроса тогда состоит из сотен и тысяч задач, каждая со своими метриками. Показать все значения по отдельности и по ним же делать выводы нереалистично, поэтому метрики задач **агрегируются по стадии**, попадают в ответ сервиса и на диаграмму уже в агрегированном виде. На SVG-плане они показаны в колонках `Tasks`, `Statistics` и `Timeline` — см. Расположение информации в плане запроса. Ниже — как устроены агрегирование и визуализация и как по ним быстро оценить план и узкие места.

Общий вид визуализации

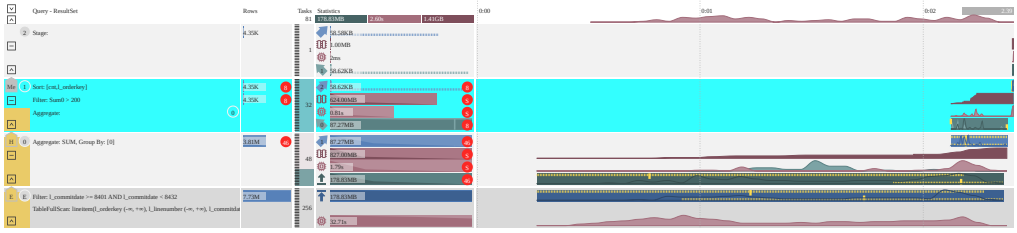
Графический план объединяет структуру запроса слева и агрегированные метрики справа:



Колонки диаграммы описаны в Расположении информации в плане запроса.

Параллельность

Число параллельно выполняющихся задач показано в колонке `Tasks`. Для стадий чтения из хранилища оно совпадает с числом шардов. Слева в той же колонке полосатый фон («зебра») отражает долю уже завершившихся задач. На финальном графике завершённого запроса полоса занимает всю высоту строки стадии; для снимка «в процессе» картина иная. Точные числа — во всплывающей подсказке при наведении на область.



> **Примечание**
>
> Цвет фона колонки `Tasks` зависит от интенсивности использования CPU и от простоев задач в стадии; см. следующий раздел.

Агрегаты

Для метрик в статистике вычисляются и передаются в ответ те же агрегаты, что и в SQL:

- `MIN` — минимум;
- `MAX` — максимум;
- `COUNT` — число значений;
- `SUM` — сумма;
- `AVG` — среднее (из `COUNT` и `SUM`).

Если `COUNT` меньше числа задач стадии (часть задач не отдала метрику), это помечается как аномалия: рядом с метрикой появляется красный круг с числом `COUNT`. Подробности — во всплывающей подсказке.

Если `COUNT` совпадает с числом задач (типичный случай), отдельный индикатор не рисуется. По умолчанию показывается `SUM`; при наведении — развёрнуто: `SUM, MIN | AVG | MAX`. Если метрику сообщила одна задача, `MIN`, `AVG` и `MAX` скрыты, остаётся `SUM` (она совпадает с единственным значением).

Дополнительные правила отображения:

- для целочисленных счётчиков (число строк) используются суффиксы `K`, `M` — множители 10^3, 10^6 и т.д.;
- для объёма (данные, память) — `KB`, `MB` и т.д. по степеням двойки: 2^{10}, 2^{20} …;
- длительности — в удобочитаемом виде часы / минуты / секунды.

Масштаб метрик

В колонке `Statistics` для каждой стадии сверху вниз выводятся метрики:

- `Egress` (тёмно-синий) — вывод данных через границу подсистемы (из хранилища в вычисление и обратно);
- `Output` (синий) — передача в следующую вычислительную стадию или в результат;
- `Memory` (тёмно-песочно-красный) — память;
- `CPU` (песочно-красный) — процессорное время;
- `Input` (зелёный) — ввод из другой вычислительной стадии;
- `Ingress` (тёмно-зелёный) — ввод через границу подсистемы.

Не у каждой стадии есть все шесть строк. У вычислительных стадий обязательны `Memory` и `CPU`. Из пары `Egress` / `Ingress` указывается одна соответствующая. Несколько входов `Input` перечисляются отдельно. Если выходов `Output` несколько, на основной строке стадии показывается один из них, остальные — у связанных «клонов», см. Множественные выходы.

Помимо числа, каждая метрика сопровождается цветной полосой. Для одного и того же типа метрики значения по стадиям **нормируются между собой**: у стадии с максимальным `SUM` полоса на всю ширину колонки `Statistics`, у остальных — пропорционально этому максимуму. Так проще сравнить, на какой стадии больше всего трафика, CPU, памяти и т.п.

Примечание

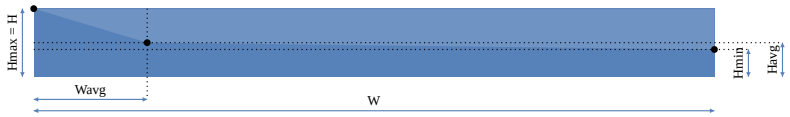
Масштабы **разных** типов метрик не сопоставимы: ширина полосы `Output` не означает то же самое, что ширина полосы `Input` — каждый тип масштабируется отдельно. Часто максимумы на выходе одной стадии и на входе другой близки, поэтому полосы выглядят согласованно. Однако это не всегда так.

Перекос данных

При параллельной обработке важно, чтобы задачи одной стадии стартовали и завершались почти синхронно. Длительность стадии задаётся **самой медленной** задачей: пока она не закончилась, стадия не считается завершённой.

Помимо `SUM` (она задаёт длину цветной полосы), по остальным агрегатам строится ломаная линия; участок полосы **выше** линии рисуется более светлым:

- `MAX` задаёт общий масштаб; ломаная начинается в левом верхнем углу полосы;
- `MIN` задаёт высоту ломаной на правом краю: отношение к `MAX` такое же, как `MIN/MAX` (например, при `MIN` вдвое меньше `MAX` конец ломаной — середина правой границы);
- `AVG` задаёт промежуточную точку между `MIN` и `MAX` по горизонтали и вертикали.




```
Hmax = H
Hmin = MIN * H / MAX
Havg = AVG * H / MAX
Wavg = (AVG – MIN) * W / (MAX – MIN)
```

Четыре типичных случая:

1. `MIN == AVG == MAX`: ломаная совпадает с верхним краем полосы, светлой зоны нет. Нагрузка по задачам равномерна (данные, время или память — в зависимости от метрики).



```
MAX = 100
AVG = 100
MIN = 100
```

2. `MIN < MAX`, но разброс небольшой: светлая область — узкий сегмент у правого верхнего угла. Роль `AVG` для визуальной оценки вторична.



```
MAX = 100
AVG = 90
MIN = 80
```

3. `MIN` существенно меньше `MAX`: важно положение `AVG`. Если оно близко к `MAX`, большинство задач загружены примерно одинаково, «лёгкие» задачи мало влияют на время стадии — перекос обычно некритичен.



```
MAX = 100
AVG = 90
MIN = 20
```

4. `MIN` сильно меньше `MAX`, а `AVG` близко к `MIN`: мало сильно перегруженных задач при большом числе простаивающих. Снижение перекоса (перераспределение работы) может заметно укоротить стадию.



```
MAX = 100
AVG = 30
MIN = 20
```

Практическое правило: **чем больше площадь светлой части полосы, тем сильнее неравномерность и тем внимательнее стоит смотреть на эту метрику.** Сильный перекос данных дополнительно помечается красным кругом с буквой `S` (data skew).

Примечание

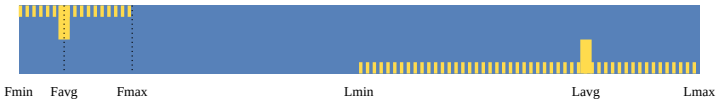
Число `SUM` рисуется поверх полосы и частично её перекрывает. Так сделано намеренно: либо полоса мала и стадия всё равно «тяжёлая», либо закрывается правый участок, менее важный для быстрой оценки; слева остаётся область с ломаной и светлым сегментом.

Перекос времени

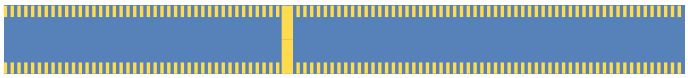
Неравномерность бывает и **по времени**: одинаковый объём данных и схожее CPU/память, но разная длительность задач (узел перегружен, вытеснение планировщиком и т.д.). Такой сценарий на метриках из предыдущего раздела не всегда виден.

Оценка по времени — по правой колонке с временной шкалой: когда стадия стартовала и завершилась, как шли задачи. Чистый перекос по времени без перекоса по данным редок; чаще проявляются оба эффекта, поэтому логично начинать с данных.

Временная шкала читается проще, чем геометрия перекоса по данным. Для каждого канала задачи отдают `FirstMessage` (`F`) и `LastMessage` (`L`) — моменты обработки первого и последнего сообщения. `Fmin` — начало активности всех задач стадии, `Lmax` — конец. Дополнительно рисуются две жёлтые «зебры»: от `Fmin` до `Fmax` вдоль верхнего края прямоугольника и от `Lmin` до `Lmax` вдоль нижнего. В точках `Favg` и `Lavg` — вертикальные штрихи до середины по высоте.



Особый случай: обе «зебры» на всю ширину, вертикальные штрихи совпадают (не обязательно ровно по центру). Это соответствует ситуации, когда у каждой задачи `FirstMessage == LastMessage` — одно сообщение на передачу всего объёма между задачами (мало данных на канал).

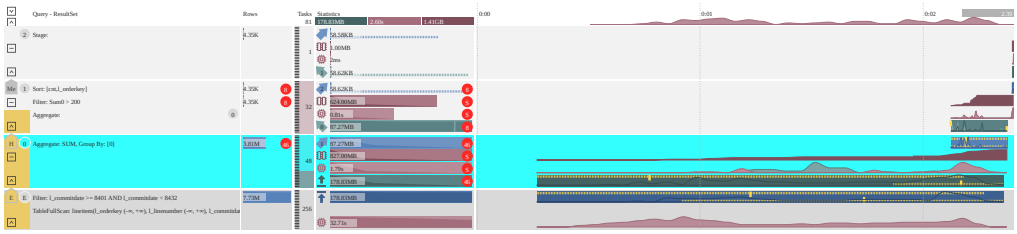


Совет

Так бывает, когда для каждой задачи в стадии `FirstMessage == LastMessage` : отправлено или принято ровно одно сообщение. Типично при небольшом объёме данных на канал.

Потребление CPU

Рассмотрим CPU на примере стадии `0` из раздела Структура фактического плана запроса. Здесь «потребление» — использование процессорного времени на полезную работу. Простой стадии не даёт вклада в CPU; при работе вклады задач суммируются обычным агрегированием.



«Загруженность» стадии и всей структуры выполнения запроса зависит от многих факторов, в том числе от других запросов на кластере. Для интерпретации CPU используется несколько представлений.

Помимо строк в `Statistics` , в правой колонке — график CPU по времени: интервалы с большей и меньшей утилизацией (песочно-красный) и ожидание данных (зелёный). При back pressure возможны и синие области; в этом разделе акцент на самом потреблении.

В примере суммарное процессорное время задач стадии `0` — 1,79 с, стадии `1` — 0,81 с, при более короткой календарной длительности. Чтобы сравнивать стадии, считается пропускная способность (throughput): число входных строк в секунду (сумма входов, если их несколько). От неё зависит насыщенность фона в `Tasks` . На диаграмме видно расхождение: у стадии `1` порядка 329 млн строк/с, у стадии `0` — около 90 млн (точные значения — во всплывающей подсказке по ячейке в `Tasks`).

Так сопоставляются разные величины — суммарный CPU и обработка строк, нормированная по длительности, — и становится проще искать узкое место в структуре выполнения.

Графики CPU по стадиям имеют **разный вертикальный масштаб**: шкала подбирается так, чтобы пики заполняли доступную высоту. У каждого типа метрики свой масштаб полос для сравнения стадий по одному показателю — иначе при разбросе на порядки менее нагруженные стадии были бы нечитаемы.

На рисунке выше видно, что временной график CPU стадии `0` в колонке `Timeline` занимает меньшую долю высоты, чем у стадии `1` , хотя суммарное время процессора у `0` больше: шкала подобрана отдельно для каждой стадии.

Потребление памяти

Память проще интерпретировать, чем CPU: ресурс менее «эластичен», и задачи чаще всего освобождают память к концу работы. Как и для CPU, временной график памяти по стадиям масштабируется независимо.

Как использовать итоги диагностики:

- Сравните полосы `Statistics` и `Timeline` — найдите стадии с наибольшим CPU, памятью или трафиком.
- Проверьте `Tasks` и перекос данных: неравномерная нагрузка удлиняет стадию.
- Сопоставьте метрики со стадиями и каналами связи в структуре плана — узкое место часто совпадает с тяжёлой стадией или каналом.
- Сравните оценки `EXPLAIN` с фактическими метриками `ANALYZE`.

Подсказки оптимизатора

Подсказки оптимизатора позволяют влиять на поведение стоимостного оптимизатора при планировании выполнения SQL-запросов. YDB поддерживает четыре типа подсказок для управления соединениями (joins) и статистикой.

Использование

Подсказки задаются через прагму `PRAGMA ydb.OptimizerHints` в начале SQL-запроса.
Если оптимизатор не смог применить хотя бы одну из указанных подсказок к запросу, пользователь будет оповещен через warning.

Синтаксис

Подсказки задаются в виде строки, содержащей массив выражений одного из четырех типов:

```
JoinType(TableList JoinType)
Rows(TableList Op Value)
Bytes(TableList Op Value)
JoinOrder(JoinTree)

где:
TableList - перечисление названий таблиц или элиасов из запроса
Op - операция:
  - `#` - задать абсолютное значение
  - `*` - умножить на значение
  - `/` - разделить на значение
  - `+` - прибавить значение
  - `-` - вычесть значение
  - `Number` - числовое значение
Value - числовое значение
JoinTree - представление бинарного дерева с помощью скобок, например: (R S) (T U)
```

Например, в следующем запросе используются три подсказки `Rows`, задающие кардинальность, а также подсказка полного порядка соединений `JoinOrder` и подсказка выбора алгоритма соединения `JoinType`:

```
PRAGMA ydb.OptimizerHints =
'
    Rows(R # 10e8)
    Rows(T # 1)
    Rows(R T # 1)
    JoinOrder( (R S) (T U) )
    JoinType(T U Broadcast)
';

SELECT * FROM
    R   INNER JOIN  S   on  R.id = S.id
        INNER JOIN  T   on  R.id = T.id
        INNER JOIN  U   on  T.id = U.id;
```

Требования к CBO (Cost Based Optimizer)

> **Примечание**
>
> Все подсказки (`Rows`, `Bytes`, `JoinOrder`) работают только с **включенным** стоимостным оптимизатором, кроме `JoinType` — его можно указывать и для выключенного CBO.

Типы подсказок

1. JoinType — алгоритм соединения

Позволяет принудительно задать алгоритм соединения для определенных таблиц.

В YDB сейчас поддерживаются три типа алгоритмов соединений:

- BroadcastJoin — это тип соединения, при котором один из наборов данных достаточно мал, чтобы его скопировать (разослать) на все необходимые узлы в кластере. Это позволяет каждому узлу выполнять объединение локально, не передавая данные по сети.

> **Примечание**
>
> Если порядок соединений не зафиксирован отдельной подсказкой, оптимизатор построит оба варианта планов: где пересылается левый и правый вход соединения. Если порядок соединений зафиксирован подсказкой, пересылаться будет правая сторона соединения.

- ShuffleJoin — это тип соединения, при котором данные разделяются (shuffle) по ключу соединения так, чтобы записи с одинаковым ключом обрабатывались на одном узле обработки. После такого перераспределения данных каждый узел выполняет локальное соединение таблиц. Результаты объединяются в один общий набор данных.
- LookupJoin — по каждой строке одного входа делается запрос в таблицу или индекс другого входа; в данный момент поддерживается только для строковых таблиц.

Синтаксис

```
JoinType(t1 t2 ... tn Broadcast | Shuffle | Lookup)
```

Параметры

- `t1 t2 ... tn` - таблицы, участвующие в соединении
- Алгоритм:
  - `Broadcast` - выбрать алгоритм BroadcastJoin
  - `Shuffle` - выбрать алгоритм ShuffleJoin
  - `Lookup` - выбрать алгоритм LookupJoin

Принцип работы

Если в плане запроса присутствует оператор соединения, который соединяет только те таблицы, что перечислены в списке, то оптимизатор выберет указанный алгоритм соединения, если он применим (например, нельзя применить алгоритм LookupJoin к колоночным таблицам). Если алгоритм не может быть применен, пользователь будет оповещен через warning.

Примеры

```
-- Использовать Broadcast для соединения таблиц nation, region
JoinType(nation region Broadcast)

-- Использовать ShuffleJoin для соединения, в поддереве которого будут только таблицы customers, orders, prod
ucts
JoinType(customers orders products Shuffle)

-- Использовать LookupJoin для соединения таблиц nation, region
JoinType(nation region Lookup)
```

Применим подсказки алгоритмов соединений к следующему запросу:

```
PRAGMA ydb.OptimizerHints =
'
    JoinType(R S Shuffle)
    JoinType(R S T Broadcast)
    JoinType(R S T U Shuffle)
    JoinType(R S T U V Broadcast)
';

SELECT * FROM
    R   INNER JOIN  S   on  R.id = S.id
        INNER JOIN  T   on  R.id = T.id
        INNER JOIN  U   on  T.id = U.id
        INNER JOIN  V   on  U.id = V.id;
```

Посмотреть план выполнения запроса можно с помощью команды CLI:

```
 ydb -p <profile_name> sql --explain -f query.sql
```

```
┌──────────────────────────────────────────────────────────────────────────────────────────────────┐
│ Operation                                                                                        │
├──────────────────────────────────────────────────────────────────────────────────────────────────┤
│ ┌> ResultSet                                                                                     │
│ └─┬> InnerJoin (MapJoin) (U.id = V.id)                                                           │
│   ├──┬> InnerJoin (Grace) (T.id = U.id)                                                          │
│   │  ├──┬> HashShuffle (KeyColumns: ["T.id"], HashFunc: "HashV2")                                │
│   │  │  └─┬> InnerJoin (MapJoin) (R.id = T.id)                                                   │
│   │  │    ├──┬> InnerJoin (Grace) (R.id = S.id)                                                 │
│   │  │    │  ├──┬> HashShuffle (KeyColumns: ["id"], HashFunc: "HashV2")                         │
│   │  │    │  │  └──> TableFullScan (Table: R, ReadColumns: ["id (-∞, +∞)","payload1","ts"])    │
│   │  │    │  └─┬> HashShuffle (KeyColumns: ["id"], HashFunc: "HashV2")                         │
│   │  │    │    └──> TableFullScan (Table: S, ReadColumns: ["id (-∞, +∞)","payload2"])          │
│   │  │    └──> TableFullScan (Table: T, ReadColumns: ["id (-∞, +∞)","payload3"])              │
│   │  └─┬> HashShuffle (KeyColumns: ["id"], HashFunc: "HashV2")                                  │
│   │    └──> TableFullScan (Table: U, ReadColumns: ["id (-∞, +∞)","payload4"])                  │
│   └──> TableFullScan (Table: V, ReadColumns: ["id (-∞, +∞)","payload5"])                       │
└──────────────────────────────────────────────────────────────────────────────────────────────────┘
```

Так как в процессе оптимизации запроса оптимизатор может изменить порядок соединений, подсказка должна отражать точный список таблиц, которые соединяются.
Например, в этом запросе предполагается, что порядок соединений будет: R с S, затем T и в итоге U. Указание другого алгоритма соединения может изменить порядок соединений в плане, и некоторые подсказки не будут применены. В таком случае можно добавить дополнительную подсказку порядка соединений.

2. Rows — подсказки по кардинальности

Позволяет изменить ожидаемое количество строк (оценку оптимизатора) для соединения или отдельных таблиц.

Принцип работы

Оптимизатор поменяет свою оценку количества строк для операции соединения, которая соединяет только те таблицы, которые перечислены в списке.

Синтаксис

```
Rows(t1 t2 ... tn (*|/|+|-|#) Number)
```

Параметры

- `t1 t2 ... tn` - таблицы
- Операция:
  - `*` - умножить на значение
  - `/` - разделить на значение
  - `+` - прибавить значение
  - `-` - вычесть значение
  - `#` - заменить значение
- `Number` - числовое значение

Примеры

```
-- Умножить ожидаемое количество строк на 2 для соединения, в поддереве которого есть только таблицы users or
ders yandex
Rows(users orders yandex * 2.0)

-- Заменить ожидаемое число строк таблицы products на 1.3e6
Rows(products # 1.3e6)

-- Уменьшить ожидаемое количество строк в 228 раз
Rows(filtered_table / 228)

-- Добавить 5000 строк к ожидаемому результату
Rows(table1 table2 + 5000)
```

Запустим запрос без подсказок кардинальности и затем посмотрим, как подсказки меняют план запроса.

```
SELECT * FROM
    R   INNER JOIN  S   on  R.id = S.id
        INNER JOIN  T   on  R.id = T.id;
```

Без подсказок оптимизатор строит следующий план:

```
┌─────────┬─────────┬─────────┬──────────────────────────────────────────────────────────────────────────────┐
| E-Cost | E-Rows | E-Size | Operation                                                                    |
├─────────┼─────────┼─────────┼──────────────────────────────────────────────────────────────────────────────┤
|        |        |        | ┌> ResultSet                                                                  |
| 114    | 10     | 300    | └─┬> InnerJoin (MapJoin) (S.id = R.id)                                          |
| 57     | 10     | 200    |   ├─┬> InnerJoin (MapJoin) (S.id = T.id)                                        |
| 0      | 10     | 100    |   | ├──> TableFullScan (Table: S, ReadColumns: ["id (-∞, +∞)","payload2"])    |
| 0      | 10     | 100    |   | └──> TableFullScan (Table: T, ReadColumns: ["id (-∞, +∞)","payload3"])    |
| 0      | 10     | 100    |   └──> TableFullScan (Table: R, ReadColumns: ["id (-∞, +∞)","payload1","ts"]) |
└─────────┴─────────┴─────────┴──────────────────────────────────────────────────────────────────────────────┘
```

Если применить следующие подсказки:

```
PRAGMA ydb.OptimizerHints =
'
    Rows(R # 10e8)
    Rows(T # 1)
    Rows(S # 10e8)
    Rows(R T # 1)
    Rows(R S # 10e8)
';
SELECT * FROM
    R   INNER JOIN  S   on  R.id = S.id
        INNER JOIN  T   on  R.id = T.id;
```

То получится следующий план:

```
┌─────────────┬─────────┬─────────┬──────────────────────────────────────────────────────────────────────────────
──┐
| E-Cost     | E-Rows | E-Size | Operation
|
├─────────────┼─────────┼─────────┼──────────────────────────────────────────────────────────────────────────────
──┤
|            |        |        | ┌> ResultSet
|
| 3.000e+09 | 1      | 100    | └─┬> InnerJoin (MapJoin) (S.id = R.id)
|
| 0          | 1e+09  | 100    |   ├──> TableFullScan (Table: S, ReadColumns: ["id (-∞, +∞)","payload2"])
|
| 1.500e+09 | 1      | 100    |   └─┬> InnerJoin (MapJoin) (R.id = T.id)
|
| 0          | 1e+09  | 100    |     ├──> TableFullScan (Table: R, ReadColumns: ["id (-∞, +∞)","payload1","t
s"]) |
| 0          | 10     | 100    |     └──> TableFullScan (Table: T, ReadColumns: ["id (-∞, +∞)","payload3"])
|
└─────────────┴─────────┴─────────┴──────────────────────────────────────────────────────────────────────────────
──┘
```

Также вернутся оповещения:

```
Warning: Unapplied hint: Rows(R S # 10e8)
```

Здесь видно, что после применения подсказок кардинальности базовых таблиц поменялся порядок соединений, и одна из подсказок не смогла примениться, так как такого соединения в плане нет.

3. Bytes — подсказки по размеру данных

Позволяет изменить ожидаемый размер данных в байтах для соединения или отдельных таблиц.

Синтаксис

```
Bytes(t1 t2 ... tn (*|/|+|-|#) Number)
```

Параметры аналогичны Rows, но применяются к размеру данных в байтах

Примеры

```
-- Умножить ожидаемый размер данных на 1.5
Bytes(large_table * 1.5)

-- Заменить размер данных для соединения на 1GB
Bytes(table1 table2 # 1073741824)

-- Уменьшить ожидаемый размер в 2 раза
Bytes(compressed_table / 2)

-- Добавить 100MB к ожидаемому размеру
Bytes(temp_table + 104857600)
```

4. JoinOrder — порядок соединений

Позволяет зафиксировать определенное поддерево соединений в общем дереве соединений.

Синтаксис

```
JoinOrder((t1 t2) (t3 (t4 ...)))
```

Параметры

- Вложенная структура скобок определяет порядок соединений
- `(t1 t2)` означает, что t1 и t2 должны быть соединены первыми
- Можно задавать произвольную глубину вложенности

Принцип работы

Оптимизатор будет рассматривать только те планы, в которых присутствует заданный частичный или полный порядок соединений.

Примеры

```
-- Принудительно соединить сначала users с orders, затем с products
JoinOrder((users orders) products)

-- Более сложный порядок соединений
JoinOrder(((customers orders) products) shipping)

-- Группировка соединений
JoinOrder((table1 table2) (table3 table4))

-- Многоуровневая структура
JoinOrder((users (orders products)) (addresses phones))
```

Применим подсказку порядка соединений к следующему запросу:

```
SELECT * FROM
    R   INNER JOIN  S   on  R.id = S.id
        INNER JOIN  T   on  R.id = T.id;
```

План запроса без подсказок выглядит так:

```
┌─────────┬─────────┬─────────┬──────────────────────────────────────────────────────────────────────────────────────┐
│ E-Cost  │ E-Rows  │ E-Size  │ Operation                                                                            │
├─────────┼─────────┼─────────┼──────────────────────────────────────────────────────────────────────────────────────┤
│         │         │         │ ┌> ResultSet                                                                         │
│ 114     │ 10      │ 300     │ └─┬> InnerJoin (MapJoin) (S.id = R.id)                                               │
│ 57      │ 10      │ 200     │   ├─┬> InnerJoin (MapJoin) (S.id = T.id)                                             │
│ 0       │ 10      │ 100     │   │ ├──> TableFullScan (Table: S, ReadColumns: ["id (-∞, +∞)","payload2"])         │
│ 0       │ 10      │ 100     │   │ └──> TableFullScan (Table: T, ReadColumns: ["id (-∞, +∞)","payload3"])         │
│ 0       │ 10      │ 100     │   └──> TableFullScan (Table: R, ReadColumns: ["id (-∞, +∞)","payload1","ts"])      │
└─────────┴─────────┴─────────┴──────────────────────────────────────────────────────────────────────────────────────┘
```

Применив следующую подсказку порядка соединений:

```
PRAGMA ydb.OptimizerHints =
'
    JoinOrder(T (R S))
';
SELECT * FROM
    R   INNER JOIN  S   on  R.id = S.id
        INNER JOIN  T   on  R.id = T.id;
```

Получим такой план:

```
┌─────────┬────────┬────────┬──────────────────────────────────────────────────────────────────────────────────────────────────────────────┐
┐
| E-Cost | E-Rows | E-Size | Operation
|
├─────────┼────────┼────────┼──────────────────────────────────────────────────────────────────────────────────────────────────────────────┤
|         |        |        | ┌> ResultSet
|
| 114     | 10     | 300    | └─┬> InnerJoin (MapJoin) (T.id = R.id)
|
| 0       | 10     | 100    |   ├──> TableFullScan (Table: T, ReadColumns: ["id (-∞, +∞)","payload3"])
|
| 57      | 10     | 200    |   └─┬> InnerJoin (MapJoin) (R.id = S.id)
|
| 0       | 10     | 100    |     ├──> TableFullScan (Table: R, ReadColumns: ["id (-∞, +∞)","payload1","ts"])
|
| 0       | 10     | 100    |     └──> TableFullScan (Table: S, ReadColumns: ["id (-∞, +∞)","payload2"])
|
└─────────┴────────┴────────┴──────────────────────────────────────────────────────────────────────────────────────────────────────────────┘
```

Здесь видно, что изменился порядок соединений на тот, который указан в подсказке.

Комбинирование подсказок

Можно использовать несколько типов подсказок одновременно в рамках одной прагмы:

```
PRAGMA ydb.OptimizerHints =
'
    JoinType(users orders Broadcast)
    Rows(users orders * 0.5)
    JoinOrder((users orders) products)
    Bytes(products # 1073741824)
';
```

Параметризованные запросы и повторная компиляция

Компиляция запроса требует времени и ресурсов, поэтому в YDB предусмотрен кэш компиляции. Результат компиляции сохраняется в кэше на узле кластера и переиспользуется только при **точном совпадении текста** запроса. Поэтому, если в коде приложения значения попадают в YQL через конкатенацию или форматирование строк, при каждом новом наборе данных сервер получает **другой текст** и компилирует его заново — хотя структура SQL может оставаться той же.

Последствия:

- увеличивается задержка на этапе компиляции;
- растёт нагрузка на CPU узлов;
- при большом числе уникальных текстов ухудшается переиспользование ограниченного кэша на узле.

Избежать лишней компиляции помогает **параметризованный запрос**: текст YQL фиксирован, а входные значения передаются отдельно через именованные параметры (например, `$userId`). Ниже оба подхода сравниваются на одном примере.

Подстановка значений в текст запроса

Приложение подставляет значения прямо в текст запроса:

```
SELECT id, name FROM users WHERE id = 123 AND status = "active";
SELECT id, name FROM users WHERE id = 456 AND status = "inactive";
```

Запросы отличаются только значениями, но тексты разные — для сервера это два разных запроса, и каждый компилируется отдельно.

Передача значений параметрами

Приложение передаёт значения отдельно от текста, текст запроса не меняется:

```
DECLARE $userId AS Uint64;
DECLARE $status AS Utf8;

SELECT id, name FROM users WHERE id = $userId AND status = $status;
```

Сервер обрабатывает вызовы такого запроса по-разному:

1. **Первый вызов** — сервер компилирует запрос и сохраняет результат в кэше на узле.
2. **Следующие вызовы** — если текст уже есть в кэше, используется готовый результат: меняются только значения `$userId` и `$status`, повторная компиляция не требуется.

Параметрами передаются только значения: имя таблицы или порядок сортировки задать параметром нельзя, и при их изменении текст запроса становится другим.

Содержимое кэша компиляции смотрите в разделе Кэш компиляции запросов, а время компиляции при выполнении запросов — в топах запросов (поле `CompileDuration`).

Размер и настройки кэша, флаг `KeepInCache`, синтаксис `DECLARE` и передача параметров из кода описаны в разделе Параметризованные запросы справки YDB SDK.

Смотрите также

- План выполнения запроса
- Кэш компиляции запросов
- Параметризованные запросы (справка SDK)
- Выполнение параметризованных запросов (справка CLI)
- Примеры приложений
- Обзор оптимизации запросов

Загрузка данных в YDB

В данном разделе предоставлены рекомендации по эффективной загрузке данных в YDB.

Существуют антипаттерны и неоптимальные настройки загрузки данных, при использовании которых не может гарантироваться приемлемая производительность загрузки данных в БД.
Для увеличения скорости загрузки данных нужно учесть следующие рекомендации:

- Следует шардировать таблицу при её создании, чтобы сразу после начала загрузки данных эффективно использовать пропускную способность системы.

  По умолчанию созданная таблица состоит из одного шарда. YDB поддерживает автоматическое шардирование таблицы по объему данных, т.е. при достижении определенного размера шард таблицы разделяется на два шарда.
  Приемлемый размер для деления шарда таблицы — 2 ГБ. С увеличением количества шардов пропускная способность загрузки данных увеличивается, но некоторое время с начала пропускная способность будет низкой.
  Поэтому при первичной заливке большого объема данных рекомендуется создавать таблицу, сразу состоящую из желаемого количества шардов. Количество шардов можно подсчитать исходя из 1 ГБ данных на шард в результирующем наборе.
- Вставка нескольких строк в каждой транзакции для уменьшения доли накладных расходов самих транзакций.

  Каждая транзакция в YDB имеет некоторые накладные расходы. Для уменьшения суммарных накладных расходов следует делать транзакции, вставляющие много строк. К хорошим показателям производительности приводит завершение транзакции при достижении объема в 1 МБ данных или 100000 строк.
  При загрузке данных избегайте транзакций, состоящих из вставки одной строки.
- В рамках каждой транзакции вставляйте строки из отсортированного по первичному ключу набора, чтобы минимизировать количество шардов, которые затрагивает транзакция.

  В YDB транзакции затрагивающие более, чем один шард, имеют большие накладные расходы по сравнению с транзакциями, затрагивающими ровно один шард. Кроме того, эти накладные расходы увеличиваются с ростом количества шардов таблицы, которые участвуют в транзакции.
  Рекомендуется подбирать вставляемые в конкретной транзакции строки таким образом, чтобы они были расположены в малом количестве шардов, в идеале — в одном.
- Если требуется залить данные в несколько таблиц, то в рамках одного запроса рекомендуется заливать данные в одну таблицу.
- Если требуется залить данные в таблицу с синхронным вторичным индексом, то рекомендуется сначала залить данные в таблицу и после завершения заливки данных построить вторичный индекс.
- Следует избегать записи последовательно в порядке возрастания или убывания первичного ключа.

  Запись в таблицу данных с монотонно возрастающим ключом приведет к тому, что все новые данные будут записываться в конец таблицы, поскольку все таблицы в YDB отсортированы по возрастанию первичного ключа. Так как YDB разделяет таблицы на шарды по диапазонам ключей, вставки будут обрабатываться одним конкретным сервером, отвечающим за "последний" шард. Сосредоточение нагрузки на одном сервере приведет к медленной загрузке данных и не эффективному использованию распределенной системы.
- В некоторых сценариях требуется записать в таблицу первоначальные данные (зачастую большого объема) прежде чем включать OLTP-нагрузку. В таком случае не требуется транзакционность на уровне отдельных запросов и можно воспользоваться вызовом `BulkUpsert` в CLI, SDK и API. За счёт отказа от транзакционности такой подход имеет значительно меньшие накладные расходы по сравнению с YQL запросами. В случае успешного ответа на запрос, метод `BulkUpsert` гарантирует фиксацию всех данных, записанных в рамках данного запроса.

> **Важно**
>
> Метод `BulkUpsert` не поддерживается для таблиц с синхронными вторичными индексами.

Исходя из данных рекомендаций можно предложить следующий алгоритм действий для эффективной загрузки данных в YDB:

1. Создайте таблицу, состоящую из желаемого количества шардов, исходя из 1 ГБ данных на шард.
2. Отсортируйте исходный набор данных по предполагаемому первичному ключу.
3. Разделите полученный набор данных на части по количеству шардов таблицы. Каждая часть будет состоять из набора последовательно расположенных строк.
4. Полученные части загружайте в шарды таблицы параллельно.
5. Делайте `COMMIT` после каждых 100000 строк или 1 МБ данных.

Постраничный вывод

В разделе приведены рекомендации по организации постраничного вывода данных.

Для организации постраничного вывода рекомендуется последовательно выбирать данные, отсортированные по первичному ключу, ограничивая количество строк ключевым словом LIMIT.

Примечание

`$lastCity, $lastNumber` - значения первичного ключа, полученные в результате предыдущего запроса.

Запрос с примером рекомендованного способа организации постраничного вывода:

```
--  Table `schools`:
-- ┌──────────┬──────────┬─────┐
-- | Name     | Type     | Key |
-- ├──────────┼──────────┼─────┤
-- | city     | Utf8?    | K0  |
-- | number   | Uint32?  | K1  |
-- | address  | Utf8?    |     |
-- └──────────┴──────────┴─────┘

DECLARE $limit AS Uint64;
DECLARE $lastCity AS Utf8;
DECLARE $lastNumber AS Uint32;

SELECT * FROM schools
WHERE (city, number) > ($lastCity, $lastNumber)
ORDER BY city, number
LIMIT $limit;
```

В примере запроса, приведенном выше, в операторе `WHERE` применено сравнение кортежей для отбора очередного множества строк. Сравнение кортежей выполняется поэлементно слева направо, поэтому порядок указания полей в кортеже должен совпадать с порядком указания полей в первичном ключе, чтобы избежать полного сканирования таблицы при выполнении запроса.

Значение NULL в ключевой колонке

В YDB все колонки, включая ключевые, могут иметь значение NULL. Несмотря на это использование NULL в качестве значений в ключевых колонках крайне не рекомендуется, так как по SQL стандарту NULL нельзя сравнивать. Как следствие, лаконичные SQL конструкции с простыми операторами сравнения будут работать некорректно. Вместо них придется использовать громоздкие конструкции с IS NULL/IS NOT NULL выражениями.

Примеры реализации постраничного вывода

- C++
- Java
- Python
- Go

Использование таймаутов

В разделе приведено описание доступных таймаутов и представлены примеры использования на различных языках программирования.

Предпосылки к использованию таймаутов

Механизм таймаутов в YDB призван решить следующие проблемы:

- Не дать запросу выполнятся так долго, что результат запроса становится не интересен для дальнейшего использования.
- Обнаружить проблемы сетевой связности.

Оба этих случая важны для обеспечения отказоустойчивости системы в целом. Рассмотрим таймауты подробнее.

Таймаут на операцию

Значение `operation_timeout` определяет время, в течение которого результат запроса интересен пользователю. Если за данное время операция не выполнилась, сервер возвращает ошибку с кодом `Timeout` и попытается прекратить выполнение запроса, однако отмена запроса не гарантируется. Таким образом, запрос, на который пользователю была возвращена ошибка `Timeout`, может быть как успешно выполнен на сервере, так и отменен.

Таймаут отмены операции

Значение `cancel_after` определяет время, через которое сервер начнет отмену запроса, если отменить запрос возможно. В случае успешной отмены запроса сервер вернет ошибку с кодом `Cancelled`.

Транспортный таймаут

На каждый запрос клиент должен выставить транспортный таймаут. Данное значение позволяет определить количество времени, которое клиент готов ждать ответа от сервера. Если за данное время сервер не ответил, то клиенту будет возвращена транспортная ошибка с кодом `DeadlineExceeded`. Важно выставить такое значение клиентского таймаута чтоб при нормальной работе приложения и сети транспортные таймауты не срабатывали.

> **Совет**
>
> Лучше выставлять транспортный таймаут с запасом относительно времени (измеренного во время нагрузочного тестирования или тестирования на отказ) ответа на запрос.
>
> Например, можно использовать удвоенное значение 99-го перцентиля времени ответа. Иными словами, таймаут должен равняться
>
> Время ответа для конкретного запроса нужно измерять в коде клиента (не стоит использовать метрики времени ответа, которые предоставляет сервер, или данные из статистики запроса).

Применение таймаутов

TableService

Всегда рекомендуется устанавливать и таймаут на операцию, и транспортный таймаут. Значение транспортного таймаута следует делать на 50-100 миллисекунд больше, чем значение таймаута на операцию, чтобы оставался некоторый запас времени, за который клиент сможет получить серверную ошибку с кодом `Timeout`.

Пример использования таймаутов:

Python

```python
import ydb

def execute_in_tx(session, query):
  settings = ydb.BaseRequestSettings()
  settings = settings.with_timeout(0.5)  # transport timeout
  settings = settings.with_operation_timeout(0.4)  # operation timeout
  settings = settings.with_cancel_after(0.4)  # cancel after timeout
  session.transaction().execute(
      query,
      commit_tx=True,
      settings=settings,
  )
```

C++

```cpp
#include <ydb/public/sdk/cpp/client/ydb.h>
#include <ydb/public/sdk/cpp/client/ydb_table.h>
#include <ydb/public/sdk/cpp/client/ydb_value.h>

using namespace NYdb;
using namespace NYdb::NTable;

TAsyncStatus ExecuteInTx(TSession& session, TString query, TParams params) {
  return session.ExecuteDataQuery(
      query
      , TTxControl::BeginTx(TTxSettings::SerializableRW()).CommitTx()
      , TExecDataQuerySettings()
      .OperationTimeout(TDuration::MilliSeconds(300))  // operation timeout
      .ClientTimeout(TDuration::MilliSeconds(400))   // transport timeout
      .CancelAfter(TDuration::MilliSeconds(300)));  // cancel after timeout
}
```

Go

```
import (
  "context"
  "time"

  ydb "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/table"
)

func executeInTx(ctx context.Context, s table.Session, query string) {
ctx, cancel := context.WithTimeout(ctx, time.Millisecond*500) // transport timeout
defer cancel()
ctx = ydb.WithOperationTimeout(ctx, time.Millisecond*400)     // operation timeout
ctx = ydb.WithOperationCancelAfter(ctx, time.Millisecond*400) // cancel after timeout
tx := table.TxControl(table.BeginTx(table.WithSerializableReadWrite()), table.CommitTx())
_, res, err := s.Execute(ctx, tx, query, table.NewQueryParameters())
}
```

QueryService

QueryService использует исключительно клиентские таймауты (дедлайны). Значение по умолчанию не задано, поэтому время выполнения запроса не ограничено.

Рекомендуется всегда явно устанавливать таймаут, чтобы не ждать ответов на запросы, которые уже не нужны. Это также позволяет избежать бесконечного ожидания ответа от сервера в условиях нестабильной сети. Для своевременного обнаружения разрывов соединения рекомендуется дополнительно настраивать параметры gRPC KeepAlive.

Python

```
import ydb

settings = (
    ydb.BaseRequestSettings()
    .with_timeout(5)
)

for result_set in session.execute(query, settings=settings):
    ...

# Or via pool with retry
result_sets = pool.execute_with_retries(query, settings=settings)
```

C++

```
#include <ydb-cpp-sdk/client/query/client.h>

auto settings = NYdb::NQuery::TExecuteQuerySettings()
    .Deadline(TDeadline::AfterDuration(std::chrono::seconds(5)));

auto result = session.ExecuteQuery(
    query,
    NYdb::NQuery::TTxControl::NoTx(),
    {},
    settings
).GetValueSync();
```

Go

```
import (
    "context"
    "time"
    "github.com/ydb-platform/ydb-go-sdk/v3"
)

ctx, cancel := context.WithTimeout(context.Background(), 5*time.Second)
defer cancel()

err := db.Query().Do(ctx, func(ctx context.Context, s query.Session) error {
    res, err := s.Query(ctx, `SELECT 1`)
    return err
}, query.WithIdempotent())
```

Java

```
import tech.ydb.query.settings.ExecuteQuerySettings;
import java.time.Duration;

ExecuteQuerySettings settings = ExecuteQuerySettings.newBuilder()
    .withRequestTimeout(Duration.ofSeconds(5))
    .build();

session.createQuery(query, TxMode.NONE, Params.empty(), settings)
    .execute(part -> {
        ResultSetReader rs = part.getResultSetReader();
        // ...
```

```
    }).join();
```

Системные представления базы данных

Для получения служебной информации о состоянии базы данных можно обращаться к системным представлениям (system views). Они доступны из корня дерева базы данных и используют системный префикс пути `.sys`.

> **Примечание**
>
> Частое обращение к системным представлениям приводит к дополнительной нагрузке на базу данных, особенно в случае большого размера базы. Превышение частоты в 1 запрос в секунду не рекомендуется.

Партиции

Следующее системное представление хранит детализированную информацию об партициях таблиц базы данных:

- `partition_stats` — содержит информацию о моментальных метриках и кумулятивных счетчиках операций. К первым относятся, например, данные о нагрузке на CPU или количестве выполняемых транзакций. Ко вторым — общее количество прочитанных строк.

Предназначено, например, для выявления неравномерно нагруженных партиций или отображения размера данных в них.

Моментальные метрики (`NodeId`, `AccessTime`, `CPUCores` и т.д.) содержат мгновенные значения.
Кумулятивные (не моментальные) метрики (`RowReads`, `RowUpdates`, `LocksAcquired` и т.д.) хранят накопленные значения с момента последнего старта таблетки (`StartTime`), обслуживающей партицию.

Структура представления:

Колонка	Описание	Тип данных	Моментальная/ кумулятивная
`OwnerId`	Идентификатор SchemeShard таблицы. Ключ: `0`.	`Uint64`	Моментальная
`PathId`	Идентификатор пути в SchemeShard. Ключ: `1`.	`Uint64`	Моментальная
`PartIdx`	Порядковый номер партиции. Ключ: `2`.	`Uint64`	Моментальная
`FollowerId`	Идентификатор реплики таблетки партиции. Значение 0 означает лидера. Ключ: `3`.	`Uint32`	Моментальная
`DataSize`	Приблизительный размер данных партиции в байтах.	`Uint64`	Моментальная
`RowCount`	Приблизительное количество строк.	`Uint64`	Моментальная
`IndexSize`	Размер индекса партиции в байтах.	`Uint64`	Моментальная
`CPUCores`	Моментальное значение нагрузки на партицию (доля времени ядра процессора, затраченного актором партиции).	`Double`	Моментальная
`TabletId`	Идентификатор таблетки партиции.	`Uint64`	Моментальная
`Path`	Полный путь к таблице.	`Utf8`	Моментальная
`NodeId`	Идентификатор ноды, на которой в данный момент обслуживается партиция.	`Uint32`	Моментальная
`StartTime`	Последний момент запуска таблетки партиции.	`Timestamp`	Моментальная
`AccessTime`	Последний момент чтения из партиции.	`Timestamp`	Моментальная
`UpdateTime`	Последний момент записи в партицию.	`Timestamp`	Моментальная
`RowReads`	Количество чтений по ключу.	`Uint64`	Кумулятивная
`RowUpdates`	Количество записанных строк .	`Uint64`	Кумулятивная
`RowDeletes`	Количество удалённых строк.	`Uint64`	Кумулятивная
`RangeReads`	Количество чтений по диапазону ключей.	`Uint64`	Кумулятивная
`RangeReadRows`	Количество строк, прочитанных в диапазонах.	`Uint64`	Кумулятивная
`InFlightTxCount`	Количество исполняющихся транзакций.	`Uint64`	Моментальная
`ImmediateTxCompleted`	Количество завершившихся одношардовых транзакций.	`Uint32`	Кумулятивная
`CoordinatedTxCompleted`	Количество завершившихся распределенных транзакций.	`Uint64`	Кумулятивная
`TxRejectedByOverload`	Количество транзакций, отменённых по причине высокой нагрузки.	`Uint64`	Кумулятивная
`TxRejectedByOutOfStorage`	Количество транзакций, отменённых из-за нехватки места в хранилище.	`Uint64`	Кумулятивная
`TxCompleteLag`	Задержка выполнения транзакций (насколько транзакции отстают от запланированного времени).	`Interval`	Моментальная

`LastTtlRunTime`	Последний момент запуска очистки партиции по TTL	`Timestamp`	Моментальная
`LastTtlRowsProcessed`	Количество проверенных строк партиции при последней очистке по TTL	`Uint64`	Моментальная
`LastTtlRowsErased`	Количество удалённых строк партиции при последней очистке по TTL	`Uint64`	Моментальная
`LocksAcquired`	Количество установленных блокировок.	`Uint64`	Кумулятивная
`LocksWholeShard`	Количество установленных блокировок "весь шард".	`Uint64`	Кумулятивная
`LocksBroken`	Количество сломанных блокировок.	`Uint64`	Кумулятивная

Примеры запросов

Топ-5 самых загруженных партиций среди всех таблиц базы данных:

```
SELECT
    Path,
    PartIdx,
    CPUCores
FROM `.sys/partition_stats`
ORDER BY CPUCores DESC
LIMIT 5
```

Список таблиц базы с размерами и нагрузкой в моменте:

```
SELECT
    Path,
    COUNT(*) as Partitions,
    SUM(RowCount) as Rows,
    SUM(DataSize) as Size,
    SUM(CPUCores) as CPU
FROM `.sys/partition_stats`
GROUP BY Path
```

Список таблиц базы с наибольшим числом сломанных блокировок:

```
SELECT
    Path,
    COUNT(*) as Partitions,
    SUM(LocksBroken) as TotalLocksBroken
FROM `.sys/partition_stats`
GROUP BY Path
ORDER BY TotalLocksBroken DESC
```

Топы запросов

Следующие системные представления хранят данные для анализа пользовательских запросов.

Наибольшее полное время выполнения запроса:

- `top_queries_by_duration_one_minute` — данные разбиты на минутные интервалы, содержит данные за последние 6 часов;
- `top_queries_by_duration_one_hour` — данные разбиты на часовые интервалы, содержит данные за последние 2 недели.

Наибольшее количество прочитанных из таблицы байт:

- `top_queries_by_read_bytes_one_minute` — данные разбиты на минутные интервалы, содержит данные за последние 6 часов;
- `top_queries_by_read_bytes_one_hour` — данные разбиты на часовые интервалы, содержит данные за последние 2 недели.

Наибольшее затраченное процессорное время:

- `top_queries_by_cpu_time_one_minute` — данные разбиты на минутные интервалы, содержит данные за последние 6 часов;
- `top_queries_by_cpu_time_one_hour` — данные разбиты на часовые интервалы, содержит данные за последние 2 недели.

Запросы с одним и тем же текстом объединяются, в выдачу попадает запрос с максимальным значением соответствующей метрики.
Каждый временной интервал (минута или час) содержит ТОП-5 запросов, выполненных в этот временной интервал.

Поля, предоставляющие информацию о затраченном процессорном времени (... `CPUTime`), выражены в микросекундах.

Текст запроса ограничен 10 килобайтами.

Все представления имеют одинаковую структуру:

Колонка	Описание
`IntervalEnd`	Момент окончания минутного или часового интервала, за который собрана статистика. Тип: `Timestamp`. Ключ: `0`.

`Rank`	Ранг запроса в топе. Тип: `Uint32`. Ключ: `1`.
`QueryText`	Текст запроса. Тип: `Utf8`.
`Duration`	Полное время исполнения запроса. Тип: `Interval`.
`EndTime`	Момент окончания исполнения запроса. Тип: `Timestamp`.
`Type`	Тип запроса ("data", "scan", "script"). Тип: `String`.
`ReadRows`	Количество прочитанных строк. Тип: `Uint64`.
`ReadBytes`	Количество прочитанных байт. Тип: `Uint64`.
`UpdateRows`	Количество записанных строк. Тип: `Uint64`.
`UpdateBytes`	Количество записанных байт. Тип: `Uint64`.
`DeleteRows`	Количество удалённых строк. Тип: `Uint64`.
`DeleteBytes`	Количество удалённых байт. Тип: `Uint64`.
`Partitions`	Количество партиций таблиц, участвовавших в исполнении запроса. Тип: `Uint64`.
`UserSID`	Security ID пользователя. Тип: `String`.
`ParametersSize`	Размер параметров запроса в байтах. Тип: `Uint64`.
`CompileDuration`	Длительность компиляции запроса. Тип: `Interval`.
`FromQueryCache`	Использовался ли кэш подготовленных запросов. Тип: `Bool`.
`CPUTime`	Общее процессорное время, использованное для исполнения запроса (микросекунды). Тип: `Uint64`.
`ShardCount`	Количество шардов, участвующих в исполнении запроса. Тип: `Uint64`.
`SumShardCPUTime`	Общее процессорное время, затраченное в шардах. Тип: `Uint64`.
`MinShardCPUTime`	Минимальное процессорное время, затраченное в шардах. Тип: `Uint64`.
`MaxShardCPUTime`	Максимальное процессорное время, затраченное в шардах. Тип: `Uint64`.
`ComputeNodesCount`	Количество вычислительных нод, задействованных в исполнении запроса. Тип: `Uint64`.
`SumComputeCPUTime`	Общее процессорное время, затраченное в вычислительных нодах. Тип: `Uint64`.
`MinComputeCPUTime`	Минимальное процессорное время, затраченное в вычислительных нодах. Тип: `Uint64`.
`MaxComputeCPUTime`	Максимальное процессорное время, затраченное в вычислительных нодах. Тип: `Uint64`.
`CompileCPUTime`	Процессорное время, затраченное на компиляцию запроса. Тип: `Uint64`.
`ProcessCPUTime`	Процессорное время, затраченное на общую обработку запроса. Тип: `Uint64`.

Примеры запросов

Топ запросов по времени выполнения. Запрос выполняется к представлению `.sys/top_queries_by_duration_one_minute`:

```
PRAGMA AnsiInForEmptyOrNullableItemsCollections;
$last = (
    SELECT
        MAX(IntervalEnd)
    FROM `.sys/top_queries_by_duration_one_minute`
```

```
);
SELECT
    IntervalEnd,
    Rank,
    QueryText,
    Duration
FROM `.sys/top_queries_by_duration_one_minute`
WHERE IntervalEnd IN $last
```

Запросы, прочитавшие больше всего байт. Запрос выполняется к представлению `.sys/top_queries_by_read_bytes_one_minute`:

```
SELECT
    IntervalEnd,
    QueryText,
    ReadBytes,
    ReadRows,
    Partitions
FROM `.sys/top_queries_by_read_bytes_one_minute`
WHERE Rank = 1
```

Подробная информация о запросах

Следующее системное представление содержит подробную информацию о запросах:

- `query_metrics_one_minute` — данные разбиты по минутным интервалам, содержит до 256 запросов за последние 6 часов.

Каждая строка представления содержит информацию о множестве случившихся за интервал запросов с одинаковым текстом. Поля представления содержат минимальное, максимальное и суммарное значение по каждой отслеживаемой характеристике запроса. В пределах интервала запросы отсортированы по убыванию суммарного потраченного процессорного времени.

Ограничения:

- текст запроса ограничен 10 килобайтами;
- статистика может быть неполной, если база испытывает сильную нагрузку.

Структура представления:

Колонка	Описание
`IntervalEnd`	Момент окончания минутного интервала, за который собрана статистика Тип: `Timestamp`. Ключ: `0`.
`Rank`	Ранг запроса в пределах интервала (по полю SumCPUTime). Тип: `Uint32`. Ключ: `1`.
`QueryText`	Текст запроса. Тип: `Utf8`.
`Count`	Количество запусков запроса. Тип: `Uint64`.
`SumDuration`	Общая длительность запросов. Тип: `Interval`.
`MinDuration`	Минимальная длительность запроса. Тип: `Interval`.
`MaxDuration`	Максимальная длительность запроса. Тип: `Interval`.
`SumCPUTime`	Общее затраченное процессорное время. Тип: `Uint64`.
`MinCPUTime`	Минимальное затраченное процессорное время. Тип: `Uint64`.
`MaxCPUTime`	Максимальное затраченное процессорное время. Тип: `Uint64`.
`SumReadRows`	Общее количество прочитанных строк. Тип: `Uint64`.
`MinReadRows`	Минимальное количество прочитанных строк. Тип: `Uint64`.
`MaxReadRows`	Максимальное количество прочитанных строк. Тип: `Uint64`.
`SumReadBytes`	Общее количество прочитанных байт. Тип: `Uint64`.
`MinReadBytes`	Минимальное количество прочитанных байт. Тип: `Uint64`.
`MaxReadBytes`	Максимальное количество прочитанных байт. Тип: `Uint64`.

`SumUpdateRows`	Общее количество записанных строк. Тип: `Uint64`.
`MinUpdateRows`	Минимальное количество записанных строк. Тип: `Uint64`.
`MaxUpdateRows`	Максимальное количество записанных строк. Тип: `Uint64`.
`SumUpdateBytes`	Общее количество записанных байт. Тип: `Uint64`.
`MinUpdateBytes`	Минимальное количество записанных байт. Тип: `Uint64`.
`MaxUpdateBytes`	Максимальное количество записанных байт. Тип: `Uint64`.
`SumDeleteRows`	Общее количество удалённых строк. Тип: `Uint64`.
`MinDeleteRows`	Минимальное количество удалённых строк. Тип: `Uint64`.
`MaxDeleteRows`	Максимальное количество удалённых строк. Тип: `Uint64`.
`LocksBrokenAsBreaker`	Количество блокировок, которые сломал данный запрос. Тип: `Uint64`.
`LocksBrokenAsVictim`	Количество блокировок данного запроса, которые были сломаны. Тип: `Uint64`.

Примечание

В данную статистику не входят:

- Блокировки, сломанные из-за изменений схемы, TTL, асинхронной репликации
- Блокировки, сломанные из-за разделения или слияния партиций
- Блокировки, сломанные при перезапуске таблеток
- Блокировки, сломанные при коммите интерактивных транзакций (когда COMMIT выполняется отдельным запросом)

Примеры запросов

Топ-10 запросов за последние 6 часов по общему количеству записанных строк в минутном интервале:

```
SELECT
    SumUpdateRows,
    Count,
    QueryText,
    IntervalEnd
FROM `.sys/query_metrics_one_minute`
ORDER BY SumUpdateRows DESC LIMIT 10
```

Недавние запросы, прочитавшие больше всего байт за минуту:

```
SELECT
    IntervalEnd,
    SumReadBytes,
    MinReadBytes,
    SumReadBytes / Count as AvgReadBytes,
    MaxReadBytes,
    QueryText
FROM `.sys/query_metrics_one_minute`
WHERE SumReadBytes > 0
ORDER BY IntervalEnd DESC, SumReadBytes DESC
LIMIT 100
```

Подробная информация о запросах сессий

Следующее системное представление содержит подробную информацию о текущих сессиях и запросах, выполняющихся в них.

- `.sys/query_sessions` — содержит информацию о сессиях и выполняемых в них запросах.

Колонка	Описание
`SessionId`	Уникальный идентификатор сессии. Тип: `Utf8`. Ключ: `0`.
`NodeId`	Идентификатор узла, на котором запущена сессия. Тип: `Uint32`.

`State`	Текущее состояние сессии. Тип: `Utf8`.
`Query`	Текст последнего или текущего выполняемого запроса. Тип: `Utf8`.
`QueryCount`	Количество запросов, выполненных в рамках данной сессии. Тип: `Uint32`.
`ClientAddress`	Сетевой адрес клиента, инициировавшего сессию. Тип: `Utf8`.
`ClientPID`	Идентификатор процесса (PID) клиентского приложения. Тип: `Utf8`.
`ClientUserAgent`	Информация о клиентском ПО (User-Agent). Тип: `Utf8`.
`ClientSdkBuildInfo`	Информация о сборке SDK клиента. Тип: `Utf8`.
`ApplicationName`	Имя приложения, указанное клиентом при подключении. Тип: `Utf8`.
`SessionStartAt`	Время начала (создания) сессии. Тип: `Timestamp`.
`QueryStartAt`	Время начала выполнения текущего запроса. Тип: `Timestamp`.
`StateChangeAt`	Время последнего изменения состояния сессии. Тип: `Timestamp`.
`UserSID`	Security ID пользователя, владеющего сессией. Тип: `Utf8`.
`WmPoolId`	Идентификатор пула Workload Manager, в котором выполняется запрос сессии. Тип: `Utf8`.
`WmState`	Состояние запроса в Workload Manager. Тип: `Utf8`.
`WmEnterTime`	Время, когда запрос перешел в статус PENDING или DELAYED. Тип: `Timestamp`.
`WmExitTime`	Время, когда запрос передан на выполнение. Тип: `Timestamp`.

Возможные значения поля `WmState`:

- `NONE` - Не обрабатывается.
- `PENDING` - Обрабатывается (в процессе классификации/маршрутизации).
- `DELAYED` - В очереди.
- `EXITED` - Передан на исполнение.

Возможные значения поля `State`:

- `IDLE` - Сессия ожидает запроса.
- `EXECUTING` - Выполняется запрос.

Примеры запросов

Просмотр всех активных сессий:

```
SELECT * FROM `.sys/query_sessions`
```

Топ-20 самых длительных выполняющихся запросов:

```
SELECT
    Query,
    SessionId,
    NodeId,
    QueryStartAt
FROM `.sys/query_sessions`
WHERE State = 'EXECUTING'
ORDER BY QueryStartAt ASC
LIMIT 20
```

Поиск сессий приложения с фильтрацией по пулу Workload Manager:

```
SELECT
    SessionId,
    Query,
    State,
    WmState,
    ClientAddress
FROM `.sys/query_sessions`
```

```
WHERE ApplicationName = 'my_analytics_app'
  AND WmPoolId = 'heavy_queries'
```

Завершение сессии

Чтобы завершить выбранную сессию вместе с выполняющимся запросом, скопируйте её `SessionId` и отправьте отдельный запрос `KILL SESSION`:

```
KILL SESSION `ydb://session/3?node_id=52910&id=MWFmNGYwYTAtYzJkN2RhOWEtZmFkMDhlMTUtZjU0ZDE0OTA%3D`;
```

Замените ID в примере значением из результата выборки. Необходимые права и результат операции описаны в справочнике команды.

Кэш компиляции запросов

Следующее системное представление содержит информацию о запросах, хранящихся в кэше компиляции на всех нодах кластера:

- `compile_cache_queries` — содержит информацию о запросах в кэше компиляции всех нод кластера.

> **Важно**
>
> Системное представление `compile_cache_queries` недоступно в режиме serverless.

У каждой ноды кластера есть собственный кэш скомпилированных запросов. Этот кэш используется всеми сессиями, запущенными на данной ноде: если во время выполнения запроса его текст уже есть в кэше, дополнительная компиляция не требуется.

Системное представление `compile_cache_queries` позволяет получить данные о состоянии кэша на всех или выбранных нодах кластера. При выполнении запроса к этому представлению запросы отправляются на каждую ноду, если только не задано ограничение по `NodeId` в условии `WHERE`. В ответ возвращается актуальная информация о содержимом кэша для каждой ноды, а итоговая таблица объединяет полученные данные.

Структура представления:

Колонка	Описание
`NodeId`	Идентификатор ноды, на которой хранится запрос в кэше. Тип: `Uint32`. Ключ: `0`.
`QueryId`	Уникальный идентификатор запроса в кэше ноды. Тип: `Utf8`. Ключ: `1`.
`Query`	Текст запроса. Если запрос больше 10 КБ, он обрезается. Тип: `Utf8`.
`AccessCount`	Количество попаданий по тексту запроса в кэше. Тип: `Uint64`.
`CompiledAt`	Время компиляции запроса. Тип: `Timestamp`.
`UserSID`	Security ID пользователя, от имени которого был скомпилирован запрос. Может быть пустым для системных запросов. Тип: `Utf8`.
`LastAccessedAt`	Время последнего обращения к результату компиляции запроса в кэше. Тип: `Timestamp`.
`CompilationDurationMs`	Длительность компиляции запроса в миллисекундах. Тип: `Uint64`.
`Warnings`	Предупреждения, возникшие при компиляции запроса. Тип: `Utf8`.
`Metadata`	Типы параметров запроса в формате JSON. Содержит ключ `parameters` с именами параметров и их типами. Тип: `Utf8`.
`IsTruncated`	Флаг, указывающий, был ли текст запроса обрезан из-за превышения лимита в 10 КБ. Тип: `Bool`.
`QueryType`	Тип запроса, значение одно из: `QUERY_TYPE_SQL_DML` — Table Service `QUERY_TYPE_SQL_GENERIC_QUERY` — Query Service `QUERY_TYPE_SQL_GENERIC_CONCURRENT_QUERY` — Query Service в конкурентном режиме Для старых записей может быть пустым. Тип: `Utf8`.
`Syntax`	Синтаксис запроса, значение одно из: `SYNTAX_YQL_V1` — YQL `SYNTAX_UNSPECIFIED` — для старых записей без информации о синтаксисе `SYNTAX_PG` — устаревшее значение для записей, скомпилированных до удаления экспериментальной PostgreSQL-совместимости; новые запросы с этим синтаксисом не принимаются Тип: `Utf8`.

Примеры запросов

Просмотр всех запросов в кэше компиляции:

```
SELECT * FROM `.sys/compile_cache_queries`
```

Топ-20 самых популярных запросов по количеству обращений с 3 нод:

```
SELECT
    Query,
    SUM(AccessCount) AS Hits
FROM `.sys/compile_cache_queries`
WHERE NodeId IN (50000, 50001, 50003)
GROUP BY Query
ORDER BY Hits DESC
LIMIT 20
```

Статистика активности по пользователям:

```
SELECT
    UserSID,
    COUNT(DISTINCT QueryId) AS Plans,
    SUM(AccessCount) AS Hits,
    AVG(CompilationDurationMs) AS AvgCompileMs
FROM `.sys/compile_cache_queries`
GROUP BY UserSID
ORDER BY Hits DESC
```

Поиск запросов с длительной компиляцией:

```
SELECT
    Query,
    NodeId,
    CompilationDurationMs,
    AccessCount
FROM `.sys/compile_cache_queries`
WHERE CompilationDurationMs > 1000
ORDER BY CompilationDurationMs DESC
```

История перегруженных партиций

Следующие системные представления содержат историю моментов высокой нагрузки на отдельные партиции таблиц БД:

- `top_partitions_one_minute` — данные разбиты на минутные интервалы, содержит историю за последние 6 часов;
- `top_partitions_one_hour` — данные разбиты на часовые интервалы, содержит историю за последние 2 недели.

В представления попадают партиции с пиковой нагрузкой более 70 % (`CPUCores` > 0,7). В пределах одного интервала партиции ранжированы по пиковому значению нагрузки.

Оба представления содержат одинаковый набор полей:

Ключами представления являются:

- `IntervalEnd` - момент окончания интервала;
- `Rank` - ранг партиции по пиковой нагрузке `CPUCores` в этом интервале.

Например, если в таблице есть 10 партиций, то `top_partitions_one_hour` для часового интервала `"20.12.2024 10:00-11:00"` выдаст 10 строк, отсортированных по порядку убывания `CPUCores`. У них будет `Rank` от 1 до 10 и одинаковый `IntervalEnd` `"20.12.2024 11:00"`.

Колонка	Описание
`IntervalEnd`	Момент окончания минутного или часового интервала, за который собрана статистика. Тип: `Timestamp`. Ключ: `0`.
`Rank`	Ранг партиции в пределах интервала (по `CPUCores`). Тип: `Uint32`. Ключ: `1`.
`TabletId`	Идентификатор таблетки, обслуживающей партицию. Тип: `Uint64`.
`FollowerId`	Идентификатор реплики таблетки партиции. Значение 0 означает лидера. Тип: `Uint32`
`Path`	Полный путь к таблице. Тип: `Utf8`.
`PeakTime`	Момент пикового значения в пределах интервала. Тип: `Timestamp`.
`CPUCores`	Пиковое значение нагрузки на партицию (доля времени ядра процессора, затраченного актором партиции). Тип: `Double`.

`NodeId`	Идентификатор ноды, на которой находилась партиция в момент пика. Тип: `Uint32`.
`DataSize`	Приблизительный размер партиции в байтах в момент пика. Тип: `Uint64`.
`RowCount`	Приблизительное количество строк в момент пика. Тип: `Uint64`.
`IndexSize`	Размер индекса партиции в таблетке в момент пика. Тип: `Uint64`.
`InFlightTxCount`	Количество транзакций, находящихся в процессе исполнения в момент пика. Тип: `Uint32`.

Примеры запросов

Следующий запрос выводит партиции с потреблением CPU более 70% в указанном интервале времени, с идентификаторами таблеток и их размерами на момент превышения. Запрос выполняется к представлению `.sys/top_partitions_one_minute`, которое содержит данные за последние 6 часов с разбиением по минутным интервалам:

```
SELECT
    IntervalEnd,
    CPUCores,
    Path,
    TabletId,
    DataSize
FROM `.sys/top_partitions_one_minute`
WHERE CPUCores > 0.7
AND IntervalEnd BETWEEN Timestamp("2000-01-01T00:00:00Z") AND Timestamp("2099-12-31T00:00:00Z")
ORDER BY IntervalEnd desc, CPUCores desc
```

Следующий запрос выводит партиции с потреблением CPU более 90% в указанном интервале времени, с идентификаторами таблеток и их размерами на момент превышения. Запрос выполняется к представлению `.sys/top_partitions_one_hour`, которое содержит данные за последние 2 недели с разбиением по часовым интервалам:

```
SELECT
    IntervalEnd,
    CPUCores,
    Path,
    TabletId,
    DataSize
FROM `.sys/top_partitions_one_hour`
WHERE CPUCores > 0.9
AND IntervalEnd BETWEEN Timestamp("2000-01-01T00:00:00Z") AND Timestamp("2099-12-31T00:00:00Z")
ORDER BY IntervalEnd desc, CPUCores desc
```

История партиций со сломанными блокировками

Следующие системные представления содержат историю моментов с ненулевым числом сломанных блокировок `LocksBroken` в отдельных партициях таблиц БД:

- `top_partitions_by_tli_one_minute` — данные разбиты на минутные интервалы, содержит историю за последние 6 часов;
- `top_partitions_by_tli_one_hour` — данные разбиты на часовые интервалы, содержит историю за последние 2 недели.

Представления выдают топ-10 партиций с ненулевым числом сломанных блокировок `LocksBroken`. В пределах одного интервала партиции ранжированы по числу сломанных блокировок `LocksBroken`.

Ключами представлений являются:

- `IntervalEnd` - момент окончания интервала;
- `Rank` - ранг партиции по числу сломанных блокировок `LocksBroken` в этом интервале.

Например, `top_partitions_by_tli_one_hour` для часового интервала `"20.12.2024 10:00-11:00"` выдаст 10 строк, отсортированных по порядку убывания `LocksBroken`. У них будет `Rank` от 1 до 10 и одинаковый `IntervalEnd` `"20.12.2024 11:00"`.

Оба представления содержат одинаковый набор полей:

Колонка	Описание
`IntervalEnd`	Момент окончания минутного или часового интервала, за который собрана статистика. Тип: `Timestamp`. Ключ: `0`.
`Rank`	Ранг партиции в пределах интервала (по `LocksBroken`). Тип: `Uint32`. Ключ: `1`.
`TabletId`	Идентификатор таблетки, обслуживающей партицию. Тип: `Uint64`.
`FollowerId`	Идентификатор реплики таблетки партиции. Значение 0 означает лидера. Тип: `Uint32`
`Path`	Полный путь к таблице. Тип: `Utf8`.

`LocksAcquired`	Число установленных блокировок "на диапазон ключей" в данном интервале. Тип: `Uint64`.
`LocksWholeShard`	Число установленных блокировок "на всю партицию" в данном интервале. Тип: `Uint64`.
`LocksBroken`	Число сломанных блокировок в данном интервале. Тип: `Uint64`.
`NodeId`	Идентификатор ноды, на которой находилась партиция в момент пика. Тип: `Uint32`.
`DataSize`	Приблизительный размер партиции в байтах в момент пика. Тип: `Uint64`.
`RowCount`	Приблизительное количество строк в момент пика. Тип: `Uint64`.
`IndexSize`	Размер индекса партиции в таблетке в момент пика. Тип: `Uint64`.

Примеры запросов

Следующий запрос выводит партиции в указанном интервале времени, с идентификаторами таблеток и числом сломанных блокировок. Запрос выполняется к представлению `.sys/top_partitions_by_tli_one_minute`:

```
SELECT
    IntervalEnd,
    LocksBroken,
    Path,
    TabletId
FROM `.sys/top_partitions_by_tli_one_hour`
WHERE IntervalEnd BETWEEN Timestamp("2000-01-01T00:00:00Z") AND Timestamp("2099-12-31T00:00:00Z")
ORDER BY IntervalEnd desc, LocksBroken desc
```

Информация о пулах ресурсов

Системное представление `resource_pools` содержит информацию о настройках пулов ресурсов.

Структура системного представления:

Колонка	Описание
`Name`	Имя пула ресурсов. Тип: `Utf8`. Ключ: `0`.
`ConcurrentQueryLimit`	Максимальное количество параллельно выполняющихся запросов в пуле ресурсов. Тип: `Int32`.
`QueueSize`	Максимальный размер очереди ожидания. Тип: `Int32`.
`DatabaseLoadCpuThreshold`	Порог загрузки CPU всей базы данных, в процентах, после которого запросы не отправляются на выполнение и остаются в очереди. Тип: `Double`.
`TotalCpuLimitPercentPerNode`	Процент доступного CPU, который могут использовать все запросы на узле в данном пуле ресурсов. Тип: `Double`.

Пример

Следующий запрос выводит информацию о настройках пула ресурсов с именем `default`:

```
SELECT
    Name,
    ConcurrentQueryLimit,
    QueueSize,
    DatabaseLoadCpuThreshold,
    TotalCpuLimitPercentPerNode
FROM `.sys/resource_pools`
WHERE Name = "default";
```

Информация о классификаторах пулов ресурсов

Системное представление `resource_pools_classifiers` содержит информацию о настройках классификаторов пулов ресурсов.

Структура системного представления:

Колонка	Описание
`Name`	Имя классификатора пула ресурсов. Тип: `Utf8`. Ключ: `0`.

`Rank`	Приоритет выбора классификатора пулов ресурсов. Тип: `Int64`.
`MemberName`	Пользователь или группа пользователей, которые будут отправлены в указанный пул ресурсов. Тип: `Utf8`.
`ResourcePool`	Имя пула ресурсов, в который будут отправлены запросы. Тип: `Utf8`.

Пример

Следующий запрос выводит информацию о настройках классификатора пула ресурсов с именем `olap`:

```
SELECT
    Name,
    Rank,
    MemberName,
    ResourcePool
FROM `.sys/resource_pools_classifiers`
WHERE Name = "olap";
```

Пользователи, группы и права доступа

Следующие системные представления содержат информацию о пользователях, группах доступа, членстве пользователей в группах, а также о предоставленных правах доступа группам или непосредственно пользователям.

Информация о пользователях

Представление `auth_users` содержит список локальных пользователей YDB. В него не входят пользователи, аутентифицированные через внешние системы, такие как LDAP.

Полный доступ к этому представлению имеют администраторы. Обычные пользователи могут просматривать только свои собственные данные.

Структура таблицы:

Колонка	Описание
`Sid`	SID пользователя. Тип: `Utf8`. Ключ: `0`.
`IsEnabled`	Указывает, разрешён ли вход данному пользователю; используется для явной блокировки администратором. Независим от `IsLockedOut`. Тип: `Bool`.
`IsLockedOut`	Указывает, что данный пользователь автоматически заблокирован из-за превышения количества неудачных аутентификаций. Не зависит от `IsEnabled`. Тип: `Bool`.
`CreatedAt`	Время создания пользователя. Тип: `Timestamp`.
`LastSuccessfulAttemptAt`	Время последней успешной аутентификации. Тип: `Timestamp`.
`LastFailedAttemptAt`	Время последней неудачной аутентификации. Тип: `Timestamp`.
`FailedAttemptCount`	Количество неудачных аутентификаций. Тип: `Uint32`.
`PasswordHash`	JSON-строка, содержащая хеш пароля, соль и алгоритм хеширования. Тип: `Utf8`.

Информация о группах

Представление `auth_groups` содержит список групп доступа.

Доступ к этому представлению имеют только администраторы.

Структура таблицы:

Колонка	Описание
`Sid`	SID группы. Тип: `Utf8`. Ключ: `0`.

Информация о членстве в группах

Представление `auth_group_members` содержит информацию о членстве в группах доступа.

Доступ к этому представлению имеют только администраторы.

Структура таблицы:

Колонка	Описание

`GroupSid`	SID группы. Тип: `Utf8`. Ключ: `0`.
`MemberSid`	SID участника группы. Может быть указан как SID пользователя, так и SID группы. Тип: `Utf8`. Ключ: `1`.

Информация о правах доступа

Представления содержат список выданных прав доступа.

Включают два представления:

- `auth_permissions`: Явно выданные права доступа.
- `auth_effective_permissions`: Эффективные права доступа с учётом наследования.

Пользователю в данном представлении отображаются только те объекты доступа, на которые у него есть право `ydb.granular.describe_schema`.

Структура таблицы:

Колонка	Описание
`Path`	Путь к объекту доступа. Тип: `Utf8`. Ключ: `0`.
`Sid`	SID субъекта доступа. Тип: `Utf8`. Ключ: `1`.
`Permission`	Название права доступа YDB. Тип: `Utf8`. Ключ: `2`.

Примеры запросов

Получение явно предоставленных прав на объект доступа - таблицу `my_table`:

```
SELECT *
FROM `.sys/auth_permissions`
WHERE Path = "my_table"
```

Получение эффективных прав на объект доступа - таблицу `my_table`:

```
SELECT *
FROM `.sys/auth_effective_permissions`
WHERE Path = "my_table"
```

Получение прав, предоставленных пользователю `user3`:

```
SELECT *
FROM `.sys/auth_permissions`
WHERE Sid = "user3"
```

Информация о владельцах объектов доступа

Представление `auth_owners` отображает информацию о владельцах объектов доступа.

Пользователю в данном представлении отображаются только те объекты доступа, на которые ему предоставлено право `ydb.granular.describe_schema`.

Структура таблицы:

Колонка	Описание
`Path`	Путь к объекту доступа. Тип: `Utf8`. Ключ: `0`.
`Sid`	SID владельца объекта доступа. Тип: `Utf8`.

Потоковые запросы

Просмотр информации о потоковых запросах

Системное представление `streaming_queries` содержит информацию о всех созданных потоковых запросах.

Пользователю в данном представлении отображаются только те потоковые запросы, на которые ему предоставлено право `ydb.granular.describe_schema`.

Структура таблицы:

Колонка	Описание

`Path`	Полный путь к запросу. Тип: `Utf8`. Ключ: `0`.
`Status`	Статус выполнения запроса, значение одно из: `CREATING` - запрос создается `CREATED` - запрос создан, но не запущен `STARTING` - запрос запускается `RUNNING` - запрос запущен `STOPPING` - запрос останавливается `STOPPED` - запрос остановлен `DELETING` - запрос удаляется `SUSPENDED` - запрос завершился с ошибкой и ожидает backoff для ретрая `COMPLETED` - запрос успешно завершил выполнение `FAILED` - запуск потокового запроса завершился с ошибкой Тип: `Utf8`
`Issues`	Ошибки выполнения запроса в формате JSON Тип: `Utf8`
`Plan`	План запроса в формате JSON Тип: `Utf8`
`Ast`	AST запроса Тип: `Utf8`
`Text`	Текст запроса Тип: `Utf8`
`Run`	Запущен ли запрос пользователем в данный момент Тип: `Bool`
`ResourcePool`	Имя пула ресурсов, к которому был привязан запрос (см. пример) Тип: `Utf8`
`RetryCount`	Число перезапусков запроса Тип: `Uint64`
`LastFailAt`	Время последней ошибки исполнения запроса Тип: `Timestamp`
`SuspendedUntil`	Время, когда будет предпринята попытка возобновить остановившийся запрос Тип: `Timestamp`

Change Data Capture

> **Важно**
>
> Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Change Data Capture (CDC) позволяет отслеживать изменения данных в строковых таблицах. YDB предоставляет доступ к потоку изменений, тем самым потребители данных могут отслеживать происходящие изменения почти в реальном времени.

Включение и выключение CDC

CDC представлен объектом схемы данных — потоком изменений, который может быть добавлен к таблице или удален директивами ADD CHANGEFEED и DROP CHANGEFEED операции YQL `ALTER TABLE`.

Чтение из топика

Прежде чем приступить к чтению данных, необходимо добавить потребителя данных (consumer). Ниже приведен пример команды, которая добавит потребителя данных с именем `my_consumer` в поток данных с именем `updates_feed` строковой таблицы `table` в директории `my`:

```
ydb topic consumer add \
  my/table/updates_feed \
  --consumer=my_consumer
```

Теперь можно воспользоваться созданным потребителем данных и запустить процесс отслеживания. Ниже приведен пример команды, с помощью которой можно отслеживать изменения данных в интерфейсе командной строки:

```
ydb topic read \
  my/table/updates_feed \
  --consumer=my_consumer \
  --format=newline-delimited \
  --wait
```

Влияние на производительность записи в строковой таблицы

При записи в строковую таблицу с включенным CDC возникают дополнительные накладные расходы на следующие операции:

- формирование записи и сохранение ее в поток изменений;
- хранением записей в потоке изменений;
- в некоторых режимах (например, `OLD_IMAGE`, `NEW_AND_OLD_IMAGES`) возникает необходимость предварительного чтения данных, даже если пользовательский запрос этого не предполагает.

Это может вызывать увеличение времени исполнения запросов, а также превышение лимитов на размер хранимых данных.

В реальных сценариях включение CDC практически не влияет время выполнения запросов (вне зависимости от режима), так как почти все данные, необходимые для формирования записей, находятся в кеше, а сами записи отправляются в топик асинхронно. Однако, фоновая активность по отправке записей незначительно (на 1–10%) увеличивает потребление CPU.

При создании потока изменений для строковой таблицы количество партиций его хранилища (топика) определяется на основании текущего количества партиций таблицы. Если количество партиций исходной таблицы значительно изменяется (например, после загрузки большого объёма данных, или в результате интенсивных обращений), то возникает дисбаланс между партициями таблицы и партициями топика. Такой дисбаланс также может приводить к увеличению времени исполнения запросов на модификацию данных в таблице, либо к излишним накладным расходам на хранение потока изменений. Для устранения дисбаланса можно пересоздать поток изменений.

Нагрузочное тестирование

В качестве генератора нагрузки можно воспользоваться встроенной в YDB CLI возможностью эмуляции работы интернет-магазина:

1. Инициализируйте тест.
2. Добавьте поток изменений:

   ```
   ALTER TABLE `orders` ADD CHANGEFEED `updates` WITH (
       FORMAT = 'JSON',
       MODE = 'UPDATES'
   );
   ```

3. Создайте потребителя данных:

   ```
   ydb topic consumer add \
     orders/updates \
     --consumer=my_consumer
   ```

4. Запустите процесс отслеживания:

   ```
   ydb topic read \
     orders/updates \
     --consumer=my_consumer \
     --format=newline-delimited \
     --wait
   ```

5. Подайте нагрузку.

В интерфейсе командной строки появится поток изменений:

```
...
{"update":{"created":"2022-06-24T11:35:00.000000Z","customer":"Name366"},"key":[1319569999286404932]}
{"update":{"created":"2022-06-24T11:35:00.000000Z","customer":"Name3894"},"key":[452209497351143909]}
{"update":{"created":"2022-06-24T11:35:00.000000Z","customer":"Name7773"},"key":[237797889418385025 8]}
...
```

Разделяемый читатель топика: устройство и ограничения

Разделяемый (общий) читатель — модель чтения из топика, при которой за потребителем закрепляется одно или несколько сообщений, а не целая партиция. Это позволяет нескольким потребителям параллельно обрабатывать сообщения из одной партиции и использовать топики в роли очередей сообщений. Типичный сценарий — обмен сообщениями между микросервисами; чтение по Amazon SQS API работает через разделяемый читатель.

Общее описание модели, настройки читателя, политики DLQ и порядка сообщений см. в разделе Разделяемый (общий) читатель. В этой статье описано внутреннее устройство механизма: как сервер отслеживает состояние обработки, распределяет запросы на чтение по партициям, какие ограничения следует учитывать при проектировании высоконагруженных очередей и какое потребление ресурсов создаёт инфлайт.

Особенность реализации

Для каждой пары «разделяемый (общий) читатель — партиция топика» сервер поддерживает собственное состояние обработки сообщений. Состояние состоит из непрерывного блока сообщений из партиции топика — инфлайта. Первым сообщением инфлайта является сообщение, которое ещё не обработано, находится в обработке или ждёт перемещения в DLQ. Если первое сообщение успешно обработано, то начало блока перемещается на следующее сообщение. Сообщения для обработки отдаются только из сообщений, находящихся в инфлайте.

В инфлайте одной партиции может находиться до 120000 сообщений. Это означает, что если из одной партиции читает большое количество потребителей (больше 120000), то сообщения для обработки будут получать не более 120000 потребителей, даже если в партиции сообщений больше. Увеличение количества партиций в топике увеличивает общий размер инфлайта — общий инфлайт топика равен количеству партиций, умноженному на 120000 сообщений. Если необходимо обрабатывать большое количество сообщений одновременно, то следует увеличить количество партиций топика.

Если топик состоит из нескольких партиций, то запросы на чтение равномерно распределяются по всем партициям. Может случиться ситуация, что запрос попадёт в партицию, в которой отсутствуют сообщения. В этом случае потребитель получит ответ, что сообщений для обработки нет. Если в партиции долгое время нет сообщений для обработки, то она исключается из распределения запросов на чтение. Партиция вернётся в распределение сразу, как только в ней появятся сообщения для обработки.

Для разделяемых (общих) читателей с сохранением порядка сообщений это означает, что если в партицию будет записано подряд большое количество сообщений из одной message-group-id (больше 120000 сообщений), то весь инфлайт займут сообщения из одной message-group-id, и сообщения из других групп выдаваться не будут, пока в инфлайте не появятся сообщения из других групп.

Честные очереди

При выдаче сообщений на чтение сервер учитывает `message-group-id` сообщений, находящихся в инфлайте. Стараясь отдавать сообщения из разных групп, сервер распределяет нагрузку между писателями более равномерно.

Это позволяет сглаживать пики от отдельных писателей: если один писатель отправляет в очередь гораздо больше сообщений, чем остальные, обработка сообщений от «малых» писателей получает приоритет и не блокируется потоком от одного источника.

Для FIFO-очередей и разделяемых читателей с сохранением порядка сообщений при этом гарантируется порядок сообщений внутри одного `message-group-id`: потребитель всегда получает сообщения группы в том порядке, в котором они были записаны.

Потребление ресурсов

Состояние инфлайта потребляет ресурсы оперативной памяти и диска. На каждое сообщение в инфлайте сервер хранит служебную информацию объёмом около 32 байт — это не тело сообщения, а метаданные о его статусе обработки.

Оперативная память

Весь инфлайт хранится в памяти сервера. Для оценки потребления памяти удобно считать от миллиона сообщений: **1 млн сообщений в инфлайте ≈ 32 МБ** оперативной памяти.

Например, топик с 10 партициями и одним разделяемым читателем при полностью заполненном инфлайте (120000 сообщений на партицию) удерживает до 1,2 млн сообщений в инфлайте — около **38 МБ** памяти только на служебное состояние. На одну пару «разделяемый (общий) читатель — партиция» при максимальном инфлайте приходится около **3,8 МБ**. Если на топике несколько разделяемых читателей, потребление памяти умножается на их количество.

При проектировании учитывайте:

- чем больше партиций и читателей, тем выше потолок потребления памяти;
- чем дольше сообщения находятся в обработке (большое время обработки сообщения, медленные потребители), тем дольше они остаются в инфлайте и занимают память;
- увеличение числа партиций расширяет параллелизм, но одновременно увеличивает суммарный объём инфлайта.

Диск

На диске также сохраняется информация о статусе сообщений — от **32 байта на сообщение**. Хранение состоит из двух частей:

1. **Снапшот** — периодически записывается полное состояние всех сообщений в инфлайте.
2. **Журнал изменений** — сервер обрабатывает ожидающие запросы на чтение пачками и при обработке каждой пачки дописывает в журнал изменения статусов сообщений.

Следовательно, высокая частота чтения и большой инфлайт увеличивают объём дисковых записей. При планировании ёмкости кластера закладывайте ресурсы диска наряду с оперативной памятью — особенно для топиков с большим числом партиций, несколькими разделяемыми читателями и длительным временем обработки сообщений.

Управление YDB с помощью Terraform

С помощью Terraform можно создавать, удалять и изменять следующие объекты внутри кластера YDB:

- строковые таблицы;
- вторичные индексы для строковых таблиц;
- потоки изменений строковых таблиц;
- топики.

> **Важно**
>
> На данный момент YDB провайдер для Terraform находится на стадии разработки и его функциональность будет расширяться.

Для начала работы необходимо:

1. Развернуть кластер YDB
2. Создать базу данных (описано в п.1 для соответствующего типа развертывания кластера)
3. Установить Terraform
4. Установить и настроить Terraform provider for YDB

Настройка Terraform провайдера для работы с YDB

1. Нужно скачать код провайдера
2. Собрать провайдер, выполнив `$ make local-build` в корневой директории кода провайдера. Для этого вам необходимо дополнительно установить утилиту make и go; провайдер установится в папку плагинов Terraform - `~/.terraform.d/plugins/terraform.storage.ydb.tech/...`
3. Добавить провайдер в `~/.terraformrc`, дописав в секцию `provider_installation` следующее содержание (если такой секции ещё не было, то создать):

```
provider_installation {
  direct {
    exclude = ["terraform.storage.ydb.tech/*/*"]
  }

  filesystem_mirror {
    path    = "/PATH_TO_HOME/.terraform.d/plugins"
    include = ["terraform.storage.ydb.tech/*/*"]
  }
}
```

4. Далее настраиваем сам YDB провайдер для работы (например в файле `provider.tf` в рабочей директории):

```
terraform {
  required_providers {
    ydb = {
      source = "terraform.storage.ydb.tech/provider/ydb"
    }
  }
  required_version = ">= 0.13"
}

provider "ydb" {
  token = "<TOKEN>"
  //OR for static credentials
  user = "<USER>"
  password = "<PASSWORD>"
}
```

Где:

- `token` - указывается токен доступа к БД, если используется аутентификация, например, с использованием стороннего IAM провайдера.
- `user` - имя пользователя для доступа к базе данных в случае использования аутентификации по логину и паролю.
- `password` - пароль для доступа к базе данных в случае использования аутентификации по логину и паролю.

Использование Terraform провайдера YDB

Для применения изменений в terraform ресурсах используются следующие команды:

1. `terraform init` - инициализация модуля terraform (выполняется в директории ресурсов terraform).
2. `terraform validate` - проверка синтаксиса конфигурационных файлов ресурсов terraform.
3. `terraform apply` - непосредственное применение конфигурации ресурсов terraform.

Для удобства использования файлы terraform рекомендуется именовать следующим образом:

1. `provider.tf` - содержит настройки самого terraform провайдера.
2. `main.tf` - содержит набор ресурсов для создания.

Соединение с базой данных (БД)

Для всех ресурсов, описывающих объекты схемы данных, необходимо задать реквизиты БД, в которой они размещаются. Для этого укажите один из двух аргументов:

- Строка соединения `connection_string` — выражение вида `grpc(s)://HOST:PORT/?database=/database/path`, где `grpc(s)://HOST:PORT/` эндпоинт, а `/database/path` — путь БД.

Например, `grpcs://example.com:2135?database=/Root/testdb0`.

- `database_endpoint` - используется при работе с ресурсом топиков (аналог `connection_string` при работе с ресурсами строковых таблиц).

> **Примечание**
>
> Если требуется, пользователь может передавать строку соединения с БД стандартными средствами Terraform - через variables.

Если вы используете создание ресурсов `ydb_table_changefeed` или `ydb_topic` и на сервере YDB не включена авторизация, то в конфиге БД config.yaml нужно указать:

```
...
pqconfig:
  require_credentials_in_new_protocol: false
  check_acl: false
```

Пример использования всех типов ресурсов YDB Terraform провайдера

Данный пример объединяет все типы ресурсов, которые доступны в YDB Terraform провайдере:

```
variable "db-connect" {
  type = string
  default = "grpc(s)://HOST:PORT/?database=/database/path" # нужно указать путь к базе данных
}

resource "ydb_table" "table" {
  path        = "1/2/3/tftest"
  connection_string = var.db-connect
  column {
    name = "a"
    type = "Utf8"
  }
  column {
    name = "b"
    type = "String"
  }
  column {
    name = "ttlBase"
    type = "Uint32"
  }
  ttl {
    column_name = "ttlBase"
    expire_interval = "P7D"
    unit = "milliseconds"
  }

  primary_key = ["b", "a"]

  partitioning_settings {
    auto_partitioning_min_partitions_count = 5
    auto_partitioning_max_partitions_count = 8
    auto_partitioning_partition_size_mb    = 256
    auto_partitioning_by_load              = true
  }
}

resource "ydb_table_index" "table_index" {
  table_path        = ydb_table.table.path
  connection_string = ydb_table.table.connection_string
  name              = "my_index"
  type              = "global_sync" # "global_async"
  columns           = ["a", "b"]

  depends_on = [ydb_table.table] # ссылка на ресурс создания строковой таблицы
}

resource "ydb_table_changefeed" "table_changefeed" {
  table_id = ydb_table.table.id
  name     = "changefeed"
  mode     = "NEW_IMAGE"
  format   = "JSON"
  consumer {
    name = "test"
    supported_codecs = ["raw", "gzip"]
  }

  depends_on = [ydb_table.table] # ссылка на ресурс создания строковой таблицы
}

resource "ydb_topic" "test" {
  database_endpoint = ydb_table.table.connection_string
  name              = "1/2/test"
  supported_codecs  = ["zstd"]

  consumer {
    name             = "test-consumer3"
    starting_message_timestamp_ms = 0
```

```
    supported_codecs = ["zstd","raw"]
  }

  consumer {
    name             = "test-consumer1"
    starting_message_timestamp_ms = 2000
    supported_codecs = ["zstd"]
  }

  consumer {
    name             = "test-consumer2"
    starting_message_timestamp_ms = 0
    supported_codecs = ["zstd"]
  }
}
```

Ниже будут подробно расписаны все ресурсы YDB Terraform провайдера

Строковая таблица

> **Важно**
>
> Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Для работы со строковыми таблицами используется ресурс `ydb_table`.

Пример:

```
resource "ydb_table" "ydb_table" {
  path = "path/to/table" # путь относительно корня базы
  connection_string = "grpc(s)://HOST:PORT/?database=/database/path" #пример подключения к БД
  column {
    name = "a"
    type = "Utf8"
    not_null = true
  }
  column {
    name = "b"
    type = "Uint32"
    not_null = true
  }
  column {
    name = "c"
    type = String
    not_null = false
  }
  column {
    name = "f"
    type = "Utf8"
  }
  column {
    name = "e"
    type = "String"
  }
  column {
    name = "d"
    type = "Timestamp"
  }
  primary_key = ["b", "a"]
}
```

Поддерживаются следующие аргументы:

- `path` — (обязательный) путь строковой таблицы, относительно корня базы (пример - `/path/to/table`).
- `connection_string` — (обязательный) строка соединения.
- `column` — (обязательный) свойства колонки (см. аргумент column).
- `family` — (необязательный) группа колонок (см. аргумент family).
- `primary_key` — (обязательный) первичный ключ таблицы, содержит упорядоченный список имён колонок первичного ключа.
- `ttl` — (необязательный) TTL (см. аргумент ttl).
- `partitioning_settings` — (необязательный) настройки партицирования (см. аргумент partitioning_settings).
- `key_bloom_filter` — (необязательный) (bool) использовать фильтра Блума для первичного ключа, значение по умолчанию - false.
- `read_replicas_settings` — (необязательный) настройки реплик для чтения.

column

Аргумент `column` описывает свойства колонки таблицы.

Важно

При помощи Terraform нельзя удалить колонку, можно только добавить. Чтобы удалить колонку, используйте средства YDB, затем удалите колонку из описания ресурса. При попытке "прокатки" изменений колонок строковой таблицы (смена типа, имени), Terraform не попытается их удалить, а попытается сделать update-in-place, но изменения применены не будут.

Пример:

```
column {
  name     = "column_name"
  type     = "Utf8"
  family   = "some_family"
  not_null = true
}
```

- `name` — (обязательный) имя колонки.
- `type` — (обязательный) тип данных YQL колонки. Допускается использовать простые типы колонок. Как пример, контейнерные типы не могут быть использованы в качестве типов данных колонок строковых таблиц.
- `family` — (необязательный) имя группы колонок (см. аргумент family).
- `not_null` — (необязательный) колонка не может содержать `NULL`. Значение по умолчанию: `false`.

family

Аргумент `family` описывает свойства группы колонок.

- `name` — (обязательный) имя группы колонок.
- `data` — (обязательный) тип устройства хранения для данных колонок этой группы.
- `compression` — (обязательный) кодек сжатия данных.

Пример:

```
family {
  name        = "my_family"
  data        = "ssd"
  compression = "lz4"
}
```

partitioning_settings

Аргумент `partitioning_settings` описывает настройки партицирования.

Пример:

```
partitioning_settings {
  auto_partitioning_min_partitions_count = 5
  auto_partitioning_max_partitions_count = 8
  auto_partitioning_partition_size_mb    = 256
  auto_partitioning_by_load              = true
}
```

- `uniform_partitions` — (необязательный) количество заранее аллоцированных партиций.
- `partition_at_keys` — (необязательный) партицирование по первичному ключу.
- `auto_partitioning_min_partitions_count` — (необязательный) минимально возможное количество партиций при автопартицировании.
- `auto_partitioning_max_partitions_count` — (необязательный) максимально возможное количество партиций при автопартицировании.
- `auto_partitioning_partition_size_mb` — (необязательный) задание значения автопартицирования по размеру в мегабайтах.
- `auto_partitioning_by_size_enabled` — (необязательный) включение автопартиционирования по размеру (bool), по умолчанию - включено (true).
- `auto_partitioning_by_load` — (необязательный) включение автопартицирования по нагрузке (bool), по умолчанию - выключено (false).

Подробнее про параметры и их значения по умолчанию написано по ссылкам выше.

ttl

Аргумент `ttl` описывает настройки Time To Live.

Пример:

```
ttl {
  column_name     = "column_name"
  expire_interval = "PT1H" # 1 час
  unit            = "seconds" # для числовых типов колонок (non ISO 8601)
}
```

- `column_name` — (обязательный) имя колонки для TTL.
- `expire_interval` — (обязательный) интервал в формате ISO 8601 (например, `P1D` — интервал длиной в 1 сутки, то есть 24 часа).
- `unit` — (необязательный) задается в случае, если колонка с ttl имеет числовой тип. Поддерживаемые значения:

- `seconds`
- `milliseconds`
- `microseconds`
- `nanoseconds`

Вторичный индекс строковой таблицы

Для работы с индексом строковой таблицы используется ресурс ydb_table_index.

Пример:

```
resource "ydb_table_index" "ydb_table_index" {
  table_path        = "path/to/table" # путь относительно корня базы
  connection_string = "grpc(s)://HOST:PORT/?database=/database/path" #пример подключения к БД
  name              = "my_index"
  type              = "global_sync" # "global_async"
  columns           = ["a", "b"]
  cover             = ["c"]
}
```

Поддерживаются следующие аргументы:

- `table_path` — путь строковой таблицы. Указывается, если не задан `table_id`.
- `connection_string` — строка соединения. Указывается, если не задан `table_id`.
- `table_id` - terraform-идентификатор строковой таблицы. Указывается, если не задан `table_path` или `connection_string`.
- `name` — (обязательный) имя индекса.
- `type` — (обязательный) тип индекса global_sync | global_async.
- `columns` — (обязательный) упорядоченный список имён колонок, участвующий в индексе.
- `cover` — (обязательный) список дополнительных колонок для покрывающего индекса.

Поток изменений строковой таблицы

> **Важно**
>
> Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Для работы с потоком изменений строковой таблицы используется ресурс `ydb_table_changefeed`.

Пример:

```
resource "ydb_table_changefeed" "ydb_table_changefeed" {
  table_id = ydb_table.ydb_table.id
  name     = "changefeed"
  mode     = "NEW_IMAGE"
  format   = "JSON"
}
```

Поддерживаются следующие аргументы:

- `table_path` — путь строковой таблицы. Указывается, если не задан `table_id`.
- `connection_string` — строка соединения. Указывается, если не задан `table_id`.
- `table_id` — terraform-идентификатор строковой таблицы. Указывается, если не задан `table_path` или `connection_string`.
- `name` — (обязательный) имя потока изменений.
- `mode` — (обязательный) режим работы потока изменений.
- `format` — (обязательный) формат потока изменений.
- `virtual_timestamps` — (необязательный) использование виртуальных таймстеймпов.
- `retention_period` — (необязательный) время хранения данных в формате ISO 8601.
- `consumer` — (необязательный) читатель потока изменений (см. аргумент #consumer).

consumer

Аргумент `consumer` описывает читателя потока изменений.

- `name` — (обязательный) имя читателя.
- `supported_codecs` — (необязательный) поддерживаемые кодек данных.
- `starting_message_timestamp_ms` — (необязательный) временная метка в формате UNIX timestamp в миллисекундах, с которой читатель начнет читать данные.

Примеры использования

Создание строковой таблицы в существующей БД

```
resource "ydb_table" "ydb_table" {
  # Путь до строковой таблицы
  path = "path/to/table" # путь относительно корня базы

  connection_string = "grpc(s)://HOST:PORT/?database=/database/path" #пример подключения к БД

  column {
```

```
    name = "a"
    type = "Uint64"
    not_null = true
  }
  column {
    name     = "b"
    type     = "Uint32"
    not_null = true
  }
  column {
    name = "c"
    type = String
    not_null = false
  }
  column {
    name = "f"
    type = "Utf8"
  }
  column {
    name = "e"
    type = "String"
  }
  column {
    name = "d"
    type = "Timestamp"
  }
  # Первичный ключ
  primary_key = [
    "a", "b"
  ]
}
```

Создание строковой таблицы, индекса и потока изменений

```
resource "ydb_table" "ydb_table" {
  # Путь до строковой таблицы
  path = "path/to/table" # путь относительно корня базы

  # ConnectionString до базы данных.
  connection_string = "grpc(s)://HOST:PORT/?database=/database/path" #пример подключения к БД

  column {
    name = "a"
    type = "Uint64"
    not_null = true
  }
  column {
    name     = "b"
    type     = "Uint32"
    not_null = true
  }
  column {
    name = "c"
    type = "Utf8"
  }
  column {
    name = "f"
    type = "Utf8"
  }
  column {
    name = "e"
    type = "String"
  }
  column {
    name = "d"
    type = "Timestamp"
  }

  # Первичный ключ
  primary_key = [
    "a", "b"
  ]


  ttl {
    column_name     = "d"
    expire_interval = "PT5S"
  }

  partitioning_settings {
    auto_partitioning_by_load = false
    auto_partitioning_partition_size_mb    = 256
    auto_partitioning_min_partitions_count = 6
    auto_partitioning_max_partitions_count = 8
  }

  read_replicas_settings = "PER_AZ:1"

  key_bloom_filter = true # Default = false
}
```

```
resource "ydb_table_changefeed" "ydb_table_changefeed" {
  table_id = ydb_table.ydb_table.id
  name = "changefeed"
  mode = "NEW_IMAGE"
  format = "JSON"

  consumer {
    name = "test_consumer"
  }

  depends_on = [ydb_table.ydb_table] # ссылка на ресурс создания строковой таблицы
}

resource "ydb_table_index" "ydb_table_index" {
  table_id = ydb_table.ydb_table.id
  name = "some_index"
  columns = ["c", "d"]
  cover = ["e"]
  type = "global_sync"

  depends_on = [ydb_table.ydb_table] # ссылка на ресурс создания строковой таблицы
}
```

Управление конфигурацией топиков YDB через Terraform

Для работы с топиками используется ресурс `ydb_topic`

> **Примечание**
>
> Топик не может быть создан в корне БД, нужно указать хотя бы один каталог в имени топика. При попытке создать топик в корне БД - провайдер выдаст ошибку.

Описание ресурса `ydb_topic`

Пример:

```
resource "ydb_topic" "ydb_topic" {
  database_endpoint = "grpcs://example.com:2135/?database=/Root/testdb0" #пример подключения к БД
  name              = "test/test1"
  supported_codecs  = ["zstd"]

  consumer {
    name             = "test-consumer1"
    starting_message_timestamp_ms = 0
    supported_codecs = ["zstd","raw"]
  }

  consumer {
    name             = "test-consumer2"
    starting_message_timestamp_ms = 2000
    supported_codecs = ["zstd"]
  }

  consumer {
    name             = "test-consumer3"
    starting_message_timestamp_ms = 0
    supported_codecs = ["zstd"]
  }
}
```

Поддерживаются следующие аргументы:

- `name` - (обязательный) имя топика.
- `database_endpoint` - (обязательный) полный путь до базы данных, например: `"grpcs://example.com:2135/?database=/Root/testdb0"`; аналог `connection_string` для строковых таблиц.
- `retention_period_ms` - длительность хранения данных в миллисекундах, значение по умолчанию - `86400000` (сутки).
- `partitions_count` - количество партиций, значение по умолчанию - `2`.
- `supported_codecs` - поддерживаемые кодеки сжатия данных, значение по умолчанию - `"gzip", "raw", "zstd"`.
- `consumer` - читатели для топика.

Описание потребителя данных `consumer`:

- `name` - (обязательное) имя читателя.
- `supported_codecs` - поддерживаемые кодировки сжатия данных, по умолчанию - `"gzip", "raw", "zstd"`.
- `starting_message_timestamp_ms` - временная метка в формате UNIX timestamp в миллисекундах, с которой читатель начнёт читать данные, по умаолчанию - 0, то есть с начала поставки.

Пользовательские атрибуты таблиц

Важно

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Вы можете хранить произвольную информацию в пользовательских атрибутах и обрабатывать ее в приложении, а также с помощью CLI.

Например, приложение использует таблицу в БД. В пользовательских атрибутах хранится версия схемы таблицы. При изменении схемы таблицы необходимо выполнить миграцию данных. Чтобы приложение при старте не применило миграцию, которая уже была выполнена, а также не забыло про миграцию, которую нужно применить, нужно проверить версию схемы в пользовательских атрибутах:

- В атрибуте версия `1`, при этом наше приложение умеет работать с версией `2` — применяем миграцию `1` > `2`, обновляем значение атрибута на `2`.
- В атрибуте версия `2`, наше приложение умеет работать с версией `2` — миграция не нужна.
- В атрибуте версия `2`, наше приложение умеет работать с версией `1` — завершаем работу аварийно с уведомлением о попытке запуска старой версии приложения на новой версии схемы данных.

С применением пользовательских атрибутов отпадает необходимость хранить версию схемы в отдельной таблице.

Установите пользовательский атрибут при создании таблицы

Go

Чтобы установить пользовательский атрибут при создании таблицы `series`, передайте ключ `scheme_version` и значение атрибута `1` в опции `options.WithAttribute` метода `CreateTable`:

```
err := client.Do(ctx, func(ctx context.Context, s table.Session) error {
  return s.CreateTable(ctx, "episodes",
    options.WithColumn("series_id", types.Optional(types.TypeUint64)),
    options.WithColumn("season_id", types.Optional(types.TypeUint64)),
    options.WithColumn("episode_id", types.Optional(types.TypeUint64)),
    options.WithColumn("title", types.Optional(types.TypeText)),
    options.WithPrimaryKeyColumn("series_id", "season_id", "episode_id"),
    options.WithAttribute("scheme_version", "1"),
  )
})
```

Установите пользовательский атрибут для существующей таблицы

Go

Чтобы установить пользовательский атрибут для существующей таблицы `series`, передайте ключ `scheme_version` и значение атрибута `1` в опции `options.WithAddAttribute` метода `AlterTable`:

```
err = db.Table().Do(ctx,
  func(ctx context.Context, s table.Session) (err error) {
    return s.AlterTable(ctx, path.Join(db.Name(), "series"),
      options.WithAddAttribute("scheme_version", "1"),
    )
  },
)
```

CLI

Чтобы установить пользовательский атрибут для существующей таблицы `series`, передайте ключ `scheme_version` и значение атрибута `1` в опции `--attribute` команды ydb table attribute add:

```
ydb table attribute add --attribute scheme_version=1 series
```

Измените пользовательский атрибут

Go

Чтобы изменить пользовательский атрибут при изменении схемы таблицы, передайте ключ `scheme_version` и новое значение атрибута `2` в опции `WithAlterAttribute` метода `AlterTable`:

```
err = db.Table().Do(ctx,
  func(ctx context.Context, s table.Session) (err error) {
    return s.AlterTable(ctx, path.Join(db.Name(), "series"),
      options.WithAddColumn("air_date", types.Optional(types.TypeUint64)),
      options.WithAlterAttribute("scheme_version", "2"),
    )
  },
)
```

CLI

Чтобы изменить пользовательский атрибут для существующей таблицы `series`, передайте ключ `scheme_version` и значение атрибута `2` в опции `--attribute` команды ydb table attribute add:

```
ydb table attribute add --attribute scheme_version=2 series
```

Просмотрите пользовательские атрибуты

Go

Чтобы получить информацию о схеме таблицы `series` включая данные о пользовательских атрибутах, используйте метод `table.Session.DescribeTable()`:

```
err := c.Do(ctx,
  func(ctx context.Context, s table.Session) error {
    description, err := s.DescribeTable(ctx, path.Join(prefix, "series"))
    if err != nil {
      return err
    }
    for k, v := range description.Attributes {
      log.Println(k, "=", v)
    }
    return nil
  },
)
```

CLI

Чтобы получить информацию о схеме таблицы `series` включая данные о пользовательских атрибутах, используйте команду ydb scheme describe:

```
ydb scheme describe series
```

Результат:

```
...
Attributes:
┌────────────────┬─────────┐
| Name           | Value |
├────────────────┼─────────┤
| scheme_version | 2     |
└────────────────┴─────────┘
...
```

Удалите пользовательский атрибут

Go

Чтобы удалить пользовательский атрибут, передайте ключ `scheme_version` в опции `WithDropAttribute` метода `AlterTable`:

```
err = db.Table().Do(ctx,
  func(ctx context.Context, s table.Session) (err error) {
    return s.AlterTable(ctx, path.Join(db.Name(), "series"),
      options.WithDropAttribute("scheme_version"),
    )
  },
)
```

CLI

Чтобы удалить пользовательский атрибут, передайте ключ `scheme_version` в опции `--attributes` команды ydb table attribute drop:

```
ydb table attribute drop --attributes scheme_version series
```

Workload Manager — управление потреблением ресурсов

Пулы ресурсов позволяют изолировать ресурсы баз данных между выполняемыми запросами или же настраивать стратегии распределения ресурсов в случае переподписки (запроса большего объёма ресурсов, чем доступно в системе). Все пулы ресурсов являются равноправными, без какой-либо иерархии, и влияют друг на друга только при общем дефиците ресурсов.

Например, одним из типовых сценариев изоляции ресурсов является разделение двух классов потребителей:

1. Регулярный роботный процесс, который раз в сутки строит отчёт.
2. Аналитики, которые выполняют ad hoc запросы.

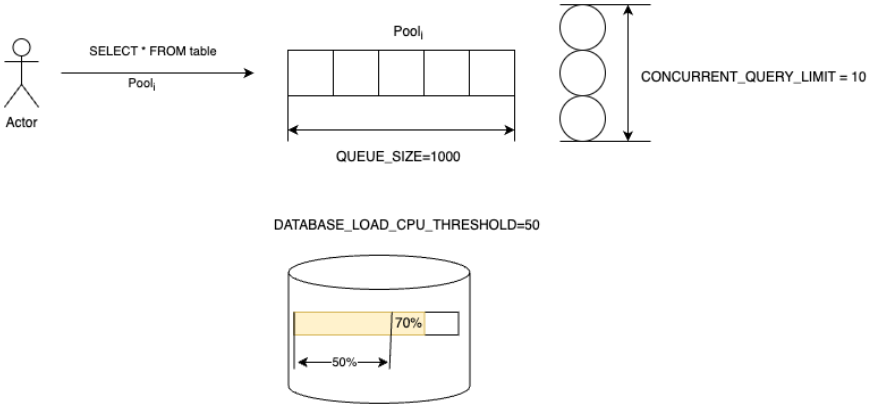
> **Важно**
>
> Представленная функциональность для управления потреблением ресурсами находится в стадии Preview.

Создание пула ресурсов

В примере ниже приведён синтаксис для создания отдельного пула ресурсов с именем "olap", в котором будут выполняться аналитические запросы:

```
CREATE RESOURCE POOL olap WITH (
    CONCURRENT_QUERY_LIMIT=10,
    QUEUE_SIZE=1000,
    DATABASE_LOAD_CPU_THRESHOLD=80,
    TOTAL_CPU_LIMIT_PERCENT_PER_NODE=70
)
```

Ознакомиться с полным перечнем параметров пулов ресурсов можно в справке по Параметры. Часть параметров является глобальными для всей базы данных (например, `CONCURRENT_QUERY_LIMIT`, `QUEUE_SIZE`, `DATABASE_LOAD_CPU_THRESHOLD`), а другие — применяются только к одному вычислительному узлу (например, `TOTAL_CPU_LIMIT_PERCENT_PER_NODE`).



Рассмотрим на примере выше, что на самом деле означают эти параметры и как они будут влиять на распределение ресурсов. Допустим, в базе данных YDB выделено узлов по . В сумме в такой базе данных . Тогда на каждом узле для пула ресурсов с именем `olap` будет выделено:

В сумме, при равномерном распределении ресурсов на всю базу данных, пулу ресурсов будет выделено:

Как работает CONCURRENT_QUERY_LIMIT и QUEUE_SIZE

Допустим, в пуле ресурсов `olap` уже выполняются 9 запросов. При поступлении нового запроса он сразу начнёт выполняться параллельно с другими 9-ю запросами. Теперь в пуле будет работать 10 запросов. Если в пул поступит 11-й запрос, он не начнёт выполняться, а будет помещён в очередь ожидания. Когда хотя бы один из 10 выполняющихся запросов завершится, 11-й запрос будет извлечён из очереди и начнёт выполняться.

Если в очереди уже находится запросов, то при отправке 1001-го запроса клиент сразу же получит в ответ ошибку, и этот запрос не будет выполнен. Пример ошибки:

```
Issues:
<main>: Error: Request was rejected, number of local pending requests is 20, number of global delayed/running
requests is 0, sum of them is larger than allowed limit 1 (including concurrent query limit 1) for pool olap
<main>: Error: Query failed during adding/waiting in workload pool olap
```

На число параллельно выполняемых запросов влияет не только `CONCURRENT_QUERY_LIMIT`, но и `DATABASE_LOAD_CPU_THRESHOLD`.

Как работает DATABASE_LOAD_CPU_THRESHOLD

Когда запрос поступает в пул ресурсов, для которого установлен `DATABASE_LOAD_CPU_THRESHOLD`, сразу резервируется 10% от доступного CPU на узле, исходя из предположения, что запрос как минимум потребует такой объём ресурсов. Затем каждые 10 секунд происходит перерасчёт потребляемых ресурсов по всей базе данных, что позволяет уточнить первоначальную 10%-ную оценку. Это означает, что если на узел кластера одновременно поступит более 10 запросов, то не более 10 запросов будут запущены на выполнение, а остальные будут ожидать уточнения фактического потребления CPU.

Как и в случае `CONCURRENT_QUERY_LIMIT`, при превышении указанного порога нагрузки запросы отправляются в очередь ожидания.

Пул ресурсов по умолчанию

Даже если ни одного пула ресурсов не было создано, в системе всегда существует пул ресурсов `default`, который не может быть удалён. Любой запрос, выполняющийся в системе, всегда принадлежит какому-либо пулу — ситуации, когда запрос не

привязан ни к одному пулу ресурсов, не бывает. По умолчанию настройки пула ресурсов `default` выглядят следующим образом:

```
CREATE RESOURCE POOL default WITH (
    CONCURRENT_QUERY_LIMIT=-1,
    QUEUE_SIZE=-1,
    DATABASE_LOAD_CPU_THRESHOLD=-1,
    TOTAL_CPU_LIMIT_PERCENT_PER_NODE=-1
)
```

Это означает, что в пуле ресурсов `default` не применяются никакие ограничения: он функционирует независимо от других пулов и не имеет ограничений на потребляемые ресурсы. В пуле ресурсов `default` можно изменять параметры с помощью запроса ALTER RESOURCE POOL, за исключением параметров `CONCURRENT_QUERY_LIMIT`, `DATABASE_LOAD_CPU_THRESHOLD` и `QUEUE_SIZE`. Это ограничение введено намеренно, чтобы минимизировать риски, связанные с некорректной настройкой пула ресурсов по умолчанию.

Управление ACL пула ресурсов

Права на создание, изменение, удаление пула ресурсов

Для создания, изменения или удаления пула ресурсов необходимо выдать права доступа в соответствии с разрешениями, описанными в справке по CREATE RESOURCE POOL. Например, для создания пулов ресурсов нужно иметь разрешение `CREATE TABLE` на директорию `.metadata/workload_manager/pools`, которое можно выдать запросом следующего вида:

```
GRANT CREATE TABLE ON `.metadata/workload_manager/pools` TO user1;
```

Права на выполнение запроса в пуле ресурсов

Чтобы выполнить запрос в пуле, пользователь должен иметь право доступа `SELECT` на этот пул. Пример выдачи прав:

```
GRANT SELECT
    ON `.metadata/workload_manager/pools/olap`
    TO `user1@domain`;
```

Создание классификатора пула ресурсов

Классификаторы пулов ресурсов позволяют задавать правила, по которым запросы будут распределяться между пулами ресурсов. В примере ниже приведён классификатор пула ресурсов, который отправляет запросы от всех пользователей в пул ресурсов с именем `olap`:

```
CREATE RESOURCE POOL CLASSIFIER olap_classifier
WITH (
    RESOURCE_POOL = 'olap',
    MEMBER_NAME = 'all-users@well-known'
);
```

- `RESOURCE_POOL` — имя пула ресурсов, в который будет отправлен запрос, удовлетворяющий требованиям, заданным в классификаторе пула ресурсов.
- `MEMBER_NAME` — группа пользователей или пользователь, запросы которых будут отправлены в указанный пул ресурсов. Подробнее о формате значения см. CREATE RESOURCE POOL CLASSIFIER.

Управление ACL классификатора пула ресурсов

Классификаторы пула ресурсов глобальны для всей базы данных и применяются ко всем пользователям. Для создания, удаления или изменения классификатора пула ресурсов необходимо иметь право доступа `USE` на всю базу данных, которое можно выдать запросом вида:

```
GRANT USE ON `/my_db` TO user1;
```

Важно

Для использования классификатора у пользователя должен быть доступ к пулу ресурсов, на который этот классификатор ссылается. Если такого доступа нет, классификатор пропускается и проверяется следующий. В будущих версиях это поведение может быть изменено, поэтому не используйте отсутствие прав на пул как механизм управления тем, какой классификатор сработает.

Порядок выбора классификатора пула ресурсов в случае конфликтов

```
CREATE RESOURCE POOL CLASSIFIER olap1_classifier
WITH (
    RESOURCE_POOL = 'olap1',
    MEMBER_NAME = 'user1@domain'
);

CREATE RESOURCE POOL CLASSIFIER olap2_classifier
WITH (
    RESOURCE_POOL = 'olap2',
    MEMBER_NAME = 'user1@domain'
);
```

Допустим, имеются два классификатора пулов ресурсов с конфликтующими условиями, и пользователь `user1@domain` подходит под оба пула ресурсов: `olap1` и `olap2`. Если до этого в системе не существовало ни одного классификатора, то

для `olap1` устанавливается `RANK=1000`, а для `olap2` — `RANK=2000`. Классификаторы пулов ресурсов с меньшим значением `RANK` имеют более высокий приоритет. В данном примере, так как у `olap1` более приоритетный `RANK`, чем у `olap2`, будет выбран именно он.

Также можно самостоятельно задавать `RANK` для классификаторов пулов ресурсов при создании с помощью синтаксической конструкции CREATE RESOURCE POOL CLASSIFIER, либо изменять `RANK` для существующих классификаторов пулов ресурсов с помощью ALTER RESOURCE POOL CLASSIFIER.

В системе не может существовать два классификатора с одинаковым значением `RANK`, что позволяет однозначно определить, какой пул ресурсов будет выбран в случае конфликтующих условий.

Пример приоритетного пула ресурсов

Рассмотрим пример разделения ресурсов между командой аналитиков и условным директором (CEO). Важно, чтобы запросы CEO продолжали выполняться даже тогда, когда аналитические задачи загружают базу данных. Конфигурация для этого сценария может выглядеть следующим образом:

```
CREATE RESOURCE POOL olap WITH (
    CONCURRENT_QUERY_LIMIT=20,
    QUEUE_SIZE=100,
    DATABASE_LOAD_CPU_THRESHOLD=80,
    TOTAL_CPU_LIMIT_PERCENT_PER_NODE=100
);

CREATE RESOURCE POOL the_ceo WITH (
    CONCURRENT_QUERY_LIMIT=20,
    QUEUE_SIZE=100,
    TOTAL_CPU_LIMIT_PERCENT_PER_NODE=100
);
```

В примере выше создаются два пула ресурсов: `olap` для команды аналитиков и `the_ceo` для CEO.

- **Пул ресурсов `olap`**: `DATABASE_LOAD_CPU_THRESHOLD=80` означает, что при достижении загрузки CPU базы данных в 80% новые аналитические запросы перестают запускаться и ожидают в очереди, вмещающей до `QUEUE_SIZE=100` запросов.
- **Пул ресурсов `the_ceo`**: параметр `DATABASE_LOAD_CPU_THRESHOLD` не задан, поэтому запросы CEO продолжают запускаться независимо от степени загрузки базы данных.

Это обеспечивает CEO приоритет при перегрузке: пул аналитики ограничивается первым, а пул CEO — нет.

> **Важно**
>
> Этот механизм управляет только *моментом запуска запросов*. Он не резервирует CPU для одного пула за счёт другого, пока запросы уже выполняются: оба пула выше ограничены значением `TOTAL_CPU_LIMIT_PERCENT_PER_NODE=100`, поэтому выполняющиеся запросы конкурируют за CPU на равных условиях.

Явный выбор пула ресурсов для запроса

При необходимости пользователь может явно указать, в каком пуле следует выполнить заданный запрос. В настоящий момент это можно сделать следующим образом:

- **YDB UI** — в окне настройки запуска запроса `Query execution settings` через параметр `Resource pool`.
- **YDB CLI** — в команде `ydb sql` с параметром `--resource-pool`, например, `ydb sql --resource-pool my_pool -s "SELECT 1"`.
- **YDB CLI (интерактивный режим)** — командой `SET resource_pool = my_pool`, где `my_pool` — наименование пула ресурсов.
- **YDB CPP SDK** — в настройках запуска запроса через параметр ResourcePool.
- **YDB GO SDK** — в настройках запуска запроса `ExecuteOption` через вызов WithResourcePool.
- **YDB Java SDK** — в настройках запуска запроса `ExecuteQuerySettings` через вызов withResourcePool, начиная с версии 2.3.12.
- **YDB Python SDK** — в параметре pool_id методов `QuerySession.execute()`, `QueryTxContext.execute()` и `QuerySessionPool.execute_with_retries()`, начиная с версии 3.31.2.

Диагностика

План запроса

Подробную информацию о планах запросов можно найти на странице структура планов запросов. Для получения информации об используемом пуле ресурсов необходимо запустить команду с получением статистики в формате `json-unicode`. Пример команды:

```
ydb -p <profile_name> sql -s 'select 1' --stats full --format json-unicode
```

В теле плана запроса, полученного с помощью приведённой выше команды, можно найти полезные атрибуты для диагностики работы с пулом ресурсов. Пример такой информации:

```
"Node Type" : "Query",
"Stats" : {
  "TotalDurationUs": 28795,
  "ProcessCpuTimeUs": 45,
  "Compilation": {
    "FromCache": false,
    "CpuTimeUs": 7280,
    "DurationUs": 21700
  },
```

```
  "ResourcePoolId": "default",
  "QueuedTimeUs": 0
},
"PlanNodeType" : "Query"
```

Полезные атрибуты:

- `TotalDurationUs` — общее время выполнения запроса, включая время ожидания в очереди.
- `ResourcePoolId` — имя пула ресурсов, к которому был привязан запрос.
- `QueuedTimeUs` — общее время ожидания запроса в очереди.

Статус выполняющегося запроса

Информацию о том, как обрабатывается запрос в Workload Manager, можно получить на основе системного представления `.sys/query_sessions`. В этом представлении есть следующие поля:

- `WmPoolId` `(Utf8)` - Идентификатор пула, в котором выполняется запрос.
- `WmState` `(Utf8)` - Статус запроса в WM.
- `WmEnterTime` `(Timestamp)` - Время, когда запрос перешел в статус PENDING или DELAYED.
- `WmExitTime` `(Timestamp)` - Время, когда запрос передан на выполнение.

Возможные значения поля WmState:

- `NONE` - Не обрабатывается.
- `PENDING` - Обрабатывается (в процессе классификации/маршрутизации).
- `DELAYED` - В очереди.
- `EXITED` - Передан на исполнение.

Следующий запрос выводит информацию обо всех активных запросах в системе:

```
select
    Query,          -- Запрос
    WmPoolId,       -- Идентификатор пула
    WmState,        -- Статус запроса в WM
    WmEnterTime,    -- Время, когда запрос перешел в статус PENDING или DELAYED
    WmExitTime      -- Время, когда запрос передан на выполнение
from `.sys/query_sessions`
where State = 'EXECUTING'
```

Метрики

Информацию о метриках пулов ресурсов можно найти в справке по метрикам.

Системные представления

Информацию о системных представлениях, связанных с пулами ресурсов и классификаторами пулов ресурсов, можно найти на странице Информация о пулах ресурсов.

См. также

- CREATE RESOURCE POOL
- ALTER RESOURCE POOL
- DROP RESOURCE POOL
- CREATE RESOURCE POOL CLASSIFIER
- ALTER RESOURCE POOL CLASSIFIER
- DROP RESOURCE POOL CLASSIFIER

Для аналитиков

В этом разделе собраны примеры и рекомендации по работе с аналитическими (OLAP) сценариями в YDB.

Данный раздел содержит следующие материалы

- Импорт датасетов

Связанные разделы

- Вопросы и ответы об аналитике в YDB
- Колоночные таблицы
- Агрегатные функции

Quickstart: Загрузка данных TPC-H из Object Storage и выполнение аналитического запроса

В этом quickstart описывается процесс загрузки тестового набора данных TPC-H (scale factor 10, 10 ГБ) в YDB из внешнего S3-совместимого хранилища и выполнение аналитического запроса.

После выполнения шагов в базе данных будут созданы:

- Внешний источник данных `EXTERNAL DATA SOURCE`.
- Внешние таблицы `EXTERNAL TABLE` для схем Parquet-файлов.
- Колоночные таблицы YDB, оптимизированные для аналитических запросов.

Действия включают:

- Загрузку данных из Object Storage в таблицы YDB с помощью `UPSERT INTO... SELECT FROM`.
- Выполнение тестового запроса (TPC-H Q0) для проверки.

Предварительные требования

- Наличие созданной базы данных YDB.
- Наличие инструмента для выполнения YQL/SQL-запросов (YDB UI или CLI).

Создание внешнего источника данных

В данном этапе будет создан объект `EXTERNAL DATA SOURCE`, который является именованной ссылкой на внешнее хранилище данных. Он содержит в себе информацию о местоположении данных и способе доступа к ним.

`EXTERNAL DATA SOURCE` — это сущность в YDB, которая описывает подключение к внешнему хранилищу, например, к S3-совместимому Object Storage.

```
CREATE EXTERNAL DATA SOURCE IF NOT EXISTS `external/tpc` WITH (
    SOURCE_TYPE="ObjectStorage",
    LOCATION="https://storage.yandexcloud.net/tpc/",
    AUTH_METHOD="NONE"
);
```

Параметры:

- `SOURCE_TYPE="ObjectStorage"`: тип хранилища (S3-совместимое).
- `LOCATION="https://storage.yandexcloud.net/tpc/"`: URL бакета.
- `AUTH_METHOD="NONE"`: публичный доступ без аутентификации.

Создание внешних таблиц

На этом шаге определяются схемы (колонки и типы) для Parquet-файлов во внешнем источнике. Это позволяет обращаться к данным через SQL-запросы в YDB, связывая логическую структуру с физическими файлами.

`EXTERNAL TABLE` — это "виртуальная" таблица, которая описывает структуру данных, находящихся во внешнем источнике. Это связывает логическую схему таблицы с физическими файлами.

```
CREATE EXTERNAL TABLE IF NOT EXISTS `external/tpch/s10/customer` (
    c_acctbal Double NOT NULL,
    c_mktsegment String NOT NULL,
    c_phone String NOT NULL,
    c_nationkey Int32 NOT NULL,
    c_custkey Int32 NOT NULL,
    c_name String NOT NULL,
    c_comment String NOT NULL,
    c_address String NOT NULL
) WITH (
    DATA_SOURCE="external/tpc",
    LOCATION="/h/s10/parquet/customer/",
    FORMAT="parquet"
);
```

Параметры:

- `DATA_SOURCE="external/tpc"`: ссылка на источник.
- `LOCATION="/h/s10/parquet/customer/"`: путь к файлам.
- `FORMAT="parquet"`: формат данных.

Создайте остальные таблицы

```
CREATE EXTERNAL TABLE IF NOT EXISTS `external/tpch/s10/customer` (
    c_acctbal Double NOT NULL,
    c_mktsegment String NOT NULL,
    c_phone String NOT NULL,
    c_nationkey Int32 NOT NULL,
    c_custkey Int32 NOT NULL,
    c_name String NOT NULL,
    c_comment String NOT NULL,
    c_address String NOT NULL
) WITH (
    DATA_SOURCE="external/tpc",
    LOCATION="/h/s10/parquet/customer/",
    FORMAT="parquet"
);
```

```
CREATE EXTERNAL TABLE IF NOT EXISTS `external/tpch/s10/lineitem` (
    l_orderkey Int64 NOT NULL,
    l_partkey Int32 NOT NULL,
    l_suppkey Int32 NOT NULL,
    l_linenumber Int32 NOT NULL,
    l_quantity Double NOT NULL,
    l_extendedprice Double NOT NULL,
    l_discount Double NOT NULL,
    l_tax Double NOT NULL,
    l_returnflag String NOT NULL,
    l_linestatus String NOT NULL,
    l_shipdate Date NOT NULL,
    l_commitdate Date NOT NULL,
    l_receiptdate Date NOT NULL,
    l_shipinstruct String NOT NULL,
    l_shipmode String NOT NULL,
    l_comment String NOT NULL
) WITH (
    DATA_SOURCE="external/tpc",
    LOCATION="/h/s10/parquet/lineitem/",
    FORMAT="parquet"
);

CREATE EXTERNAL TABLE IF NOT EXISTS `external/tpch/s10/nation` (
    n_nationkey Int32 NOT NULL,
    n_name String NOT NULL,
    n_regionkey Int32 NOT NULL,
    n_comment String NOT NULL
) WITH (
    DATA_SOURCE="external/tpc",
    LOCATION="/h/s10/parquet/nation/",
    FORMAT="parquet"
);

CREATE EXTERNAL TABLE IF NOT EXISTS `external/tpch/s10/orders` (
    o_orderkey Int64 NOT NULL,
    o_custkey Int32 NOT NULL,
    o_orderstatus String NOT NULL,
    o_totalprice Double NOT NULL,
    o_orderdate Date NOT NULL,
    o_orderpriority String NOT NULL,
    o_clerk String NOT NULL,
    o_shippriority Int32 NOT NULL,
    o_comment String NOT NULL
) WITH (
    DATA_SOURCE="external/tpc",
    LOCATION="/h/s10/parquet/orders/",
    FORMAT="parquet"
);

CREATE EXTERNAL TABLE IF NOT EXISTS `external/tpch/s10/part` (
    p_partkey Int32 NOT NULL,
    p_name String NOT NULL,
    p_mfgr String NOT NULL,
    p_brand String NOT NULL,
    p_type String NOT NULL,
    p_size Int32 NOT NULL,
    p_container String NOT NULL,
    p_retailprice Double NOT NULL,
    p_comment String NOT NULL
) WITH (
    DATA_SOURCE="external/tpc",
    LOCATION="/h/s10/parquet/part/",
    FORMAT="parquet"
);

CREATE EXTERNAL TABLE IF NOT EXISTS `external/tpch/s10/partsupp` (
    ps_partkey Int32 NOT NULL,
    ps_suppkey Int32 NOT NULL,
    ps_availqty Int32 NOT NULL,
    ps_supplycost Double NOT NULL,
    ps_comment String NOT NULL
) WITH (
    DATA_SOURCE="external/tpc",
    LOCATION="/h/s10/parquet/partsupp/",
    FORMAT="parquet"
);

CREATE EXTERNAL TABLE IF NOT EXISTS `external/tpch/s10/region` (
    r_regionkey Int32 NOT NULL,
    r_name String NOT NULL,
    r_comment String NOT NULL
) WITH (
    DATA_SOURCE="external/tpc",
    LOCATION="/h/s10/parquet/region/",
    FORMAT="parquet"
);

CREATE EXTERNAL TABLE IF NOT EXISTS `external/tpch/s10/supplier` (
    s_suppkey Int32 NOT NULL,
    s_name String NOT NULL,
```

```
    s_address String NOT NULL,
    s_nationkey Int32 NOT NULL,
    s_phone String NOT NULL,
    s_acctbal Double NOT NULL,
    s_comment String NOT NULL
) WITH (
    DATA_SOURCE="external/tpc",
    LOCATION="/h/s10/parquet/supplier/",
    FORMAT="parquet"
);
```

Создание колоночных таблиц

В данном этапе создаются внутренние таблицы YDB для хранения данных. Они используют колоночный формат и партиционирование, что обеспечивает высокую производительность для аналитических запросов за счет параллельной обработки.

```
CREATE TABLE IF NOT EXISTS `tpch/s10/customer` (
    c_acctbal Double NOT NULL,
    c_mktsegment String NOT NULL,
    c_phone String NOT NULL,
    c_nationkey Int32 NOT NULL,
    c_custkey Int32 NOT NULL,
    c_name String NOT NULL,
    c_comment String NOT NULL,
    c_address String NOT NULL,
    PRIMARY KEY (c_custkey)
)
PARTITION BY HASH (c_custkey)
WITH (
    STORE = COLUMN,
    PARTITION_COUNT = 64
);
```

Параметры:

- `PRIMARY KEY (c_custkey)`: первичный ключ таблицы.
- `PARTITION BY HASH (c_custkey)`: хеш-партиционирование для параллелизма.
- `STORE = COLUMN`: колоночный формат хранения данных для ускорения аналитических запросов.
- `PARTITION_COUNT = 64`: количество партиций для распределения данных.

Создайте остальные таблицы

```
CREATE TABLE IF NOT EXISTS `tpch/s10/customer` (
    c_acctbal Double NOT NULL,
    c_mktsegment String NOT NULL,
    c_phone String NOT NULL,
    c_nationkey Int32 NOT NULL,
    c_custkey Int32 NOT NULL,
    c_name String NOT NULL,
    c_comment String NOT NULL,
    c_address String NOT NULL,
    PRIMARY KEY (c_custkey)
)
PARTITION BY HASH (c_custkey)
WITH (
    STORE = COLUMN,
    PARTITION_COUNT = 64
);

CREATE TABLE IF NOT EXISTS `tpch/s10/lineitem` (
    l_orderkey Int64 NOT NULL,
    l_partkey Int32 NOT NULL,
    l_suppkey Int32 NOT NULL,
    l_linenumber Int32 NOT NULL,
    l_quantity Double NOT NULL,
    l_extendedprice Double NOT NULL,
    l_discount Double NOT NULL,
    l_tax Double NOT NULL,
    l_returnflag String NOT NULL,
    l_linestatus String NOT NULL,
    l_shipdate Date NOT NULL,
    l_commitdate Date NOT NULL,
    l_receiptdate Date NOT NULL,
    l_shipinstruct String NOT NULL,
    l_shipmode String NOT NULL,
    l_comment String NOT NULL,
    PRIMARY KEY (l_orderkey, l_linenumber)
)
PARTITION BY HASH (l_orderkey)
WITH (
    STORE = COLUMN,
    PARTITION_COUNT = 64
);

CREATE TABLE IF NOT EXISTS `tpch/s10/nation` (
    n_nationkey Int32 NOT NULL,
    n_name String NOT NULL,
```

```
    n_regionkey Int32 NOT NULL,
    n_comment String NOT NULL,
    PRIMARY KEY (n_nationkey)
)
PARTITION BY HASH (n_nationkey)
WITH (
    STORE = COLUMN,
    PARTITION_COUNT = 64
);

CREATE TABLE IF NOT EXISTS `tpch/s10/orders` (
    o_orderkey Int64 NOT NULL,
    o_custkey Int32 NOT NULL,
    o_orderstatus String NOT NULL,
    o_totalprice Double NOT NULL,
    o_orderdate Date NOT NULL,
    o_orderpriority String NOT NULL,
    o_clerk String NOT NULL,
    o_shippriority Int32 NOT NULL,
    o_comment String NOT NULL,
    PRIMARY KEY (o_orderkey)
)
PARTITION BY HASH (o_orderkey)
WITH (
    STORE = COLUMN,
    PARTITION_COUNT = 64
);

CREATE TABLE IF NOT EXISTS `tpch/s10/part` (
    p_partkey Int32 NOT NULL,
    p_name String NOT NULL,
    p_mfgr String NOT NULL,
    p_brand String NOT NULL,
    p_type String NOT NULL,
    p_size Int32 NOT NULL,
    p_container String NOT NULL,
    p_retailprice Double NOT NULL,
    p_comment String NOT NULL,
    PRIMARY KEY (p_partkey)
)
PARTITION BY HASH (p_partkey)
WITH (
    STORE = COLUMN,
    PARTITION_COUNT = 64
);

CREATE TABLE IF NOT EXISTS `tpch/s10/partsupp` (
    ps_partkey Int32 NOT NULL,
    ps_suppkey Int32 NOT NULL,
    ps_availqty Int32 NOT NULL,
    ps_supplycost Double NOT NULL,
    ps_comment String NOT NULL,
    PRIMARY KEY (ps_partkey, ps_suppkey)
)
PARTITION BY HASH (ps_partkey, ps_suppkey)
WITH (
    STORE = COLUMN,
    PARTITION_COUNT = 64
);

CREATE TABLE IF NOT EXISTS `tpch/s10/region` (
    r_regionkey Int32 NOT NULL,
    r_name String NOT NULL,
    r_comment String NOT NULL,
    PRIMARY KEY (r_regionkey)
)
PARTITION BY HASH (r_regionkey)
WITH (
    STORE = COLUMN,
    PARTITION_COUNT = 64
);

CREATE TABLE IF NOT EXISTS `tpch/s10/supplier` (
    s_suppkey Int32 NOT NULL,
    s_name String NOT NULL,
    s_address String NOT NULL,
    s_nationkey Int32 NOT NULL,
    s_phone String NOT NULL,
    s_acctbal Double NOT NULL,
    s_comment String NOT NULL,
    PRIMARY KEY (s_suppkey)
)
PARTITION BY HASH (s_suppkey)
WITH (
    STORE = COLUMN,
    PARTITION_COUNT = 64
);
```

Загрузка данных в колоночные таблицы

На данном этапе выполняется копирование данных из внешнего хранилища в внутренние таблицы YDB. Это обеспечивает локальное хранение для быстрой аналитики. Процесс может занять время (для 10 ГБ — порядка минут, в зависимости от ресурсов).

```
UPSERT INTO `tpch/s10/customer` SELECT * FROM `external/tpch/s10/customer`;
UPSERT INTO `tpch/s10/lineitem` SELECT * FROM `external/tpch/s10/lineitem`;
UPSERT INTO `tpch/s10/nation`   SELECT * FROM `external/tpch/s10/nation`;
UPSERT INTO `tpch/s10/orders`   SELECT * FROM `external/tpch/s10/orders`;
UPSERT INTO `tpch/s10/part`     SELECT * FROM `external/tpch/s10/part`;
UPSERT INTO `tpch/s10/partsupp` SELECT * FROM `external/tpch/s10/partsupp`;
UPSERT INTO `tpch/s10/region`   SELECT * FROM `external/tpch/s10/region`;
UPSERT INTO `tpch/s10/supplier` SELECT * FROM `external/tpch/s10/supplier`;
```

Выполнение тестового аналитического запроса

На данном этапе выполняется тестовый запрос TPC-H Q0 для проверки корректности загрузки и производительности. Запрос агрегирует метрики по таблице `lineitem` (около 6 млн строк), с группировкой и последующей сортировкой данных, демонстрируя эффективность колоночных таблиц.

```
SELECT
    l_returnflag,
    l_linestatus,
    SUM(l_quantity) AS sum_qty,
    SUM(l_extendedprice) AS sum_base_price,
    SUM(l_extendedprice * (1 - l_discount)) AS sum_disc_price,
    SUM(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
    AVG(l_quantity) AS avg_qty,
    AVG(l_extendedprice) AS avg_price,
    AVG(l_discount) AS avg_disc,
    COUNT(*) AS count_order
FROM
    `tpch/s10/lineitem`
WHERE
    l_shipdate <= Date('1998-12-01') - Interval("P90D")
GROUP BY
    l_returnflag,
    l_linestatus
ORDER BY
    l_returnflag,
    l_linestatus;
```

Дальнейшие шаги

- Выполнение других запросов из набора TPC-H к загруженным таблицам с помощью YDB CLI и тестовой нагрузки TPC-H.
- Интеграция с BI-системами для визуализации и анализа данных.

Обзор

На этих страницах описаны популярные датасеты, которые вы можете загрузить в YDB для ознакомления с функциональностью базы данных и тестирования различных сценариев использования.

Предварительные требования

Для загрузки датасетов вам потребуется:

1. Установленный YDB CLI
2. [Опционально] Настроенный профиль подключения к YDB, чтобы не указывать параметры подключения при каждом вызове

Общая информация о загрузке данных

YDB поддерживает импорт данных из CSV-файлов с помощью команды `ydb import file csv`. Пример запуска команды:

```
ydb import file csv --header --null-value "" --path <путь_к_таблице> <файл>.csv
```

Где:

- `--header` означает, что в первой строке файла содержится список имён колонок, а сами данные начинаются со второй строки;
- `--null-value ""` означает, что пустая строка вместо значения в CSV будет интерпретироваться как null-значение при импорте данных из csv-файла в таблицу.

Для импорта данных нужна заранее созданная таблица в YDB. Основной способ создания таблицы - выполнить YQL-запрос `CREATE TABLE`. Чтобы не составлять его полностью вручную, можно попробовать выполнить команду импорта из файла, как в любом примере ниже, не создавая перед этим таблицу. В таком случае CLI предложит текст `CREATE TABLE`, который можно будет взять за основу, при необходимости отредактировать и выполнить.

Для импорта данных в YDB таблица должна быть заранее создана. Обычно таблицу создают с помощью YQL-запроса `CREATE TABLE`. Однако, вместо того чтобы писать такой запрос вручную, можно запустить команду импорта `ydb import file csv` так же, как показано в примерах импорта в этом разделе. Если таблица отсутствует, CLI автоматически предложит готовый текст запроса `CREATE TABLE`, который можно будет использовать для создания таблицы.

> **Выбор первичного ключа**
>
> YDB требует, чтобы у таблицы был первичный ключ. Он значительно ускоряет загрузку и обработку данных, а также позволяет выполнять дедупликацию: строки с одинаковыми значениями в столбцах первичного ключа заменяются друг другом.
>
> Если в импортируемом наборе данных нет подходящих столбцов для первичного ключа, мы добавляем новый столбец с номерами строк и используем его в качестве первичного ключа, так как номер каждой строки уникален в пределах файла.

Особенности и ограничения

При работе с загрузкой CSV-файлов в YDB следует учитывать следующие моменты:

1. **Имена колонок**: Названия колонок не должны содержать пробелы или специальные символы.
2. **Типы данных**:
   - Строки в формате даты/времени с указанием временной зоны (например, "2019-11-01 00:00:00 UTC") будут импортированы как тип Text
   - Тип Bool не поддерживается в качестве типа колонки, используйте Text или Int64

Доступные датасеты

- Chess Position Evaluations - оценки шахматных позиций от движка Stockfish
- Video Game Sales - данные о продажах видеоигр
- E-Commerce Behavior Data - данные о поведении пользователей в интернет-магазине
- COVID-19 Open Research Dataset - открытый набор данных исследований COVID-19
- Netflix Movies and TV Shows - данные о фильмах и сериалах на Netflix
- Animal Crossing New Horizons Catalog - каталог предметов из игры

Обзор

На этих страницах описаны популярные датасеты, которые вы можете загрузить в YDB для ознакомления с функциональностью базы данных и тестирования различных сценариев использования.

Предварительные требования

Для загрузки датасетов вам потребуется:

1. Установленный YDB CLI
2. [Опционально] Настроенный профиль подключения к YDB, чтобы не указывать параметры подключения при каждом вызове

Общая информация о загрузке данных

YDB поддерживает импорт данных из CSV-файлов с помощью команды `ydb import file csv`. Пример запуска команды:

```
ydb import file csv --header --null-value "" --path <путь_к_таблице> <файл>.csv
```

Где:

- `--header` означает, что в первой строке файла содержится список имён колонок, а сами данные начинаются со второй строки;
- `--null-value ""` означает, что пустая строка вместо значения в CSV будет интерпретироваться как null-значение при импорте данных из csv-файла в таблицу.

Для импорта данных нужна заранее созданная таблица в YDB. Основной способ создания таблицы - выполнить YQL-запрос `CREATE TABLE`. Чтобы не составлять его полностью вручную, можно попробовать выполнить команду импорта из файла, как в любом примере ниже, не создавая перед этим таблицу. В таком случае CLI предложит текст `CREATE TABLE`, который можно будет взять за основу, при необходимости отредактировать и выполнить.

Для импорта данных в YDB таблица должна быть заранее создана. Обычно таблицу создают с помощью YQL-запроса `CREATE TABLE`. Однако, вместо того чтобы писать такой запрос вручную, можно запустить команду импорта `ydb import file csv` так же, как показано в примерах импорта в этом разделе. Если таблица отсутствует, CLI автоматически предложит готовый текст запроса `CREATE TABLE`, который можно будет использовать для создания таблицы.

> **Выбор первичного ключа**
>
> YDB требует, чтобы у таблицы был первичный ключ. Он значительно ускоряет загрузку и обработку данных, а также позволяет выполнять дедупликацию: строки с одинаковыми значениями в столбцах первичного ключа заменяются друг другом.
>
> Если в импортируемом наборе данных нет подходящих столбцов для первичного ключа, мы добавляем новый столбец с номерами строк и используем его в качестве первичного ключа, так как номер каждой строки уникален в пределах файла.

Особенности и ограничения

При работе с загрузкой CSV-файлов в YDB следует учитывать следующие моменты:

1. **Имена колонок**: Названия колонок не должны содержать пробелы или специальные символы.
2. **Типы данных**:
   - Строки в формате даты/времени с указанием временной зоны (например, "2019-11-01 00:00:00 UTC") будут импортированы как тип Text
   - Тип Bool не поддерживается в качестве типа колонки, используйте Text или Int64

Доступные датасеты

- Chess Position Evaluations - оценки шахматных позиций от движка Stockfish
- Video Game Sales - данные о продажах видеоигр
- E-Commerce Behavior Data - данные о поведении пользователей в интернет-магазине
- COVID-19 Open Research Dataset - открытый набор данных исследований COVID-19
- Netflix Movies and TV Shows - данные о фильмах и сериалах на Netflix
- Animal Crossing New Horizons Catalog - каталог предметов из игры

Chess Position Evaluations

> **Примечание**
>
> Эта страница является частью раздела Импорт датасетов, где описаны примеры загрузки популярных наборов данных в YDB. Перед началом работы ознакомьтесь с общей информацией о требованиях и процессе импорта.

Датасет включает 513 миллионов оценок шахматных позиций, выполненных движком Stockfish для анализа на платформе Lichess.

Источник: Kaggle - Chess Position Evaluations

Размер: 59.66 GB

Пример загрузки

1. Скачайте файл `evals.csv` с Kaggle
2. Создайте таблицу в YDB одним из следующих способов:

YDB UI

Подробнее про YDB UI.

```
CREATE TABLE `evals` (
    `fen` Text NOT NULL,
    `line` Text NOT NULL,
    `depth` Uint64,
    `knodes` Uint64,
    `cp` Double,
    `mate` Double,
    PRIMARY KEY (`fen`, `line`)
)
WITH (
    STORE = COLUMN,
    UNIFORM_PARTITIONS = 50
);
```

YDB CLI

```
ydb sql -s \
'CREATE TABLE `evals` (
    `fen` Text NOT NULL,
    `line` Text NOT NULL,
    `depth` Uint64,
    `knodes` Uint64,
    `cp` Double,
    `mate` Double,
    PRIMARY KEY (`fen`, `line`)
)
WITH (
    STORE = COLUMN,
    UNIFORM_PARTITIONS = 50
);'
```

3. Выполните команду импорта:

```
ydb import file csv --header --null-value "" --path evals evals.csv
```

Пример аналитического запроса

Определим позиции с наибольшим количеством ходов, проанализированных движком Stockfish:

YDB UI

```
SELECT
    fen,
    MAX(depth) AS max_depth,
    SUM(knodes) AS total_knodes
FROM evals
GROUP BY fen
ORDER BY max_depth DESC
LIMIT 10;
```

YDB CLI

```
ydb sql -s \
'SELECT
    fen,
    MAX(depth) AS max_depth,
    SUM(knodes) AS total_knodes
FROM evals
GROUP BY fen
ORDER BY max_depth DESC
LIMIT 10;'
```

Этот запрос выполняет следующие действия:

- Находит позиции (представленные в формате FEN) с максимальной глубиной анализа (depth).
- Суммирует количество проанализированных узлов (knodes) для каждой позиции.
- Сортирует результаты по максимальной глубине анализа в порядке убывания.
- Выводит топ-10 позиций с наибольшей глубиной анализа.

Video Game Sales

> **Примечание**
>
> Эта страница является частью раздела Импорт датасетов, где описаны примеры загрузки популярных наборов данных в YDB. Перед началом работы ознакомьтесь с общей информацией о требованиях и процессе импорта.

Данные о продажах видеоигр.

Источник: Kaggle - Video Game Sales

Размер: 1.36 MB

Пример загрузки

1. Скачайте и разархивируйте файл `vgsales.csv` с Kaggle
2. Создайте таблицу в YDB одним из следующих способов:

YDB UI

Подробнее про YDB UI.

```
CREATE TABLE `vgsales` (
    `Rank` Uint64 NOT NULL,
    `Name` Text NOT NULL,
    `Platform` Text NOT NULL,
    `Year` Text NOT NULL,
    `Genre` Text NOT NULL,
    `Publisher` Text NOT NULL,
    `NA_Sales` Double NOT NULL,
    `EU_Sales` Double NOT NULL,
    `JP_Sales` Double NOT NULL,
    `Other_Sales` Double NOT NULL,
    `Global_Sales` Double NOT NULL,
    PRIMARY KEY (`Rank`)
)
WITH (
    STORE = COLUMN
);
```

YDB CLI

```
ydb sql -s \
'CREATE TABLE `vgsales` (
    `Rank` Uint64 NOT NULL,
    `Name` Text NOT NULL,
    `Platform` Text NOT NULL,
    `Year` Text NOT NULL,
    `Genre` Text NOT NULL,
    `Publisher` Text NOT NULL,
    `NA_Sales` Double NOT NULL,
    `EU_Sales` Double NOT NULL,
    `JP_Sales` Double NOT NULL,
    `Other_Sales` Double NOT NULL,
    `Global_Sales` Double NOT NULL,
    PRIMARY KEY (`Rank`)
)
WITH (
    STORE = COLUMN
);'
```

3. Выполните команду импорта:

```
ydb import file csv --header --null-value "" --path vgsales vgsales.csv
```

Пример аналитического запроса

Чтобы определить издателя, у которого наибольшая средняя продажа игр в Северной Америке, выполните запрос:

YDB UI

```
SELECT
    Publisher,
    AVG(NA_Sales) AS average_na_sales
FROM vgsales
GROUP BY Publisher
ORDER BY average_na_sales DESC
LIMIT 1;
```

YDB CLI

```
ydb sql -s \
'SELECT
    Publisher,
    AVG(NA_Sales) AS average_na_sales
```

```
FROM vgsales
GROUP BY Publisher
ORDER BY average_na_sales DESC
LIMIT 1;'
```

Результат:

```
┌─────────────┬──────────────────┐
│ Publisher │ average_na_sales │
├─────────────┼──────────────────┤
│ "Palcom"  │ 3.38             │
└─────────────┴──────────────────┘
```

Запрос позволит найти, какой издатель достиг наибольшего успеха в Северной Америке по средней продаже.

E-Commerce Behavior Data

> **Примечание**
>
> Эта страница является частью раздела Импорт датасетов, где описаны примеры загрузки популярных наборов данных в YDB. Перед началом работы ознакомьтесь с общей информацией о требованиях и процессе импорта.

Данные о поведении пользователей в мультикатегорийном интернет-магазине.

Источник: Kaggle - E-commerce behavior data

Размер: 9 GB

Пример загрузки

1. Скачайте и разархивируйте файл `2019-Nov.csv` с Kaggle
2. Датасет включает в себя полностью идентичные строки. Поскольку YDB требует указания уникальных значений первичного ключа, добавим в файл новую колонку под названием `row_id`, где значение ключа будет равно номеру строки в исходном файле. Это позволит предотвратить удаление повторяющихся данных. Эту операцию можно осуществить с помощью команды awk:

```
awk 'NR==1 {print "row_id," $0; next} {print NR-1 "," $0}' 2019-Nov.csv > temp.csv && mv temp.csv 2019-Nov.cs
v
```

3. Создайте таблицу в YDB одним из следующих способов:

YDB UI

Подробнее про YDB UI.

```
CREATE TABLE `ecommerce_table` (
    `row_id` Uint64 NOT NULL,
    `event_time` Text NOT NULL,
    `event_type` Text NOT NULL,
    `product_id` Uint64 NOT NULL,
    `category_id` Uint64,
    `category_code` Text,
    `brand` Text,
    `price` Double NOT NULL,
    `user_id` Uint64 NOT NULL,
    `user_session` Text NOT NULL,
    PRIMARY KEY (`row_id`)
)
WITH (
    STORE = COLUMN,
    UNIFORM_PARTITIONS = 50
);
```

YDB CLI

```
ydb sql -s \
'CREATE TABLE `ecommerce_table` (
    `row_id` Uint64 NOT NULL,
    `event_time` Text NOT NULL,
    `event_type` Text NOT NULL,
    `product_id` Uint64 NOT NULL,
    `category_id` Uint64,
    `category_code` Text,
    `brand` Text,
    `price` Double NOT NULL,
    `user_id` Uint64 NOT NULL,
    `user_session` Text NOT NULL,
    PRIMARY KEY (`row_id`)
)
WITH (
    STORE = COLUMN,
    UNIFORM_PARTITIONS = 50
);'
```

4. Выполните команду импорта:

```
ydb import file csv --header --null-value "" --path ecommerce_table 2019-Nov.csv
```

Пример аналитического запроса

Определим самые популярные категории продуктов 1 ноября 2019 года:

YDB UI

```
SELECT
    category_code,
    COUNT(*) AS view_count
FROM ecommerce_table
WHERE
    SUBSTRING(CAST(event_time AS String), 0, 10) = '2019-11-01'
```

```
    AND event_type = 'view'
GROUP BY category_code
ORDER BY view_count DESC
LIMIT 10;
```

YDB CLI

```
ydb sql -s \
'SELECT
    category_code,
    COUNT(*) AS view_count
FROM ecommerce_table
WHERE
    SUBSTRING(CAST(event_time AS String), 0, 10) = "2019-11-01"
    AND event_type = "view"
GROUP BY category_code
ORDER BY view_count DESC
LIMIT 10;'
```

Результат:

```
┌──────────────────────────────────────┬────────────┐
| category_code                        | view_count |
├──────────────────────────────────────┼────────────┤
| null                                 | 453024     |
├──────────────────────────────────────┼────────────┤
| "electronics.smartphone"             | 360650     |
├──────────────────────────────────────┼────────────┤
| "electronics.clocks"                 | 43581      |
├──────────────────────────────────────┼────────────┤
| "computers.notebook"                 | 40878      |
├──────────────────────────────────────┼────────────┤
| "electronics.video.tv"               | 40383      |
├──────────────────────────────────────┼────────────┤
| "electronics.audio.headphone"        | 37489      |
├──────────────────────────────────────┼────────────┤
| "apparel.shoes"                      | 31013      |
├──────────────────────────────────────┼────────────┤
| "appliances.kitchen.washer"          | 28028      |
├──────────────────────────────────────┼────────────┤
| "appliances.kitchen.refrigerators"   | 27808      |
├──────────────────────────────────────┼────────────┤
| "appliances.environment.vacuum"      | 26477      |
└──────────────────────────────────────┴────────────┘
```

COVID-19 Open Research Dataset

> **Примечание**
>
> Эта страница является частью раздела Импорт датасетов, где описаны примеры загрузки популярных наборов данных в YDB. Перед началом работы ознакомьтесь с общей информацией о требованиях и процессе импорта.

Открытый набор данных исследований COVID-19.

Источник: Kaggle - COVID-19 Open Research Dataset Challenge

Размер: 1.65 GB (файл metadata.csv)

Пример загрузки

1. Скачайте и разархивируйте файл `metadata.csv` с Kaggle
2. Датасет включает в себя полностью идентичные строки. Поскольку YDB требует указания уникальных значений первичного ключа, добавим в файл новую колонку под названием `row_id`, где значение ключа будет равно номеру строки в исходном файле. Это позволит предотвратить удаление повторяющихся данных. Эту операцию можно осуществить с помощью команды awk:

```
awk 'NR==1 {print "row_id," $0; next} {print NR-1 "," $0}' metadata.csv > temp.csv && mv temp.csv metadata.cs
v
```

3. Создайте таблицу в YDB одним из следующих способов:

YDB UI

Подробнее про YDB UI.

```
CREATE TABLE `covid_research` (
    `row_id` Uint64 NOT NULL,
    `cord_uid` Text NOT NULL,
    `sha` Text NOT NULL,
    `source_x` Text NOT NULL,
    `title` Text NOT NULL,
    `doi` Text NOT NULL,
    `pmcid` Text NOT NULL,
    `pubmed_id` Text NOT NULL,
    `license` Text NOT NULL,
    `abstract` Text NOT NULL,
    `publish_time` Text NOT NULL,
    `authors` Text NOT NULL,
    `journal` Text NOT NULL,
    `mag_id` Text,
    `who_covidence_id` Text,
    `arxiv_id` Text,
    `pdf_json_files` Text NOT NULL,
    `pmc_json_files` Text NOT NULL,
    `url` Text NOT NULL,
    `s2_id` Uint64,
    PRIMARY KEY (`row_id`)
)
WITH (
    STORE = COLUMN
);
```

YDB CLI

```
ydb sql -s \
'CREATE TABLE `covid_research` (
    `row_id` Uint64 NOT NULL,
    `cord_uid` Text NOT NULL,
    `sha` Text NOT NULL,
    `source_x` Text NOT NULL,
    `title` Text NOT NULL,
    `doi` Text NOT NULL,
    `pmcid` Text NOT NULL,
    `pubmed_id` Text NOT NULL,
    `license` Text NOT NULL,
    `abstract` Text NOT NULL,
    `publish_time` Text NOT NULL,
    `authors` Text NOT NULL,
    `journal` Text NOT NULL,
    `mag_id` Text,
    `who_covidence_id` Text,
    `arxiv_id` Text,
    `pdf_json_files` Text NOT NULL,
    `pmc_json_files` Text NOT NULL,
    `url` Text NOT NULL,
    `s2_id` Uint64,
    PRIMARY KEY (`row_id`)
)
WITH (
    STORE = COLUMN
);'
```

4. Выполните команду импорта:

```
ydb import file csv --header --null-value "" --path covid_research metadata.csv
```

Пример аналитического запроса

Выполните запрос для определения журналов с наибольшим количеством публикаций:

YDB UI

```
SELECT
    journal,
    COUNT(*) AS publication_count
FROM covid_research
WHERE journal IS NOT NULL AND journal != ''
GROUP BY journal
ORDER BY publication_count DESC
LIMIT 10;
```

YDB CLI

```
ydb sql -s \
'SELECT
    journal,
    COUNT(*) AS publication_count
FROM covid_research
WHERE journal IS NOT NULL AND journal != ""
GROUP BY journal
ORDER BY publication_count DESC
LIMIT 10;'
```

Результат:

```
┌──────────────────────────────────┬──────────────────┐
| journal                          | publication_count |
├──────────────────────────────────┼──────────────────┤
| "PLoS One"                       | 9953             |
├──────────────────────────────────┼──────────────────┤
| "bioRxiv"                        | 8961             |
├──────────────────────────────────┼──────────────────┤
| "Int J Environ Res Public Health" | 8201             |
├──────────────────────────────────┼──────────────────┤
| "BMJ"                            | 6928             |
├──────────────────────────────────┼──────────────────┤
| "Sci Rep"                        | 5935             |
├──────────────────────────────────┼──────────────────┤
| "Cureus"                         | 4212             |
├──────────────────────────────────┼──────────────────┤
| "Reactions Weekly"               | 3891             |
├──────────────────────────────────┼──────────────────┤
| "Front Psychol"                  | 3541             |
├──────────────────────────────────┼──────────────────┤
| "BMJ Open"                       | 3515             |
├──────────────────────────────────┼──────────────────┤
| "Front Immunol"                  | 3442             |
└──────────────────────────────────┴──────────────────┘
```

Netflix Movies and TV Shows

> **Примечание**
>
> Эта страница является частью раздела Импорт датасетов, где описаны примеры загрузки популярных наборов данных в YDB. Перед началом работы ознакомьтесь с общей информацией о требованиях и процессе импорта.

Данные о фильмах и сериалах на платформе Netflix.

Источник: Kaggle - Netflix Movies and TV Shows

Размер: 3.4 MB

Пример загрузки

1. Скачайте и разархивируйте файл `netflix_titles.csv` с Kaggle
2. Создайте таблицу в YDB одним из следующих способов:

YDB UI

Подробнее про YDB UI.

```
CREATE TABLE `netflix` (
    `show_id` Text NOT NULL,
    `type` Text NOT NULL,
    `title` Text NOT NULL,
    `director` Text NOT NULL,
    `cast` Text,
    `country` Text NOT NULL,
    `date_added` Text NOT NULL,
    `release_year` Uint64 NOT NULL,
    `rating` Text NOT NULL,
    `duration` Text NOT NULL,
    `listed_in` Text NOT NULL,
    `description` Text NOT NULL,
    PRIMARY KEY (`show_id`)
)
WITH (
    STORE = COLUMN
);
```

YDB CLI

```
ydb sql -s \
'CREATE TABLE `netflix` (
    `show_id` Text NOT NULL,
    `type` Text NOT NULL,
    `title` Text NOT NULL,
    `director` Text NOT NULL,
    `cast` Text,
    `country` Text NOT NULL,
    `date_added` Text NOT NULL,
    `release_year` Uint64 NOT NULL,
    `rating` Text NOT NULL,
    `duration` Text NOT NULL,
    `listed_in` Text NOT NULL,
    `description` Text NOT NULL,
    PRIMARY KEY (`show_id`)
)
WITH (
    STORE = COLUMN
);'
```

3. Выполните команду импорта:

```
ydb import file csv --header --null-value "" --path netflix netflix_titles.csv
```

Пример аналитического запроса

Определим три страны, из которых было добавлено больше всего контента на Netflix в 2020 году:

YDB UI

```
SELECT
    country,
    COUNT(*) AS count
FROM netflix
WHERE
    CAST(SUBSTRING(CAST(date_added AS String), 7, 4) AS Int32) = 2020
    AND date_added IS NOT NULL
GROUP BY country
ORDER BY count DESC
LIMIT 3;
```

YDB CLI

```
ydb sql -s \
'SELECT
    country,
    COUNT(*) AS count
FROM netflix
WHERE
    CAST(SUBSTRING(CAST(date_added AS String), 7, 4) AS Int32) = 2020
    AND date_added IS NOT NULL
GROUP BY country
ORDER BY count DESC
LIMIT 3;'
```

Результат:

```
┌─────────────────┬───────┐
| country         | count |
├─────────────────┼───────┤
| "United States" | 22    |
├─────────────────┼───────┤
| ""              | 7     |
├─────────────────┼───────┤
| "Canada"        | 3     |
└─────────────────┴───────┘
```

Animal Crossing New Horizons Catalog

> **Примечание**
>
> Эта страница является частью раздела Импорт датасетов, где описаны примеры загрузки популярных наборов данных в YDB. Перед началом работы ознакомьтесь с общей информацией о требованиях и процессе импорта.

Каталог предметов из популярной игры Animal Crossing: New Horizons.

Источник: Kaggle - Animal Crossing New Horizons Catalog

Размер: 51 KB

Пример загрузки

1. Скачайте и разархивируйте файл `accessories.csv` с Kaggle
2. Этот файл включает в себя маску BOM (Byte Order Mark). Однако команда импорта не поддерживает файлы с маской BOM. Чтобы устранить проблему, удалите BOM-байты из начала файла, выполнив следующую команду:

```
sed -i '1s/^\xEF\xBB\xBF//' accessories.csv
```

3. Имена колонок в файле содержат пробелы, что не совместимо с YDB, поскольку YDB не поддерживает пробелы в именах колонок. Необходимо заменить пробелы в именах колонок, например, на символы подчеркивания. Вы можете сделать это, выполнив следующую команду:

```
sed -i '1s/ /_/g' accessories.csv
```

4. Создайте таблицу в YDB одним из следующих способов:

YDB UI

Подробнее про YDB UI.

```
CREATE TABLE `accessories` (
    `Name` Text NOT NULL,
    `Variation` Text NOT NULL,
    `DIY` Text NOT NULL,
    `Buy` Text NOT NULL,
    `Sell` Uint64 NOT NULL,
    `Color_1` Text NOT NULL,
    `Color_2` Text NOT NULL,
    `Size` Text NOT NULL,
    `Miles_Price` Text NOT NULL,
    `Source` Text NOT NULL,
    `Source_Notes` Text NOT NULL,
    `Seasonal_Availability` Text NOT NULL,
    `Mannequin_Piece` Text NOT NULL,
    `Version` Text NOT NULL,
    `Style` Text NOT NULL,
    `Label_Themes` Text NOT NULL,
    `Type` Text NOT NULL,
    `Villager_Equippable` Text NOT NULL,
    `Catalog` Text NOT NULL,
    `Filename` Text NOT NULL,
    `Internal_ID` Uint64 NOT NULL,
    `Unique_Entry_ID` Text NOT NULL,
    PRIMARY KEY (`Unique_Entry_ID`)
)
WITH (
    STORE = COLUMN
);
```

YDB CLI

```
ydb sql -s \
'CREATE TABLE `accessories` (
    `Name` Text NOT NULL,
    `Variation` Text NOT NULL,
    `DIY` Text NOT NULL,
    `Buy` Text NOT NULL,
    `Sell` Uint64 NOT NULL,
    `Color_1` Text NOT NULL,
    `Color_2` Text NOT NULL,
    `Size` Text NOT NULL,
    `Miles_Price` Text NOT NULL,
    `Source` Text NOT NULL,
    `Source_Notes` Text NOT NULL,
    `Seasonal_Availability` Text NOT NULL,
    `Mannequin_Piece` Text NOT NULL,
    `Version` Text NOT NULL,
    `Style` Text NOT NULL,
    `Label_Themes` Text NOT NULL,
    `Type` Text NOT NULL,
    `Villager_Equippable` Text NOT NULL,
    `Catalog` Text NOT NULL,
    `Filename` Text NOT NULL,
    `Internal_ID` Uint64 NOT NULL,
```

```
    `Unique_Entry_ID` Text NOT NULL,
    PRIMARY KEY (`Unique_Entry_ID`)
)
WITH (
    STORE = COLUMN
);'
```

5. Выполните команду импорта:

```
ydb import file csv --header --path accessories accessories.csv
```

Пример аналитического запроса

Определим пять самых популярных основных цветов аксессуаров:

YDB UI

```
SELECT
    Color_1,
    COUNT(*) AS color_count
FROM accessories
GROUP BY Color_1
ORDER BY color_count DESC
LIMIT 5;
```

YDB CLI

```
ydb sql -s \
'SELECT
    Color_1,
    COUNT(*) AS color_count
FROM accessories
GROUP BY Color_1
ORDER BY color_count DESC
LIMIT 5;'
```

Результат:

```
┌─────────┬─────────────┐
| Color_1  | color_count |
├─────────┼─────────────┤
| "Black"  | 31          |
├─────────┼─────────────┤
| "Green"  | 27          |
├─────────┼─────────────┤
| "Pink"   | 20          |
├─────────┼─────────────┤
| "Red"    | 20          |
├─────────┼─────────────┤
| "Yellow" | 19          |
└─────────┴─────────────┘
```

Работа с медленно меняющимися измерениями

В текущем разделе собраны практические руководства по реализации медленно меняющихся измерений (Slowly Changing Dimensions, SCD) — популярного подхода к управлению историческими данными в аналитических хранилищах.

В разделе описываются варианты SCD1 и SCD2:

- SCD Type 1: Старые значения атрибутов заменяются новыми, сохраняя только актуальное состояние данных.
- SCD Type 2: Сохраняется полная история изменений путем добавления новых записей для каждой новой версии атрибута.

Особенности SCD1

Медленно меняющиеся измерения (тип 1) или SCD1 (Type 1) — это подход, при котором при изменении атрибута измерения старое значение заменяется новым. Хранится только текущее состояние данных. Этот подход используется, когда:

- историческая информация не требуется;
- важно иметь только актуальные данные;
- необходимо минимизировать размер хранилища данных;
- требуется простая структура данных для аналитики.

Особенности SCD2 и подход append-only

Медленно меняющиеся измерения (тип 2) или SCD2 (Type 2) — это подход, при котором при изменении атрибута измерения создаётся новая запись, а старая помечается как неактуальная. Таким образом, сохраняется история изменений. Этот подход используется, когда:

- требуется отслеживать историю изменений данных;
- необходимо выполнять анализ данных с учётом временных периодов;
- важно сохранять аудиторский след изменений;
- требуется возможность восстановления состояния данных на определённый момент времени.

Доступные руководства

В разделе рассматриваются различные технические способы реализации этих механизмов:

- С использованием связки Change Data Capture (CDC) и Transfer для автоматической потоковой репликации изменений из таблиц-источников.
  - Использование механизма TRANSFER для реализации SCD1 на основе данных из CDC-источников в YDB
  - Использование механизма TRANSFER для реализации SCD2 на основе данных из CDC-источников в YDB
- С помощью периодических YQL-запросов, которые обрабатывают пакеты изменений из промежуточной таблицы и подмерживают их в основную SCD-таблицу.
  - Использование процесса подмерживания изменения для реализации SCD2 в YDB

Работа с медленно меняющимися измерениями

В текущем разделе собраны практические руководства по реализации медленно меняющихся измерений (Slowly Changing Dimensions, SCD) — популярного подхода к управлению историческими данными в аналитических хранилищах.

В разделе описываются варианты SCD1 и SCD2:

- SCD Type 1: Старые значения атрибутов заменяются новыми, сохраняя только актуальное состояние данных.
- SCD Type 2: Сохраняется полная история изменений путем добавления новых записей для каждой новой версии атрибута.

Особенности SCD1

Медленно меняющиеся измерения (тип 1) или SCD1 (Type 1) — это подход, при котором при изменении атрибута измерения старое значение заменяется новым. Хранится только текущее состояние данных. Этот подход используется, когда:

- историческая информация не требуется;
- важно иметь только актуальные данные;
- необходимо минимизировать размер хранилища данных;
- требуется простая структура данных для аналитики.

Особенности SCD2 и подход append-only

Медленно меняющиеся измерения (тип 2) или SCD2 (Type 2) — это подход, при котором при изменении атрибута измерения создаётся новая запись, а старая помечается как неактуальная. Таким образом, сохраняется история изменений. Этот подход используется, когда:

- требуется отслеживать историю изменений данных;
- необходимо выполнять анализ данных с учётом временных периодов;
- важно сохранять аудиторский след изменений;
- требуется возможность восстановления состояния данных на определённый момент времени.

Доступные руководства

В разделе рассматриваются различные технические способы реализации этих механизмов:

- С использованием связки Change Data Capture (CDC) и Transfer для автоматической потоковой репликации изменений из таблиц-источников.
  - Использование механизма TRANSFER для реализации SCD1 на основе данных из CDC-источников в YDB
  - Использование механизма TRANSFER для реализации SCD2 на основе данных из CDC-источников в YDB
- С помощью периодических YQL-запросов, которые обрабатывают пакеты изменений из промежуточной таблицы и подмерживают их в основную SCD-таблицу.
  - Использование процесса подмерживания изменения для реализации SCD2 в YDB

Использование механизма TRANSFER для реализации SCD1 на основе данных из CDC-источников в YDB

В этой статье описывается реализация паттерна Slowly Changing Dimensions Type 1 (SCD1) в YDB.

Используемые инструменты

Для поставки данных в SCD1-таблицу в данной статье будет использоваться следующая комбинация функциональных возможностей YDB:

1. Таблица-источник будет строковой для оперативных транзакционных изменений;
2. Таблица-приёмник будет колоночной для эффективного выполнения аналитических запросов;
3. Подписка на изменения в таблице-источнике будет осуществляться через механизм Change Data Capture (CDC);
4. За буферизацию изменений будет отвечать неявно создаваемый под CDC топик;
5. За автоматическое перекладывание данных из CDC-топика в таблицу-приёмник будет отвечать трансфер.

> **Важно**
>
> Transfer не может удалять строки из таблиц, а может выполнять только `UPSERT` в таблицу-приёмник.
> Поэтому с помощью Transfer можно реализовать только SCD1-soft: удаление помечается флагом и сохраняется момент удаления. Если нужна физическая очистка данных, её можно выполнять через явную команду `DELETE`.

Создайте таблицу-источник данных, которая будет генерировать CDC-события

```
CREATE TABLE source_customers (
    id Utf8 NOT NULL,
    attribute1 Utf8,
    attribute2 Utf8,
    change_time Timestamp,
    PRIMARY KEY (id)
);

ALTER TABLE `source_customers` ADD CHANGEFEED `updates` WITH (
  FORMAT = 'DEBEZIUM_JSON',
  MODE = 'NEW_AND_OLD_IMAGES'
);
```

Создайте таблицу-приемник данных в формате SCD1

```
CREATE TABLE dimension_scd1 (
    id Utf8 NOT NULL,
    attribute1 Utf8,
    attribute2 Utf8,
    is_deleted Uint8,
    last_update_timestamp Timestamp,
    PRIMARY KEY (id)
)
PARTITION BY HASH(id)
WITH (
  STORE=COLUMN
)
```

Создайте трансфер и lambda-функцию для обработки CDC-данных

Особенности обработки CDC данных в формате Debezium:

- Формат сообщения содержит поля `payload.before` и `payload.after` с состоянием до и после изменения.
- Поле `payload.op` указывает тип операции: "c" (create), "u" (update), "d" (delete).
- Поле `payload.ts_ms` содержит временную метку события в миллисекундах.
- При удалении (`op = "d"`) данные находятся в поле `before` , а при создании/обновлении - в поле `after` . Если выполняется удаление несуществующей строки, то будет сформировано сообщение с типом операции `op = "d"` , но с пустыми полями `before` и `after` .

```
$transformation_lambda = ($msg) -> {
    $cdc_data = CAST($msg._data AS Json);

    -- Определяем тип операции
    $operation = Json::ConvertToString($cdc_data.payload.op);
    $is_deleted = $operation == "d";

    -- Получаем данные в зависимости от типа операции
    $data = IF($is_deleted, $cdc_data.payload.before, $cdc_data.payload.after);

    -- Если данные не пришли (выполнена команда DELETE на несуществующий ключ, то проигнорируем запись)
    return IF($data IS NOT NULL,
        <|
            id: Unwrap(CAST(Json::ConvertToString($data.id) AS Utf8)),
            attribute1: CAST(Json::ConvertToString($data.attribute1) AS Utf8),
            attribute2: CAST(Json::ConvertToString($data.attribute2) AS Utf8),
            is_deleted: CAST($is_deleted AS Uint8),
            last_update_timestamp: DateTime::FromMilliseconds(Json::ConvertToUint64($cdc_data.payload.source.
ts_ms))
      |>
    , NULL);
};
```

```
-- В данном примере настраивается высокая частота обновлений таблицы-приемника.
-- Это делается исключительно для наглядности. Для production-сценариев стоит настраивать большие значения
CREATE TRANSFER dimension_scd1_cdc_transfer
  FROM `source_customers/updates` TO dimension_scd1 USING $transformation_lambda
  WITH (
    FLUSH_INTERVAL=Interval("PT1S")
  );
```

Примечание

Колонка `id` в принимающей таблице `dimension_scd1` объявлена как `Utf8 NOT NULL`, при этом в Json CDC могут быт переданы данные, которые невозможно привести к строке, то есть результатом конвертации данных из Json может быть значение `NULL`. Функция `Unwrap` гарантирует, что после ее выполнения не может быть значения `NULL` или будет ошибка времени выполнения. Это позволяет гарантировать, что результатом выполнения lambda-функции или будет полностью корректная структура данных, или будет ошибка времени выполнения.

Демонстрация работы

Для демонстрации работы с данными CDC запишем данные в таблицу-источник в YDB, которая будет генерировать CDC-события:

Вставка новой записи

```
INSERT INTO source_customers (id, attribute1, attribute2)
VALUES ('CUSTOMER_1001', 'John Doe', 'New York');
```

Это действие создаст CDC-событие с типом операции `op = "c"` примерно следующего вида:

```
{
  "payload":{
    "op":"c",
    "source":{
      "txId":18446744073709551615,
      "connector":"ydb",
      "version":"1.0.0",
      "step":1755868513260,
      "ts_ms":1755868513219,
      "snapshot":false
    },
    "after":{
      "attribute1":"John Doe",
      "id":"CUSTOMER_1001",
      "attribute2":"New York"
    }
  }
}
```

В результате исполнения команды выше в таблице `dimension_scd1` будет следующее содержимое:

id	attribute1	attribute2	is_deleted	last_update_timestamp
CUSTOMER_1001	John Doe	New York	0	2025-08-22T13:15:13.219000Z

Обновление существующей записи

```
UPSERT INTO source_customers (id, attribute1, attribute2)
VALUES ('CUSTOMER_1001', 'John Doe', 'Los Angeles');
```

Это действие создаст CDC-событие с типом операции `op = "u" примерно следующего вида:

```
{
  "payload":{
      "op":"u",
      "source":{
        "txId":18446744073709551615,
        "connector":"ydb",
        "version":"1.0.0",
        "step":1755868795050,
        "ts_ms":1755868795719,
        "snapshot":false
    },
    "after":{
      "attribute1":"John Doe",
      "id":"CUSTOMER_1001",
      "attribute2":"Los Angeles"
    },
    "before":{
      "attribute1":"John Doe",
      "id":"CUSTOMER_1001",
      "attribute2":"New York"
    }
```

```
  }
}
```

В результате исполнения команды выше в таблице `dimension_scd1` будет следующее содержимое:

id	attribute1	attribute2	is_deleted	last_update_timestamp
CUSTOMER_1001	John Doe	Los Angeles	0	2025-08-22T13:19:55.719000Z

Удаление записи

```
DELETE FROM source_customers WHERE id = 'CUSTOMER_1001';
```

Это действие создаст CDC-событие с типом операции `op = "d"` примерно следующего вида:

```
{
  "payload":{
    "op":"d",
    "source":{
      "txId":18446744073709551615,
      "connector":"ydb",
      "version":"1.0.0",
      "step":1755868931000,
      "ts_ms":1755868931109,
      "snapshot":false
    },
    "before":{
      "attribute1":"John Doe",
      "id":"CUSTOMER_1001",
      "attribute2":"Los Angeles"
    }
  }
}
```

В результате исполнения команды выше в таблице `dimension_scd1` будет следующее содержимое:

id	attribute1	attribute2	is_deleted	last_update_timestamp
CUSTOMER_1001	John Doe	Los Angeles	1	2025-08-22T13:22:11.109000Z

Пример удаления данных из таблицы-приемника

Для удаления данных из таблицы-приемника можно использовать явную команду `DELETE`:

```
DELETE from dimension_scd1 WHERE is_deleted != 0ut
```

Использование механизма TRANSFER для реализации SCD2 на основе данных из CDC-источников в YDB

В этой статье описывается реализация паттерна Slowly Changing Dimensions Type 2 (SCD2) в YDB.

Используемые инструменты

Для поставки данных в SCD2 таблицу в данной статье будет использоваться следующая комбинация из доступной в YDB функциональности:

1. Таблица-источник будет строковой для оперативных транзакционных изменений;
2. Таблица-приёмник будет колоночной для эффективного выполнения аналитических запросов;
3. Подписка на изменения в таблице-источнике будет осуществляться через механизм Change Data Capture (CDC);
4. За буферизацию изменений будет отвечать неявно создаваемый для CDC топик;
5. За автоматическое перекладывание данных из CDC-топика в таблицу-приёмник будет отвечать трансфер.

Создайте таблицу-источник данных, которая будет генерировать CDC-события

```
CREATE TABLE source_customers (
    id Utf8 NOT NULL,
    attribute1 Utf8,
    attribute2 Utf8,
    change_time Timestamp,
    PRIMARY KEY (id)
);

ALTER TABLE `source_customers` ADD CHANGEFEED `updates` WITH (
  FORMAT = 'DEBEZIUM_JSON',
  MODE = 'NEW_AND_OLD_IMAGES'
);
```

Создайте таблицу-приемник данных в формате SCD2

```
  CREATE TABLE dimension_scd2 (
    id Utf8 NOT NULL,
    attribute1 Utf8,
    attribute2 Utf8,
    valid_from Timestamp NOT NULL,
    deleted_at Timestamp ,
    PRIMARY KEY (valid_from, id)
  )
  PARTITION BY HASH(valid_from, id)
  WITH (
    STORE=COLUMN
  )
```

Создайте трансфер и lambda-функцию для обработки CDC-данных

Особенности обработки CDC данных в формате Debezium:

- При создании записи (`op = "c"`) данные берутся из поля `after`, в поле `valid_from` сохраняется значение момента времени изменения записи из поля `changed_at`.
- При обновлении записи (`op = "u"`) данные также берутся из поля `after`, в поле `valid_from` сохраняется значение момента времени изменения записи из поля `changed_at`.
- При удалении записи (`op = "d"`) данные берутся из поля `before`, а в поле `deleted_at` устанавливается системный момент времени получения информации про удаление записи.

```
$transformation_lambda = ($msg) -> {
$cdc_data = CAST($msg._data AS Json);

-- Определяем тип операции
$operation = Json::ConvertToString($cdc_data.payload.op);
$is_deleted = $operation == "d";

-- Получаем данные в зависимости от типа операции
$data = IF($is_deleted, $cdc_data.payload.before, $cdc_data.payload.after);

$result =  IF($data is not NULL, <|
        id: Unwrap(CAST(Json::ConvertToString($data.id) AS Utf8)),
        attribute1: CAST(Json::ConvertToString($data.attribute1) AS Utf8),
        attribute2: CAST(Json::ConvertToString($data.attribute2) AS Utf8),
        deleted_at: IF($is_deleted, DateTime::FromMilliseconds(Json::ConvertToUint64($cdc_data.payload.sourc
e.ts_ms)), NULL),
        valid_from : Unwrap(cast(datetime::MakeDatetime(datetime::ParseIso8601(Json::ConvertToString($data.ch
ange_time))) as timestamp)),
    |>, NULL);

return $result;
};

-- В данном примере настраивается высокая частота обновлений таблицы-приемника.
-- Это делается исключительно для наглядности. Для production-сценариев стоит настраивать большие значения
  CREATE TRANSFER dimension_scd2_cdc_transfer
    FROM `source_customers/updates` TO dimension_scd2 USING $transformation_lambda
    WITH (
```

```
    FLUSH_INTERVAL=Interval("PT1S")
  );
```

> **Примечание**
>
> Колонка `id` в принимающей таблице `dimension_scd2` объявлена как `Utf8 NOT NULL`, при этом в Json CDC могут быт переданы данные, которые невозможно привести к строке, то есть результатом конвертации данных из Json может быть значение `NULL`. Функция `Unwrap` гарантирует, что после ее выполнения не может быть значения `NULL` или будет ошибка времени выполнения. Это позволяет гарантировать, что результатом выполнения lambda-функции или будет полностью корректная структура данных, или будет ошибка времени выполнения.

Демонстрация работы

Для демонстрации работы с данными CDC запишем данные в таблицу-источник в YDB, которая будет генерировать CDC-события:

Вставка новой записи

```
INSERT INTO source_customers (id, attribute1, attribute2, change_time)
VALUES ('CUSTOMER_1001', 'John Doe', 'Los Angeles', CurrentUtcTimestamp());
```

Это действие создаст CDC-событие с типом операции `op = "c"` примерно следующего вида:


```
{
  "payload":{
    "op":"c",
    "source":{
      "txId":18446744073709551615,
      "connector":"ydb",
      "version":"1.0.0",
      "step":1755883683690,
      "ts_ms":1755883683652,
      "snapshot":false
    },
    "after":{
      "attribute1":"John Doe",
      "change_time":"2025-08-22T17:28:03.648313Z",
      "id":"CUSTOMER_1001",
      "attribute2":"Los Angeles"
    }
  }
}
```


В результате исполнения команды выше в таблице `dimension_scd2` будет следующее содержимое:

id	attribute1	attribute2	deleted_at	valid_from
CUSTOMER_1001	John Doe	New York	NULL	2025-08-22T17:28:03.648313Z

Обновление новой записи

```
UPSERT INTO source_customers (id, attribute1, attribute2, change_time)
VALUES ('CUSTOMER_1001', 'John Doe 2', 'Los Angeles 2', CurrentUtcTimestamp());
```

Это действие создаст CDC-событие с типом операции `op = "u"` примерно следующего вида:


```
{
  "payload":{
    "op":"u",
    "source":{
      "txId":18446744073709551615,
      "connector":"ydb",
      "version":"1.0.0",
      "step":1755883278000,
      "ts_ms":1755883278357,
      "snapshot":false
    },
    "after":{
      "attribute1":"John Doe 2",
      "change_time":"2025-08-22T17:31:18.357503Z",
      "id":"CUSTOMER_1001",
      "attribute2":"Los Angeles 2"
    },
    "before":{
      "attribute1":"John Doe 2",
      "change_time":"2025-08-22T17:28:03.648313Z",
      "id":"CUSTOMER_1001",
      "attribute2":"Los Angeles 2"
    }
  }
}
```


В результате исполнения команды выше в таблице `dimension_scd2` будет следующее содержимое:

id	attribute1	attribute2	deleted_at	valid_from
CUSTOMER_1001	John Doe	New York	NULL	2025-08-22T17:28:03.648313Z
CUSTOMER_1001	John Doe 2	New York 2	NULL	2025-08-22T17:31:18.357503Z

Удаление записи

```
DELETE FROM source_customers WHERE id = 'CUSTOMER_1001';
```

Это действие создаст CDC-событие с типом операции `op = "d"` примерно следующего вида:

```
{
  "payload":{
    "op":"d",
    "source":{
      "txId":18446744073709551615,
      "connector":"ydb",
      "version":"1.0.0",
      "step":1755883827000,
      "ts_ms":1755883827948,
      "snapshot":false
    },
    "before":{
      "attribute1":"John Doe",
      "change_time":"2025-08-22T17:38:03.648313Z",
      "id":"CUSTOMER_1001",
      "attribute2":"Los Angeles"
    }
  }
}
```

В результате исполнения команды выше в таблице `dimension_scd2` будет следующее содержимое:

id	attribute1	attribute2	deleted_at	valid_from
CUSTOMER_1001	John Doe	New York	NULL	2025-08-22T17:28:03.648313Z
CUSTOMER_1001	John Doe 2	New York 2	NULL	2025-08-22T17:31:18.357503Z
CUSTOMER_1001	John Doe 2	New York 2	2025-08-22T17:38:03.648313Z	2025-08-22T17:38:03.648313Z

Пример запроса для получения актуальных данных

Для получения данных из SCD2 append only таблиц можно использовать следующий запрос, который получает данные на момент времени `2025-08-22 19:11:30`:

```
DECLARE $as_of AS Timestamp;
$as_of = Timestamp("2025-08-22T19:11:30.000000Z");

SELECT
  id,
  data.attribute1 AS attribute1,
  data.attribute2 AS attribute2,
  data.valid_from AS valid_from,
  data.deleted_at AS deleted_at
FROM (
  SELECT
    id,
    MAX_BY(TableRow(), valid_from) AS data
  FROM dimension_scd2
  WHERE
      valid_from <= $as_of
      AND (deleted_at IS NULL OR deleted_at > $as_of)  -- запись ещё не удалена к моменту X
  GROUP BY id
)
```

Использование процесса подмерживания изменения для реализации SCD2 в YDB

В этой статье описывается реализация паттерна Slowly Changing Dimensions Type 2 (SCD2) в YDB с использованием процесса подмерживания изменений.

Используемые инструменты

Для поставки данных в SCD2 таблицу в данной статье будет использоваться следующая комбинация из доступной в YDB функциональности:

1. Таблица-источник `dimension_scd_changes`, содержащая информацию об атрибутах, их значениях и моментах изменений данных.
2. Таблица-приёмник `dimension_scd2_final` для хранения результирующих данных.
3. Периодически внешнее приложение должно вызывать запрос, который будет подмерживать изменения данных, накопившиеся в таблице `dimension_scd_changes`, в таблицу `dimension_scd2_final`.
4. Для поставки данных из строковых таблиц для хранения их в формате SCD2 удобно использовать встроенный в YDB механизм трансфера.

> **Примечание**
>
> Таблицы `dimension_scd_changes`, `dimension_scd2_final` приведены для иллюстрации. Для реальных запросов вам нужно скорректировать структуру таблиц и их атрибутов.

Создание таблицы для приёма всех изменений `dimension_scd_changes`

```
CREATE TABLE dimension_scd_changes (
    id Utf8 NOT NULL, -- Бизнес-ключ
    attribute1 Utf8,  -- Атрибут данных
    attribute2 Utf8,  -- Атрибут данных
    change_time Timestamp NOT NULL, -- Момент изменения данных
    operation Utf8, -- Тип изменений данных
    PRIMARY KEY (change_time, id)
)
PARTITION BY HASH(change_time, id)
WITH (
    STORE=COLUMN
)
```

Описание полей таблицы:

- `id` — бизнес-ключ записи;
- `attribute1`, `attribute2` — атрибуты измерения;
- `change_time` — момент времени изменения данных;
- `operation` — тип изменения данных: `CREATE`, `UPDATE`, `DELETE`.

Первичный ключ создается как `PRIMARY KEY (change_time, id)`, так как по одному и тому же бизнес-ключу данных может происходить множество изменений и все эти изменения по одному ключу важно сохранять. Подробнее про выбор первичного ключа и ключа партиционирования, можно прочесть в документации - выбор первичного ключа, выбор ключа партиционирования

Создание финальной SCD2 таблицы `dimension_scd2_final`

```
CREATE TABLE dimension_scd2_final (
    id Utf8 NOT NULL, -- Бизнес-ключ данных
    attribute1 Utf8,  -- Атрибут данных
    attribute2 Utf8,  -- Атрибут данных
    valid_from Timestamp NOT NULL, -- Момент времени, с которого данные актуальны
    valid_to Timestamp,            -- Момент времени, до которого данные актуальны.
                                   -- Если данные актуальны прямо сейчас, то в valid_to находится NULL
    is_current Uint8,              -- Признак, что данные актуальны прямо сейчас.
    is_deleted Uint8,              -- Признак, что данные были удалены. Если данные были удалены, то is_curre
nt = FALSE
    PRIMARY KEY (valid_from, id)
)
PARTITION BY HASH(valid_from, id)
WITH(
    STORE=COLUMN
)
```

Описание полей таблицы:

- `id` — бизнес-ключ записи;
- `attribute1`, `attribute2` — атрибуты измерения;
- `valid_from` — момент времени, с которого запись становится актуальной;
- `valid_to` — момент времени, до которого запись была актуальной, или `NULL` для текущих записей;
- `is_current` — флаг, указывающий, является ли запись текущей (1 - текущая запись) или (0 - запись историческая);
- `is_deleted` — флаг, указывающий, была ли запись удалена (1 - запись была удалена) или (0 - запись не была удалена).

Первичный ключ создается как `PRIMARY KEY (valid_from, id)`, так как по одному и тому же ключу данных может происходить множество изменений и все эти изменения по одному ключу важно сохранять.

Загрузка данных в таблицу изменений

Для загрузки данных в таблицу изменений можно использовать любой способ загрузки данных и автоматическую поставку изменений с помощью механизма трансфер.

Пример запроса для явной загрузки изменений:

```
UPSERT INTO dimension_scd_changes (id, attribute1, attribute2, change_time, operation)
VALUES ('CUSTOMER_1001', 'John Doe', 'Los Angeles', Unwrap(CAST('2025-08-22T17:00:00Z' as Timestamp)), 'CREAT
E');

UPSERT INTO dimension_scd_changes (id, attribute1, attribute2, change_time, operation)
VALUES ('CUSTOMER_1002', 'John Doe', 'New York', Unwrap(CAST('2025-08-22T17:00:00Z' as Timestamp)), 'CREAT
E');

UPSERT INTO dimension_scd_changes (id, attribute1, attribute2, change_time, operation)
VALUES ('CUSTOMER_1001', 'John Doe', 'San Francisco', Unwrap(CAST('2025-08-22T19:00:00Z' as Timestamp)), 'UPD
ATE');

UPSERT INTO dimension_scd_changes (id, attribute1, attribute2, change_time, operation)
VALUES ('CUSTOMER_1002', 'John Doe', 'New York', Unwrap(CAST('2025-08-22T21:00:00Z' as Timestamp)), 'DELET
E');
```

Запрос для размещения изменений в формате SCD2

Чтобы преобразовать данные из таблицы изменений в формат SCD2 и загрузить их в финальную таблицу, используется специальный запрос. Этот запрос нужно запускать регулярно — с такой периодичностью, с какой вы хотите обновлять данные в финальной таблице. Для автоматического запуска можно воспользоваться интеграцию YDB с Apache Airflow™:

```
-- Шаг 1: Читаем все новые события из таблицы `dimension_scd_changes`.
-- Это именованное выражение ($changes) является исходным набором данных для всей последующей обработки в рам
ках этого запуска.
$changes = (
    SELECT
        id,
        attribute1,
        attribute2,
        change_time,
        String::AsciiToUpper(operation) AS op
    FROM dimension_scd_changes
);

-- Шаг 2: Фильтруем события, оставляя только те, которых еще нет в целевой таблице.
-- Цель этого шага - обеспечить идемпотентность на уровне чтения, чтобы не обрабатывать
-- уже загруженные данные в случае сбоя и перезапуска скрипта.
$unprocessed_data = (
    SELECT
        chg.id AS id,
        chg.attribute1 AS attribute1,
        chg.attribute2 AS attribute2,
        chg.change_time AS change_time,
        chg.op AS op
    FROM $changes AS chg
    LEFT JOIN dimension_scd2_final AS scd
        ON chg.id = scd.id AND chg.change_time = scd.valid_from -- Ищем записи по каждой сущности (id) и врем
ени изменения
    WHERE scd.id IS NULL -- для исключения строк, которые уже были перенесены в таблицу dimension_scd2_final
ранее
);

-- Шаг 3: Находим в целевой таблице активные записи (`is_current=1`), для которых пришли обновления.
-- Формируем для них "закрывающие" версии, устанавливая `valid_to` равным времени
-- самого первого изменения из новой пачки ($unprocessed_data).
$close_open_intervals = (
    SELECT
        target.id AS id,
        target.attribute1 as attribute1,
        target.attribute2 as attribute2,
        target.valid_from as valid_from,
        0ut AS is_current, -- Закрываемая запись больше не является текущей
        unprocessed_data.change_time AS valid_to,
        target.is_deleted as is_deleted
    FROM dimension_scd2_final AS target
    INNER JOIN (
        SELECT
            id,
            MIN(change_time) AS change_time
        FROM $unprocessed_data
        GROUP BY id
    ) AS unprocessed_data
    ON target.id = unprocessed_data.id
    WHERE target.is_current = 1ut
);

-- Шаг 4: Преобразуем поток необработанных событий в версионные записи (строки для вставки).
-- Здесь вычисляются все необходимые атрибуты для новых версий: `valid_to`, `is_current`, `is_deleted`.
$updated_data = (
    SELECT
        t.id AS id,
        t.attribute1 AS attribute1,
```

```
        t.attribute2 AS attribute2,
        t.is_deleted AS is_deleted,
        -- Логика флага `is_current`: он устанавливается в 1 только для последней
        -- записи в цепочке (`next_change_time IS NULL`), и только если это не
        -- операция удаления (`is_deleted == 0`).
        IF(t.next_change_time IS NOT NULL OR t.is_deleted == 1ut, 0ut, 1ut) AS is_current,
        t.change_time AS valid_from,
        t.next_change_time AS valid_to
    FROM (
        -- Подзапрос вычисляет для каждой строки флаг удаления (`is_deleted`)
        -- и временную метку следующего события (`next_change_time`) с помощью оконной функции LEAD.
        SELECT
            unprocessed_data.id AS id,
            unprocessed_data.attribute1 AS attribute1,
            unprocessed_data.attribute2 AS attribute2,
            unprocessed_data.op AS op,
            unprocessed_data.change_time AS change_time,
            IF(unprocessed_data.op = "DELETE", 1ut, 0ut) AS is_deleted,
            LEAD(unprocessed_data.change_time) OVER (PARTITION BY id ORDER BY unprocessed_data.change_time) A
S next_change_time
        FROM $unprocessed_data AS unprocessed_data
    ) AS t
);

-- Шаг 5: Атомарно применяем все рассчитанные изменения к целевой таблице.
-- UPSERT обновит существующие записи (из $close_open_intervals) и вставит новые (из $updated_data).
UPSERT INTO dimension_scd2_final (id, attribute1, attribute2, is_current, is_deleted, valid_from, valid_to)
SELECT
    id,
    attribute1,
    attribute2,
    is_current,
    is_deleted,
    valid_from,
    valid_to
FROM $close_open_intervals
UNION ALL
SELECT
    id,
    attribute1,
    attribute2,
    is_current,
    is_deleted,
    valid_from,
    valid_to
FROM $updated_data;

-- Шаг 6: Очищает стейджинг-таблицу от обработанных записей.
DELETE FROM dimension_scd_changes ON
SELECT id, change_time FROM $changes;
```

Демонстрация работы

В примере ниже рассматривается сущность **Customer**:

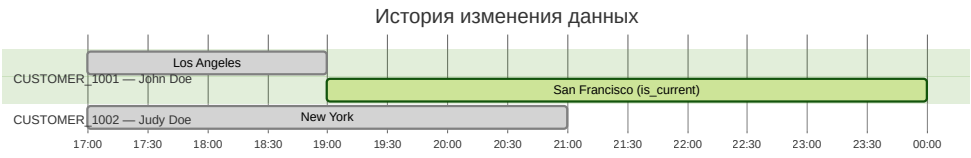
- бизнес-ключ — поле `id`,
- атрибуты — `attribute1` (полное имя) и `attribute2` (город).

В момент времени `2025-08-22 17:00` создаются два клиента (John в Los Angeles - с id `CUSTOMER_1001` и Judy в New York с id `CUSTOMER_1002`), в момент времени `2025-08-22 19:00` клиент `CUSTOMER_1001` меняет город на San Francisco `UPDATE`, а в момент `2025-08-22 21:00` клиент `CUSTOMER_1002` удаляется `DELETE`.

id	attribute1	attribute2	change_time	operation
CUSTOMER_1001	John Doe	Los Angeles	2025-08-22 17:00	CREATE
CUSTOMER_1002	Judy Doe	New York	2025-08-22 17:00	CREATE
CUSTOMER_1001	John Doe	San Francisco	2025-08-22 19:00	UPDATE
CUSTOMER_1002	Judy Doe	New York	2025-08-22 21:00	DELETE

Процесс SCD2 преобразует такие события в интервальные версии записей с полями `valid_from` и `valid_to`. Например, у `CUSTOMER_1001` получится две последовательные версии: сначала с городом LA, затем с городом SF (актуальная запись, у которой `valid_to = NULL`). У `CUSTOMER_1002` будет одна устаревшая версия и последняя запись с флагами `is_deleted=1` и `is_current=0`, которая показывает, что пользователь удалён.

Ниже показаны исходные события и соответствующие им версии в финальной таблице.



id	attribute1	attribute2	valid_from	valid_to	is_current	is_deleted
CUSTOMER_1001	John Doe	Los Angeles	2025-08-22 17:00	2025-08-22 19:00	0	0

CUSTOMER_1001	John Doe	San Francisco	2025-08-22 19:00	NULL	1	0
CUSTOMER_1002	Judy Doe	New York	2025-08-22 17:00	2025-08-22 21:00	0	0
CUSTOMER_1002	Judy Doe	New York	2025-08-22 21:00	NULL	0	1

Получение данных из SCD2-таблицы

Получение актуальных данных

```
SELECT
    id,
    attribute1,
    attribute2,
    valid_from,
    valid_to
FROM dimension_scd2_final
WHERE is_current = 1ut;
```

Результат:

id	attribute1	attribute2	valid_from	valid_to	is_current	is_deleted
CUSTOMER_1001	John Doe	San Francisco	2025-08-22 19:00	NULL	1	0

Получение данных на определённый момент времени

```
$as_of = Timestamp("2025-08-22T19:11:30.000000Z");

SELECT
    id,
    attribute1,
    attribute2,
    valid_from,
    valid_to
FROM dimension_scd2_final
WHERE valid_from <= $as_of
    AND (valid_to IS NULL OR valid_to > $as_of) -- Получаем записи, которые действовали в $as_of момент време
ни
    AND is_deleted = 0ut -- Только записи, которые не удалены
```

Результат:

id	attribute1	attribute2	valid_from	valid_to
CUSTOMER_1001	John Doe	San Francisco	2025-08-22 19:00	NULL
CUSTOMER_1002	Judy Doe	New York	2025-08-22 17:00	2025-08-22 21:00

Получение истории изменений для конкретной записи

```
SELECT
    id,
    attribute1,
    attribute2,
    valid_from,
    valid_to,
    is_current,
    is_deleted
FROM dimension_scd2_final
WHERE id = 'CUSTOMER_1001'
ORDER BY valid_from;
```

Результат:

id	attribute1	attribute2	valid_from	valid_to	is_current	is_deleted
CUSTOMER_1001	John Doe	Los Angeles	2025-08-22 17:00	2025-08-22 19:00	0	0
CUSTOMER_1001	John Doe	San Francisco	2025-08-22 19:00	NULL	1	0

Известные ограничения

В данном разделе собраны важные особенности YDB, которые необходимо учитывать при проектировании приложений и написании запросов. Для каждой особенности описывается текущее поведение и возможные способы обхода.

Транзакции и изоляция

Уровни изоляции транзакций

YDB поддерживает разные уровни изоляции для строковых (OLTP) и колоночных (OLAP) таблиц.

- Строковые таблицы. Поддерживают транзакции с уровнями изоляции Serializable, `Online Read-Only` и другими. Это обеспечивает строгую консистентность для OLTP-нагрузок.
- Колоночные таблицы. Все операции выполняются в режиме `Serializable`. Это гарантирует, что любая аналитическая транзакция работает с консистентным срезом данных, как если бы в базе данных больше не происходило никаких изменений.

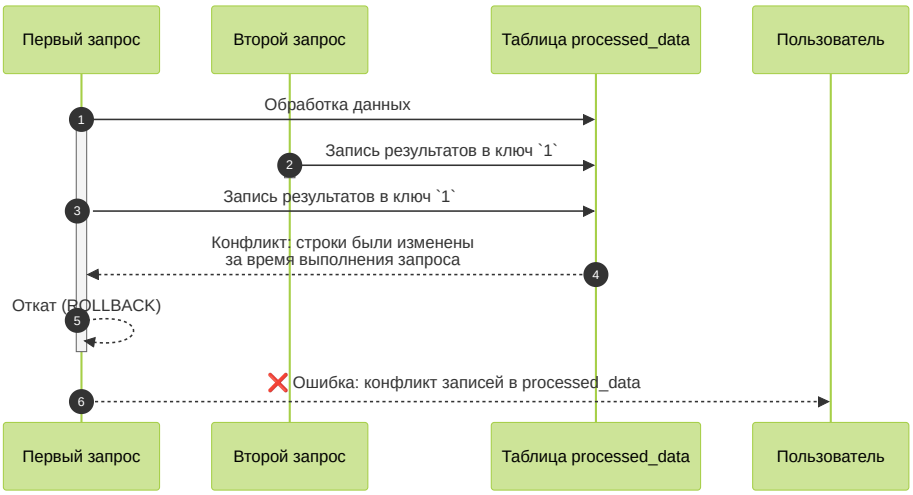
Далее рассматриваются особенности работы с режимом `Serializable` в контексте аналитических (OLAP) запросов.

Особенности Serializable-изоляции при параллельной работе с данными

Уровень изоляции `Serializable` гарантирует отсутствие аномалий чтения, но накладывает определённые требования на проектирование ETL/ELT-пайплайнов. Конфликты возникают, если данные, которые транзакция прочитала или собирается изменить, были изменены другой, уже завершённой транзакцией.

Конфликт записи (Write-Write Conflict)

Две параллельные задачи пытаются записать данные в один и тот же диапазон ключей в выходной таблице. Первая задача, которая начала выполнение, но работает медленнее, будет отменена при попытке коммита, потому что вторая, более быстрая задача, уже успела изменить эти строки.

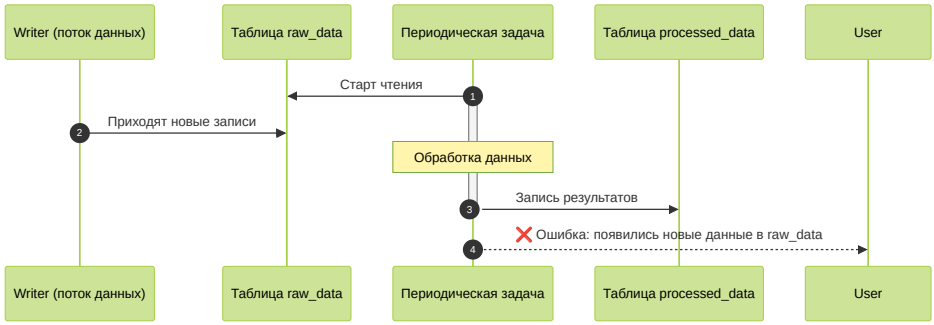


Варианты решения:

- **Партиционирование нагрузки:** разделите входные данные так, чтобы параллельные запросы записывали данные в разные таблицы или ключи. Например, обрабатывайте данные по дням или по идентификаторам пользователей в разных задачах;
- **Использование промежуточных таблиц:** каждая задача пишет результат в свою временную таблицу. После завершения всех задач данные из временных таблиц переносятся в итоговую таблицу одной транзакцией с помощью `INSERT INTO ... SELECT FROM`.

Конфликт чтения (Read-Write Conflict)

Длительный аналитический запрос читает данные из таблицы `raw_data`, в которую в это же время продолжается запись. К моменту, когда запрос завершает чтение и обработку, исходная таблица уже изменилась. YDB обнаруживает это и отменяет запрос, чтобы гарантировать консистентность среза.



Решение

Использование промежуточных таблиц: Создайте временную таблицу с необходимыми данными и выполняйте дальнейшую обработку уже с ней. Это фиксирует состояние данных и исключает конфликты.

```
CREATE TABLE temp_snapshot AS SELECT * FROM raw_data WHERE ...;
-- Далее работаем только с temp_snapshot
INSERT INTO processed_data SELECT process(t.*) FROM temp_snapshot AS t;
```

Модифицирующие запросы одновременно к колоночным и строковым таблицам запрещены

В рамках одной транзакции невозможно выполнять операции изменения данных (DML) одновременно и для строковых, и для колоночных таблиц.

Решение

Разделяйте логику на две последовательные транзакции. Сначала выполните операцию со строковой таблицей, а после ее успешного завершения — с колоночной (или наоборот, в зависимости от бизнес-логики).

Особенности синтаксиса

Common table expression (CTE) не поддерживаются

YQL не поддерживает синтаксис `WITH ... AS (CTE)`. Вместо этого используется механизм именованных выражений с помощью переменных, начинающихся со знака `$`.

Решение

Использование именованных выражений (named expressions). Это синтаксический аналог CTE, который позволяет декомпозировать сложные запросы.

Часть запроса можно выделить в отдельное выражение и дать ему имя, начинающейся с `$`, с помощью механизма именованных выражений. Такое выражение можно использовать несколько раз в рамках одного запроса. Поддерживается использование как для табличных, так и для скалярных выражений.

```
-- объявление параметра
DECLARE $days AS Int32;
$cutoff = CurrentUtcTimestamp() - $days * Interval("P1D"); -- создание скалярного именованного выражения

-- создание табличного именованного выражения
$base = (
  SELECT *
  FROM raw_events
  WHERE event_ts >= $cutoff -- использование скалярной переменной
);

-- использование именованного выражения
SELECT * FROM $base WHERE event_ts > CurrentUtcTimestamp()
```

YDB гарантирует, что при многократном использовании именованного выражения в пределах одной транзакции будут прочитаны одни и те же данные. Это обеспечивается уровнем изоляции транзакций Serializable.

Отсутствие поддержки коррелированных подзапросов

Коррелированный подзапрос — это подзапрос, который ссылается на столбцы из внешнего запроса. В YQL такие подзапросы не поддерживаются.
Большинство случаев использования коррелированных подзапросов можно заменить с помощью `JOIN` и агрегатных функций.
Полная матрица поддержки и правила преобразования приведены в разделе Коррелированные подзапросы, EXISTS и NOT EXISTS.

EXISTS

Преобразование коррелированного `EXISTS` → `LEFT SEMI JOIN` или `INNER JOIN` с уникальными ключами справа.

Оригинальный запрос:

```
SELECT a.* FROM A a WHERE EXISTS (
  SELECT 1 FROM B b WHERE b.key = a.key AND b.flag = 1
);
```

Решение с LEFT SEMI JOIN

```
$B_match = (
  SELECT key
  FROM B
  WHERE flag = 1
);

SELECT a.*
FROM A AS a
LEFT SEMI JOIN $B_match AS b
ON b.key = a.key;
```

Альтернатива с INNER JOIN

Перед соединением удалите повторяющиеся ключи с правой стороны. Тогда несколько подходящих строк не будут дублировать строку слева. Применять `DISTINCT` к колонкам левой стороны не требуется: он может объединить одинаковые строки внешней выборки.

```
$B_match = (
  SELECT key
  FROM B
  WHERE flag = 1
  GROUP BY key
);
```

```
SELECT a.*
FROM A AS a
INNER JOIN $B_match AS b
ON b.key = a.key;
```

Подзапрос с агрегатом

Скалярный подзапрос с агрегатом → агрегирование + JOIN

Оригинальный запрос:

```
SELECT a.*, (SELECT MAX(ts) FROM B b WHERE b.user_id = a.user_id) AS last_ts
FROM A a;
```

Решение

```
$B_last = (
  SELECT user_id, MAX(ts) AS last_ts
  FROM B
  GROUP BY user_id
);

SELECT a.*, bl.last_ts
FROM A AS a
LEFT JOIN $B_last AS bl
ON bl.user_id = a.user_id;
```

NOT EXISTS

Преобразование коррелированного `NOT EXISTS` → `LEFT ONLY JOIN`.

Оригинальный запрос:

```
SELECT a.* FROM A a WHERE NOT EXISTS (
  SELECT 1 FROM B b WHERE b.key = a.key AND b.flag = 1
);
```

Решение

```
$B_match = (
  SELECT key
  FROM B
  WHERE flag = 1
);

SELECT a.*
FROM A AS a
LEFT ONLY JOIN $B_match AS b
ON b.key = a.key;
```

Поддержка только equi-JOIN (соединения по равенству)

Условия в JOIN могут содержать только оператор равенства (=). Соединения по неравенству (>, <, >=, <=, BETWEEN) не поддерживаются. Условие неравенства можно вынести в секцию WHERE после CROSS JOIN.

> **Важно**
>
> CROSS JOIN создает декартово произведение таблиц. Этот подход не рекомендуется для больших таблиц, так как он приводит к взрывному росту объема промежуточных данных и деградации производительности. Используйте его только в том случае, если одна из таблиц очень маленькая или обе таблицы были предварительно отфильтрованы до небольшого размера.

Оригинальный запрос:

```
SELECT e.event_id, e.user_id, e.ts
FROM events AS e
JOIN periods AS p
ON e.user_id = p.user_id
 AND e.ts >= p.start_ts
```

Решение

```
SELECT e.event_id, e.user_id, e.ts
FROM events AS e
CROSS JOIN periods AS p
WHERE e.user_id = p.user_id
  AND e.ts >= p.start_ts
```

Импорт данных с помощью федеративных запросов

YDB поддерживает федеративные запросы к внешним источникам данных (таким как ClickHouse, PostgreSQL и др.). Этот механизм предназначен для быстрой ad-hoc аналитики и объединения данных «на лету», но не является оптимальным инструментом для массовой и регулярной загрузки больших объёмов данных (ETL/ELT). При использовании федеративных запросов для импорта вы можете столкнуться с ограничениями по поддерживаемым типам данных и выполнению запросов.

Решение

1. Выгрузите данные из вашей СУБД в один из открытых форматов (рекомендуется `CSV`) в бакет Object Storage. Используйте команду `INSERT INTO ... SELECT FROM` для чтения данных из внешней таблицы, связанной с вашим бакетом в Object Storage. Этот подход позволяет эффективно распараллелить чтение данных.
2. Для задач непрерывной репликации или построения сложных пайплайнов используйте стандартные инструменты индустрии, которые интегрируются с YDB:
   - Change Data Capture (CDC): инструменты вроде Debezium могут захватывать изменения из транзакционного лога вашей OLTP-базы и доставлять их в YDB.
   - ETL/ELT-фреймворки: системы, такие как Apache Spark или Apache NiFi, имеют коннекторы к YDB и позволяют строить гибкие и мощные пайплайны по обработке и загрузке данных.

Список ограничений:

> **Важно**
>
> Внешние коннекторы являются экспериментальной возможностью YDB. Для работы с внешними СУБД через коннекторы необходимо развернуть fq-connector-go и явно включить соответствующие источники в конфигурации кластера YDB. Функциональность может измениться, поэтому использование в производственной среде не рекомендуется без тщательного тестирования.

- ClickHouse
- Greenplum
- Microsoft SQL Server
- MySQL
- PostgreSQL
- YDB (федеративные запросы между базами YDB)

```
-- Параметры (для примера — как переменные)
DECLARE $yc_key    AS String;
DECLARE $yc_secret AS String;

-- 1) Внешний источник S3 с endpoint Yandex Cloud
CREATE EXTERNAL DATA SOURCE s3_backup_ds
WITH (
  SOURCE_TYPE = "S3",
  LOCATION    = "https://storage.yandexcloud.net",  -- endpoint YC
  AUTH_METHOD = "AWS",
  AWS_ACCESS_KEY_ID     = $yc_key,
  AWS_SECRET_ACCESS_KEY = $yc_secret
);

-- 2) Внешняя "таблица" (папка с CSV в бакете)
CREATE EXTERNAL TABLE s3_my_columnstore_table_backup
WITH (
  DATA_SOURCE = "s3_backup_ds",
  LOCATION    = "s3://my-bucket/ydb-backups/processed_data/full/",  -- путь в Object Storage
  FORMAT      = "CSV"
)(
  id Uint64,
  event_dt Datetime,
  value String
  -- ... остальные колонки вашей CS-таблицы ...
);

-- 3) Экспорт
INSERT INTO s3_my_columnstore_table_backup
SELECT * FROM my_columnstore_table;

-- 4) Восстановление данных
INSERT INTO my_columnstore_table
SELECT *
FROM s3_my_columnstore_table_backup;
```

Число партиций фиксируется при создании

В YDB число партиций (сегментов данных) для колоночной таблицы задаётся один раз при её создании и не может быть изменено в дальнейшем.

Правильный выбор числа партиций — важный аспект проектирования схемы данных.

- Слишком мало партиций: может привести к неравномерной загрузке вычислительных узлов (hotspots) и ограничить параллелизм выполнения запросов.
- Слишком много партиций: может создать избыточную нагрузку на управляющий компонент базы данных (Scheme Shard) и увеличить накладные расходы на обработку запросов.

Решение

- Начальное количество партиций: для базовой оценки числа партиций можно использовать формулу `(количество узлов * 4)`. Это позволит максимально утилизировать ресурсы кластера при выполнении параллельных запросов.
- Выбирайте число партиций с учётом ожидаемого роста объёма данных и увеличения количества узлов в кластере.
- Суммарное количество партиций во всех таблицах одной базы данных не должно превышать **2000**.
- Если необходимо увеличить число партиций, можно создать новую таблицу и перенести в неё данные с помощью запроса:

```
CREATE TABLE (PRIMARY KEY (a, b)) PARTITION BY HASH(a)
WITH(STORE=COLUMN, PARTITION_COUNT=96) new_table AS SELECT * FROM old_table;
```

Отсутствуют вторичные индексы и скип-индексы

Производительность аналитических запросов при колоночном хранении данных достигается за счёт механизмов, основанных на физической организации данных: колоночное хранение, партиционирование, сортировка по первичному ключу.

Решение

Уделите особое внимание ключам партиционирования и первичным ключам, как описано в разделе Выбор ключей для максимальной производительности колоночных таблиц.

Не рекомендуется смешивать OLTP и OLAP в одной базе

Нагрузки OLTP и OLAP предъявляют противоположные требования к ресурсам, и их смешивание почти всегда приводит к взаимной деградации производительности.

- OLTP-нагрузка (строковые таблицы) — это множество коротких, быстрых транзакций (чтение/запись по ключу), критичных к задержкам (latency).
- OLAP-нагрузка (колоночные таблицы) — как правило, длительные и ресурсоёмкие запросы, сканирующие большие объёмы данных, выполняющие агрегацию и активно потребляющие CPU, память и дисковый ввод-вывод.

Когда тяжёлый OLAP-запрос начинает выполняться, он может монополизировать ресурсы кластера, из-за чего быстрые OLTP-запросы встают в очередь, и их время отклика увеличивается.

Не рекомендуется совмещать OLTP- и OLAP-нагрузки в одной базе данных. OLAP-нагрузки могут негативно влиять на OLTP, увеличивая время выполнения запросов.

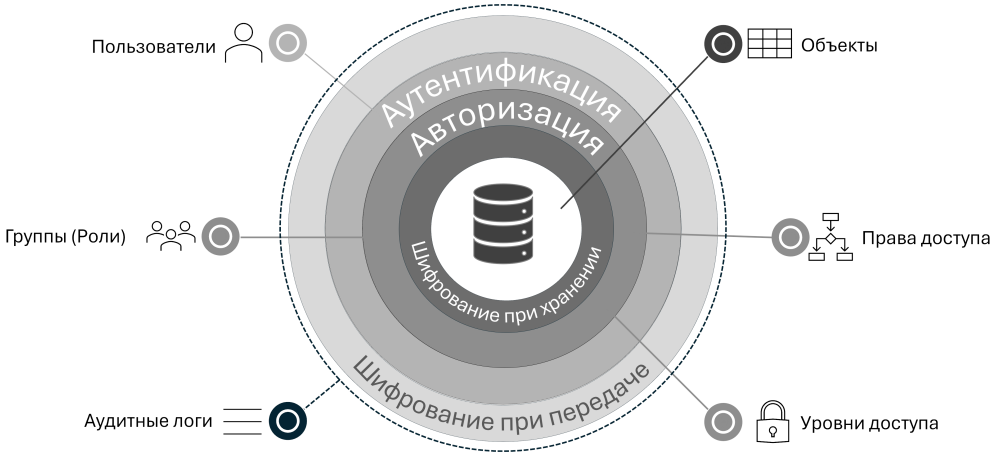
Решение

Для обеспечения стабильной и предсказуемой производительности обоих контуров используйте две отдельные базы данных YDB: одну для OLTP, другую для OLAP. Для синхронизации данных между ними используйте механизм Change Data Capture (CDC) и сервис Трансфер данных, который позволяет в потоковом режиме доставлять изменения из OLTP-базы в OLAP-базу.

YDB для инженеров по безопасности

В этом разделе документации YDB рассматриваются аспекты работы с YDB, связанные с безопасностью. Также он будет полезен для целей обеспечения compliance.

Основные элементы и концепции безопасности YDB



Система безопасности YDB оперирует следующими понятиями:

- **Субъекты доступа**:
  - **Пользователи**. YDB позволяет работать как с локальными пользователями, так и с пользователями из внешних источников (LDAP-каталоги, IAM-системы и пр).
  - **Группы**. YDB позволяет объединять пользователей в именованные множества. Состав пользователей в группе можно изменять в дальнейшем. Группа может быть и пустой — не содержать ни одного пользователя.
- **Объекты доступа**. В YDB объектами доступа являются объекты схемы (таблицы, системные представления и т.п.), для которых настраиваются права доступа.
- **Права доступа**. С помощью **прав доступа** в YDB определяется перечень допустимых операций с объектами доступа для конкретного пользователей или группы.

  Права доступа — это возможность выполнять определенные действия (создание, удаление, чтение, обновление и т.п.) над объектом доступа.

  Права доступа могут быть выданы конкретному пользователю, или группе. Пользователь, которого включили в группу, наделяется правами, выданными ранее этой группе, на время нахождения в этой группе.

  Подробнее о правах доступа см. в разделе Права доступа.
- **Уровни доступа**. С помощью **уровней доступа** в YDB определяется доступ к дополнительным возможностям по управлению кластером для конкретного пользователя или группы. YDB использует иерархические уровни доступа: database, viewer, monitoring и administration. Более высокие уровни автоматически включают все более низкие. Дополнительно есть два специальных неиерархических списка: bootstrap (для начальной инициализации кластера) и register node (для регистрации динамических узлов).

  Подробнее об уровнях доступа см. в разделе Списки уровней доступа.
- **Аутентификация и авторизация**. Система управления доступом в YDB предоставляет механизм защиты данных в кластере YDB. Только аутентифицированные субъекты доступа (пользователи и группы) могут работать с данными, а доступ к данным может ограничиваться.
  - **Аутентификация устройств**. При открытии TLS-соединения YDB может проверять сертификат клиента и тем самым ограничивать сетевой периметр. Такая аутентификация опциональна и настраивается для поддерживаемых интерфейсов. После прохождения аутентификации устройств для доступа к данным по-прежнему может требоваться аутентификация пользователя.
  - **Аутентификация клиентов**. При доступе к кластеру YDB пользователи проходят аутентификацию — процесс проверки, подтверждающий подлинность пользователя. YDB поддерживает различные механизмы аутентификации, их детальное описание можно найти в соответствующем разделе аутентификации.

    Важно отметить,что независимо от используемого механизма, в результате успешной аутентификации пользователи получают идентификатор (SID) и аутентификационный токен.

    - Идентификатор в виде SID используется для идентификации пользователя в YDB. Например, в качестве SID для локальных пользователей выступает login пользователя. Для внешних пользователей SID также содержит информацию об источнике пользователя. SID пользователя также можно встретить в системных представлениях, описывающих текущие настройки безопасности.
    - Аутентификационный токен используется узлами YDB для авторизации доступа пользователя перед обработкой его запросов.

      Полученный аутентификационный токен пользователь может многократно использовать в дальнейшем в своей работе при выполнении запросов к кластеру YDB. Более подробную информацию про аутентификационный токен, например о параметрах кеширования и т.п., см. в разделе auth_config.
  - **Авторизация**. На основе аутентификационных данных проводится авторизация — процесс проверки наличия определённых прав доступа и уровней доступа для выполнения пользователем запрошенной операции.
- **Аудитные логи**. Действия, направленные на изменение настроек безопасности, дополнительно логируются в отдельный журнал, называемый аудитный лог. Данный журнал, в первую очередь, будет интересен для тех, кто отвечает за безопасность информации. В аудитный лог попадают такие действия как создание или удаление объектов доступа, создание или удаление пользователей, смена паролей, выдача или отзыв прав доступа и т.п.
- **Шифрование**. YDB является распределённой системой, обычно работающей на кластерах, расположенных в нескольких датацентрах. Для защиты пользовательских данных YDB предоставляет механизмы:
  - шифрования данных при передаче для обеспечения безопасности данных, передаваемых между клиентом и YDB, между узлами самого кластера YDB;
  - шифрования данных при хранении.

Аутентификация

После успешной установки сетевого соединения сервер принимает к обработке запросы от клиента. Аутентификационная информация передаётся в виде аутентификационного токена или клиентского сертификата. На её основании сервер определяет SID клиента и проверяет его доступы на выполнение запроса.

> **Примечание**
>
> Под клиентом аутентификации понимается пользователь, проходящий процедуру проверки подлинности при доступе к YDB. Примерами клиентов являются приложения, использующие SDK или CLI.

Поддерживаются следующие виды аутентификации:

- Анонимная аутентификация.
- Аутентификация по логину и паролю.
- Аутентификация с использованием LDAP-каталога.
- Аутентификация с использованием внешнего провайдера идентификации по протоколу OpenID Connect.
- Аутентификация по клиентскому сертификату.
- Аутентификация с использованием стороннего IAM-провайдера, например Yandex Identity and Access Management.

Анонимная аутентификация

По умолчанию, YDB допускает выполнение запросов без указания аутентификационных данных, таких как имя пользователя или токен. Проверка прав доступа (авторизация) при этом также не проводится.

> **Важно**
>
> Использовать анонимную аутентификацию следует только в ознакомительных целях для локальных баз данных, не имеющих доступа по сети.

За выключение режима анонимной аутентификации отвечает флаг `enforce_user_token_requirement` в настройках режима аутентификации YDB.

В зависимости от настроек режима аутентификации, реальная аутентификация может быть не анонимной:

- Отсутствующий в запросах токен может подменяться на токен по умолчанию
- Явно указанный в запросах токен может проверяться по надлежащим правилам

Тогда запросы будут выполняться не анонимно, и будет также выполняться проверка прав.

В зависимости от настроек уровней доступа, анонимные запросы могут в том числе выполнять в системе действия, требующие административного уровня доступа.

Аутентификация по логину и паролю

Аутентификация по логину и паролю через сервер YDB доступна только для локальных пользователей. В аутентификации внешних пользователей участвуют сервера внешних систем.

Данный вид доступа подразумевает наличие у каждого пользователя базы данных логина и пароля.
Логин пользователя может содержать только строчные буквы латинского алфавита, цифры и символ `@`.
Допускается установление различных критериев на сложность пароля.

Логин пользователя и хеш пароля хранятся в таблице внутри компонента аутентификации. Пароль хеширован методом Argon2. Доступ к этой таблице имеет только администратор системы.

В ответ на логин и пароль возвращается аутентификационный токен. Время жизни токена по умолчанию составляет 12 часов. Для ротации токенов клиент, например, SDK, самостоятельно обращается к сервису аутентификации. Использование токена ускоряет процесс аутентификации и повышает безопасность.

Процесс аутентификации по логину и паролю включает следующие шаги:

1. Клиент обращается к базе данных и передаёт логин и пароль пользователя сервису аутентификации YDB.
2. Сервис проверяет аутентификационные данные, при успешном сопоставлении создаёт токен и возвращает его клиенту.
3. Клиент обращается к БД, передавая в качестве аутентификационной информации токен.

Для включения аутентификации по логину и паролю, необходимо убедиться, что параметры `use_login_provider` и `enable_login_authentication` выставлены в значение по умолчанию `true` в конфигурации. Кроме того, чтобы отключить анонимную аутентификацию, необходимо выставить значение параметра `enforce_user_token_requirement` в `true`.

Об управлении ролями и пользователями читайте в Авторизация.

Сложность пароля

YDB позволяет настраивать требования к сложности паролей. Если пароль, предоставляемый через команды `CREATE USER` или `ALTER USER`, не соответствует критериям сложности, выполнение команды завершится ошибкой.
По умолчанию на пароли не накладываются никакие ограничения: принимается пароль любой длины, включая пустую строку; в пароле могут присутствовать произвольное количество цифр и букв в любом регистре, а также специальные символы из списка `!@#$%^&*()_+{}|<>?=`. Чтобы задать ограничения на сложность пароля, необходимо заполнить секцию `password_complexity` в конфигурации.

Принудительная блокировка/разблокировка пользователя

Существует ещё один способ запретить пользователю выполнять аутентификацию — принудительная блокировка администратором кластера или базы данных. Администраторы могут снимать блокировку как с пользователей, заблокированных принудительно, так и с пользователей, оказавшихся заблокированными из-за превышения лимита на количество попыток ввода неправильного пароля. Подробную информацию о принудительной блокировке и разблокировке пользователей можно найти в описании команды `ALTER USER LOGIN/NOLOGIN`.

Защита от перебора пароля

YDB предоставляет защиту от перебора пароля пользователем. Пользователь будет считаться заблокированным, если превысит число попыток ввода неправильного пароля. По истечении указанного времени он снова получит возможность аутентифицироваться.

> **Примечание**
>
> Данный механизм применим только для пользователей, которые обслуживаются самой YDB, для, так называемых, встроенных пользователей. Пользователи, обслуживаемые внешними источниками аутентификации, такими как LDAP серверы, не подпадают под действие механизма защиты от перебора пароля.

По умолчанию пользователю даётся 4 попытки ввести правильный пароль. В противном случае аутентификация для него будет запрещена в течение одного часа. Настроить критерии блокировки пользователя можно в конфигурации.

При необходимости администратор кластера или базы данных может досрочно разблокировать пользователя.

Информацию о статусе блокировки пользователя и количества попыток неправильного ввода пароля можно узнать из системного представления пользователя.

Аутентификация с использованием LDAP-каталога

В YDB интегрировано взаимодействие с LDAP каталогом. LDAP каталог является внешним по отношению к YDB сервисом и используется для аутентификации и авторизации пользователей базы данных. Прежде чем воспользоваться данным способом аутентификации и авторизации необходимо иметь развернутый LDAP сервис и настроенный сетевой доступ между ним и серверами YDB.

Примеры поддерживаемых реализаций LDAP каталогов: OpenLdap, Active Directory.

Аутентификация

Аутентификация с помощью LDAP протокола похожа на процесс аутентификации по логину и паролю. Отличие заключается лишь в том, что роль компонента аутентификации играет LDAP каталог. LDAP каталог используется для проверки пары логин/пароль и для определения групп, к которым принадлежит пользователь.

> **Примечание**
>
> Так как LDAP каталог является внешним независимым сервисом, YDB не имеет возможности управлять учетными записями пользователей в каталоге. Для успешной аутентификации пользователь уже должен быть создан в LDAP каталоге. Использование команд `CREATE USER`, `CREATE GROUP`, `ALTER USER`, `ALTER GROUP`, `DROP USER`, `DROP GROUP` не окажет влияния на список пользователей и групп в каталоге. Информацию об управлении учетными записями необходимо искать в документации используемого LDAP каталога.

На данный момент YDB поддерживает только один способ LDAP-аутентификации пользователей — *search+bind*: после получения логина и пароля выполняется служебный *bind* от имени сервисного аккаунта (логин и пароль в `bind_dn` / `bind_password` либо сертификат и SASL EXTERNAL — см. конфигурацию), затем поиск записи пользователя и второй *bind* уже от его имени.

> **Примечание**
>
> Под сервисным аккаунтом понимается отдельная учётная запись в каталоге LDAP, которую используют приложения/сервисы для подключения к LDAP и выполнения нужных операций.

Учетные данные сервисного аккаунта для подключения к LDAP указываются в настройках конфигурации: используйте параметры `bind_dn` и `bind_password`, либо настройте mTLS (подробнее см. в разделе Аутентификация сервисного аккаунта).

Далее процесс аутентификации проходит по следующей схеме:

1. YDB подключается к LDAP от имени сервисного аккаунта.
2. После успешного подключения выполняется поиск пользователя, который пытается пройти аутентификацию. Поиск идет по всему поддереву, указанному в параметре конфигурации `base_dn`, и по фильтру, заданному в параметре `search_filter`.
3. Если пользователь найден, YDB выполняет повторную операцию bind — уже от имени найденного пользователя, с использованием его пароля.
4. Итоговый результат — успешная или неуспешная аутентификация — определяется результатом второго bind (от лица пользователя).

Таким образом, YDB не хранит пароли пользователей и полностью полагается на механизм аутентификации LDAP.

В результате успешной проверки логина и пароля пользователя в LDAP каталоге возвращается аутентификационный токен YDB. Далее этот токен используется вместо логина и пароля. Оперирование токеном ускоряет процесс аутентификации и повышает безопасность.

> **Примечание**
>
> При использовании LDAP-аутентификации, никакие пароли пользователей в YDB не хранятся.

Аутентификация сервисного аккаунта

Сервисный аккаунт можно аутентифицировать двумя основными способами:

- С помощью логина и пароля.
  В этом случае в конфигурации нужно указать логин (`bind_dn`) и пароль (`bind_password`). Эти параметры будут использоваться для подключения к LDAP-серверу от имени сервисного аккаунта.
- С помощью mTLS (mutual TLS) через механизм SASL EXTERNAL.
  В этом варианте вместо логина и пароля для аутентификации используются сертификаты. Это позволяет не хранить

пароль сервисного аккаунта в конфигурации — достаточно указать файлы сертификата (`use_tls.cert_file`) и закрытого ключа (`use_tls.key_file`), а также включить специальный флаг (`extended_settings.enable_sasl_external_bind`). Подробную информацию по настройке см. в разделе ldap_authentication.

Проверка токена

После аутентификации пользователя формируется аутентификационный токен, который передаётся с запросами к YDB. При проверке токена узел определяет пользователя, от имени которого выполняется запрос, и группы, в которые он входит. В зависимости от способа аутентификации может выполняться криптографическая проверка токена на стороне YDB или обращение к внешней системе аутентификации. Например, для пользователя из LDAP-каталога токен не содержит информацию о группах, поэтому узел выполняет ещё один запрос к LDAP-серверу для получения списка групп пользователя. Сетевые обращения увеличивают время проверки аутентификационного токена и обработки запроса, а также нагрузку на внешнюю систему. Поэтому узлы YDB кешируют результаты проверки.

Группы, как и сам пользователь, являются субъектами выполнения операций над объектами схемы базы данных. Для разграничения доступа к различным ресурсам базы данных субъектам могут назначаться права доступа. И в соответствии со списком назначенных прав субъекты будут авторизованы на выполнение тех или иных операций.

Процесс получения списка групп пользователя из LDAP каталога похож на те действия, которые выполняются при аутентификации. Сначала выполняется операция *bind* для сервисного пользователя, учетные данные которого записаны в параметрах `bind_dn` и `bind_password` секции ldap_authentication файла конфигурации. После успешной аутентификации выполняется поиск записи пользователя, для которого был ранее сформирован токен. Поиск также выполняется в соответствии с параметром `search_filter` . Если пользователь всё ещё существует, то в качестве возвращаемого результата операции *search* будет список значений атрибута, записанного в параметре `requested_group_attribute` . В случае, если этот параметр пуст, то аттрибутом обратного членства в группе будет являться `memberOf` . Атрибут `memberOf` хранит уникальные имена (Distinguished Name, DN) групп, в которых состоит пользователь.

Получение групп

По умолчанию YDB выполняет поиск только тех групп, в которых пользователь состоит непосредственно. С помощью включения флага `extended_settings.enable_nested_groups_search` секции ldap_authentication YDB будет пытаться получить группы на всех уровнях вложенности, а не только те, в которые пользователь входит напрямую. Если YDB настроен для работы с Active Directory, для поиска всех вложенных групп будет использовано специфичное для Active Directory правило соответствия LDAP_MATCHING_RULE_IN_CHAIN. Это правило позволяет получить все вложенные группы одним запросом. Для LDAP-серверов на основе OpenLDAP поиск групп будет выполнен рекурсивным обходом графа, что в общем случае требует выполнения нескольких запросов. Как для Active Directory, так и для OpenLDAP поиск групп будет выполнен только для поддерева, корень которого берётся из параметра конфигурации `base_dn` .

Примечание

В текущей реализации имена групп, которыми будет оперировать YDB, совпадают с теми значениями, которые записаны в атрибуте `memberOf` . Они могут иметь большую длину и быть сложночитаемыми.

Пример:

```
cn=Developers,ou=Groups,dc=mycompany,dc=net@ldap
```

Примечание

Частота обновления информации о пользователе и его группах задаётся параметром `auth_config.refresh_time` . Подробнее см. в статье о кешировании результатов аутентификации.

Важно

Нужно заметить, что на данный момент YDB не имеет возможности отслеживать переименования групп, сделанные на стороне LDAP сервера. Тем самым, группа с новым именем не будет обладать теми же правами, которые были у группы с прежним именем.

LDAP-пользователи и LDAP-группы в YDB

Так как YDB позволяет использовать разные способы аутентификации пользователя, то, оперируя именами пользователей и групп, часто бывает полезно различать, где именно аутентифицировался пользователь. Для всех видов аутентификации, кроме аутентификации по логину и паролю, имена групп и пользователей дополняются суффиксом вида `@<auth-domain>` .

Для LDAP пользователей *auth-domain* задается в параметре конфигурации `ldap_authentication_domain` . По умолчанию имеет значение `ldap` , поэтому все имена пользователей, аутентифицированных с помощью LDAP каталога, и имена групп, в которых они состоят, в YDB будут иметь следующий вид:

- `user1@ldap`
- `group1@ldap`
- `group2@ldap`

Важно

Чтобы различать, что введенный логин должен быть логином пользователя из LDAP каталога, а не логином локального пользователя YDB, к нему нужно добавлять суффикс `@ldap` .

Ниже приведены примеры аутентификации пользователя `user1` используя YDB CLI:

- Аутентификация пользователя из LDAP каталога: `ydb --user user1@ldap -p ydb_profile scheme ls`
- Аутентификация пользователя внутренним механизмом YDB: `ydb --user user1 -p ydb_profile scheme ls`

TLS соединение

В зависимости от указанных параметров конфигурации YDB может устанавливать либо зашифрованное либо незашифрованное соединение. Зашифрованное соединение с LDAP сервером устанавливается с использованием протокола TLS. Такой способ рекомендуется использовать для production кластеров. Существует два способа включить TLS соединение:

- Автоматически. Используется схема соединения `ldaps`
- Использование расширения LDAP протокола `StartTls`

При использовании незашифрованного соединения, все данные, которые передаются в запросах к LDAP серверу, будут передаваться в открытом виде, в том числе и пароли. Таким способом соединения проще начать пользоваться, он больше подходит для экспериментов или тестирования.

LDAPS

Для того чтобы YDB автоматически установила зашифрованное соединение с LDAP-сервером, необходимо в параметре конфигурации **scheme** установить значение `ldaps`. TLS-диалог будет инициирован на порту, указанном в конфигурации. Если порт не указан, для схемы `ldaps` будет применён порт по умолчанию 636. LDAP-сервер должен быть настроен на приём TLS-соединений на указанных портах.

Расширение LDAP протокола `StartTls`

`StartTls` — это расширение протокола LDAP, используемое для шифрования сообщений по протоколу TLS. Оно позволяет передавать в рамках одного соединения с LDAP-сервером одни сообщения в зашифрованном виде, а другие — открыто. Сообщение с этим расширением отправляется от YDB к LDAP-серверу для инициирования TLS-соединения. В случае YDB не предусмотрено включение и выключение TLS-соединения в рамках одного соединения. Поэтому при использовании расширения `StartTls` после установления зашифрованного соединения YDB будет отправлять все дальнейшие сообщения к LDAP-серверу в зашифрованном виде. Одним из преимуществ использования данного расширения вместо схемы `ldaps` (при соответствующей настройке LDAP-сервера) является возможность установить TLS-соединение на нешифрованном порту. Расширение включается в секции `use_tls` файла конфигурации.

Аутентификация с использованием внешнего IdP по протоколу OpenID Connect

YDB может аутентифицировать пользователей по JWT-токенам, выпущенным внешним провайдером идентификации (Identity Provider, IdP), который поддерживает протокол OpenID Connect (OIDC). Провайдер отвечает за аутентификацию пользователя и выдачу токена, а YDB проверяет подпись и утверждения (claims) о субъекте и условиях действия токена и формирует SID пользователя и его групп.

Получение и обновление JWT-токена выполняются на стороне клиента и IdP. Сервер YDB не перенаправляет пользователя на страницу входа IdP и не обменивает `authorization code` на токены. Клиент передаёт уже полученный JWT-токен как Bearer-токен при каждом запросе.

> **Примечание**
>
> Единый вход (Single Sign-On, SSO) через внешний IdP в веб-интерфейс доступен через YDB Enterprise Manager. Подробнее о настройке — в статье Настройка SSO в YDB EM.

Принцип работы

1. Клиент проходит аутентификацию во внешнем IdP и получает подписанный JWT-токен.
2. Клиент передаёт токен в запросе к YDB с типом `Bearer`.
3. Узел YDB запрашивает OIDC Discovery-документ по адресу `<issuer>/.well-known/openid-configuration`, где `<issuer>` — настроенный URL провайдера и идентификатор издателя токенов. Например, для `issuer: https://idp.example.com` запрос отправляется по адресу `https://idp.example.com/.well-known/openid-configuration`. Значение поля `issuer` в документе должно в точности совпадать с настроенным адресом провайдера.
4. Из поля `jwks_uri` Discovery-документа извлекается URL, по которому YDB периодически получает набор открытых ключей JWKS.
5. По полям `alg` (algorithm, алгоритм подписи) и `kid` (key ID, идентификатор ключа) заголовка JWT выбирается открытый ключ. YDB проверяет подпись токена, издателя, получателя и сроки действия.
6. Из полей токена извлекаются идентификатор пользователя и список групп. К ним добавляется настроенный суффикс домена аутентификации, после чего полученные SID используются для авторизации.

Discovery URL, `issuer` и `jwks_uri` должны использовать схему `https://`. Если Discovery-документ или JWKS временно недоступны, YDB повторяет запросы с увеличивающимся интервалом. Полученные ключи кешируются и периодически обновляются для поддержки ротации ключей на стороне IdP. После истечения времени жизни кеша устаревшие ключи удаляются; до успешного обновления JWKS новые проверки токенов завершаются временной ошибкой.

Требования к токену и ключам

JWT-токен должен иметь корректный компактный формат сериализации и содержать в заголовке поля `alg` и `kid`. Поддерживаются только асимметричные алгоритмы подписи:

- RSA PKCS#1: `RS256`, `RS384`, `RS512`;
- RSA-PSS: `PS256`, `PS384`, `PS512`;
- ECDSA: `ES256`, `ES384`, `ES512`.

Симметричные алгоритмы семейства `HS*` не поддерживаются. Открытый ключ в JWKS должен иметь совпадающие `kty` (key type, тип ключа) и `kid` и содержать `x5c` (X.509 certificate chain, цепочку сертификатов X.509). Открытый ключ извлекается из первого сертификата `x5c`; JWK, содержащие только параметры RSA или EC без `x5c`, пропускаются.

Для успешной аутентификации обязательны следующие поля:

- `alg` и `kid` в заголовке JWT;
- утверждение `iss` (issuer, издатель токена), совпадающее со значением `issuer` в конфигурации;
- непустой строковый идентификатор пользователя. Для его получения используется утверждение, заданное параметром `subject_claim_name`. Если это утверждение отсутствует или имеет другой тип, используется стандартное утверждение `sub` (subject, идентификатор субъекта).

Остальные проверяемые поля необязательны:

- утверждение `aud` (audience, получатель токена) проверяется, если в конфигурации задан параметр `audience`; рекомендуется всегда задавать ожидаемую audience, чтобы токены, выпущенные для других сервисов, не принимались;
- при проверке утверждений, связанных со временем, таких как `exp` (expiration time, время истечения срока действия), `nbf` (not before, время начала действия) и `iat` (issued at, время выпуска), учитывается допустимое расхождение часов, заданное параметром `allowed_clock_skew`. Если `exp` отсутствует, YDB устанавливает для результата аутентификации срок действия 10 минут;
- утверждение, заданное параметром `groups_claim_name`, содержит список групп пользователя. Оно должно быть массивом; из массива извлекаются только строковые элементы. Если утверждение отсутствует или имеет другой тип, список групп считается пустым.

Формирование SID

К идентификатору пользователя и каждой группе из JWT добавляется суффикс `@<auth-domain>`. Значение `<auth-domain>` задаётся параметром `external_idp_authentication_domain`; по умолчанию используется значение `sso`.

Например, при `sub: user1`, `groups: [admins, developers]` и домене по умолчанию будут сформированы следующие SID:

- пользователь `user1@sso`;
- группы `admins@sso` и `developers@sso`.

YDB использует список групп из токена без дополнительного обращения к IdP и без раскрытия вложенных групп. Управлять пользователями и группами внешнего IdP с помощью команд `CREATE USER`, `ALTER USER`, `CREATE GROUP` и `ALTER GROUP` нельзя. Права назначаются сформированным SID средствами, описанными в разделе Авторизация.

Настройка сервера

Аутентификация через внешний IdP включается в конфигурации аутентификации при наличии секции `external_idp_config`.

Аутентификация клиентов по сертификату

YDB может аутентифицировать клиента по данным клиентского сертификата, полученного при установлении TLS-соединения. Проверка выполняется на уровне протокола приложения (gRPC и др.), когда сервер уже принимает запросы по открытому соединению.

Такой способ подходит, например, в корпоративных сценариях с централизованной выдачей сертификатов.

Принцип работы

1. Клиент устанавливает TLS-соединение с сервером YDB, передавая клиентский сертификат (и цепочку доверия).
2. При обработке запроса сервер извлекает сертификат из TLS-контекста.
3. Сервер использует сертификат для аутентификации и проверяет его по правилам секции client_certificate_authorization.
4. В результате успешной проверки сертификата клиенту назначается идентификатор защиты SID, который обладает всеми правами, назначенными соответствующему идентификатору.

Аутентификация по сертификату применяется только для запросов без аутентификационного токена. Если клиент передаёт аутентификационный токен — например, в заголовке `Authorization` для HTTP или через механизмы SDK/CLI для IAM, логина и пароля, — приоритет имеет токен. В этом случае сертификат передаётся на уровне TLS, но не используется для аутентификации.

Формирование SID

> **Примечание**
>
> Проверка клиентского сертификата при аутентификации устройств и аутентификации пользователя по клиентскому сертификату — разные механизмы. Аутентификация устройств ограничивает сетевой периметр без формирования SID; аутентификация пользователя по клиентскому сертификату формирует SID и группы для авторизации.

Успешная аутентификация по сертификату создаёт SID пользователя с суффиксом `@<domain>`, где `<domain>` — значение параметра `certificate_authentication_domain` в секции `auth_config` (по умолчанию: `cert`). Имя формируется из всех атрибутов поля Subject сертификата в нотации `Имя=Значение,...@<domain>`. Порядок атрибутов соответствует порядку полей в сертификате. Пример:

```
C=RU,ST=MSK,O=MyOrg,CN=account1.apps.example.net@cert
```

Получение групп

Если в секции client_certificate_authorization заданы блоки `client_certificate_definitions`, сертификат принимается при соответствии хотя бы одному из них. Для каждого подходящего блока клиент включается в группы из `member_groups`. Если `member_groups` не указан, используется группа по умолчанию — `default_group` (значение по умолчанию: `DefaultClientAuth@cert`).

Настройка сервера

Правила проверки сертификатов и назначения групп задаются в секции client_certificate_authorization статической конфигурации кластера. Для включения запросов клиентского сертификата при TLS-handshake по gRPCs задайте параметр `request_client_certificate: true`.

Настройка клиента

Подробнее о настройке YDB CLI — в разделе Параметры TLS-соединения.

Аутентификация устройств по сертификату

Аутентификация устройств — проверка клиентского сертификата при установлении TLS-соединения; SID при этом не формируется. Если сертификат предъявлен, проверяется цепочка доверия к CA; недоверенный сертификат приводит к

отклонению соединения до обработки прикладных запросов. Обязательность предъявления сертификата зависит от интерфейса (см. раздел Использование в YDB).

Зачем нужна аутентификация устройств

Аутентификация устройств решает следующие задачи в YDB:

1. Изоляция кластера — ограничить круг хостов и приложений, которые могут установить TLS-соединение с узлами YDB.
2. Защита от ошибок конфигурации — не допустить подключения к чужим кластерам YDB, например, при неверном параметре node-broker динамический узел не подключится к чужому кластеру, тогда как при обычном TLS такое соединение могло бы установиться.
3. Усложнение атак на прикладной уровень — процесс на постороннем хосте без подходящего сертификата не получает доступ к API кластера, даже если сетевой маршрут до порта существует.

После прохождения аутентификации устройств для доступа к данным может требоваться аутентификация пользователя или приложения. Она может осуществляться не только по проверенному клиентскому сертификату, но и с помощью других способов аутентификации в YDB, например, по логину и паролю.

Принцип работы

1. Клиент устанавливает TLS-соединение с сервером YDB и, если интерфейс соединения требует, предъявляет клиентский сертификат.
2. Если сертификат предъявлен, сервер проверяет его на уровне TLS: цепочку доверия к настроенному центру сертификации (CA), срок действия и т. п. Дополнительные правила сопоставления полей сертификата (например, `require_same_issuer`, Subject и SAN) применяются при аутентификации клиентов по сертификату.
3. При успешной проверке соединение открывается; при неуспешной — отклоняется.

Использование в YDB

Аутентификация устройств опциональна и настраивается независимо: механизм может быть включен на одних портах и отключен на других.

- **Interconnect** — при включении TLS в секции interconnect_config Interconnect требует клиентский сертификат.
- **Kafka API** — при включении mTLS требует клиентский сертификат; проверяется только цепочка доверия к CA, соединение без сертификата или с недоверенным сертификатом не устанавливается. Настройка сервера описана в секции kafka_proxy_config, а подключение клиента — в разделе Аутентификация устройств по mTLS.
- **gRPC** и **YDB Monitoring** — можно включить запрос сертификата клиента для аутентификации устройств, а также отдельно включить его обязательную проверку (недоверенный сертификат отклоняется всегда). Настройка gRPC описана в секциях grpc_config и client_certificate_authorization, а подключение клиента — в разделе Параметры TLS-соединения; настройка мониторинга YDB Monitoring описана в секции monitoring_config.

Аутентификация с использованием стороннего IAM-провайдера

- **Access Token** — параметром для клиента (SDK или CLI) задается фиксированный токен, который передается в запросы.
- **Refresh Token** — параметром для клиента (SDK или CLI) задается OAuth токен персональной учетной записи, на основании которого клиент периодически в фоновом режиме обращается к IAM API для ротации (получения следующего) токена, передаваемого в запросы.
- **Service Account Key** — параметром для клиента (SDK или CLI) задаются атрибуты сервисной учетной записи и ключ для подписи, на основании которых клиент периодически в фоновом режиме обращается к IAM API для ротации (получения следующего) токена, передаваемого в запросы.
- **Metadata** — клиент (SDK или CLI) периодически обращается к локальному сервису для ротации (получения следующего) токена, передаваемого в запросы.
- **OAuth 2.0 token exchange** - клиент (SDK или CLI) обменивает токен другого типа на access token по протоколу обмена токенов OAuth 2.0, который затем передаётся в запросы YDB API.

Любой обладатель валидного токена может получить доступ на выполнение операций, поэтому главная задача системы безопасности — обеспечение секретности токена и предотвращение его компрометации.

Режимы аутентификации с ротацией токена **Refresh Token** и **Service Account Key** обеспечивают больший уровень безопасности по сравнению с режимом с фиксированным токеном **Access Token**, так как на сервер YDB по сети передаются только короткоживущие секреты.

Максимальная безопасность и производительность обеспечивается при использовании режима **Metadata**, так как он исключает необходимость работы с секретами при развертывании приложения, а также позволяет обратиться к IAM и закешировать токен заранее, до запуска приложения.

При выборе режима аутентификации среди поддерживаемых сервером и окружением, следует руководствоваться следующими рекомендациями:

- **Anonymous** обычно применяется на самостоятельно разворачиваемых локальных кластерах YDB, недоступных по сети.
- **Access Token** применяется при отсутствии поддержки других режимов на стороне сервера или в настроечных/отладочных целях. Он не требует взаимодействий клиента с IAM. Однако, если IAM поддерживает API для ротации токенов, то обычно выдаваемые таким IAM фиксированные токены имеют короткий срок жизни, что вынуждает регулярно обновлять их в IAM вручную заново.
- **Refresh Token** может применяться при выполнении разовых ручных операций под персональной учетной записью, например, связанных с обслуживанием данных в БД, выполнением ad-hoc операций в CLI, или запусками приложений с рабочей станции. Такой токен можно получить вручную в IAM один раз на долгий срок и сохранить в переменной окружения на личной рабочей станции для автоматического применения при запуске CLI без дополнительных параметров аутентификации.
- **Service Account Key** применяется в первую очередь для приложений, рассчитанных на эксплуатацию в окружениях, где поддерживается режим **Metadata**, при их тестировании вне таких окружений (например, на рабочей станции). Также он может применяться для приложений вне таких окружений, работая как аналог **Refresh Token** для сервисных учетных записей. В отличие от персональной учетной записи, объекты доступа и роли сервисной учетной записи могут быть ограничены.
- **Metadata** применяется при разворачивании приложений в облаках. В настоящее время этот режим поддерживается на виртуальных машинах и в Cloud Functions Yandex.Cloud.

Токен для указания в параметрах может быть получен в системе IAM, с которой связана конкретная установка YDB. В частности, для сервиса YDB в Yandex.Cloud применяется Yandex.Passport OAuth и сервисные аккаунты Yandex.Cloud. При использовании YDB в корпоративных контекстах могут применяться стандартные для данной организации системы централизованной аутентификации.

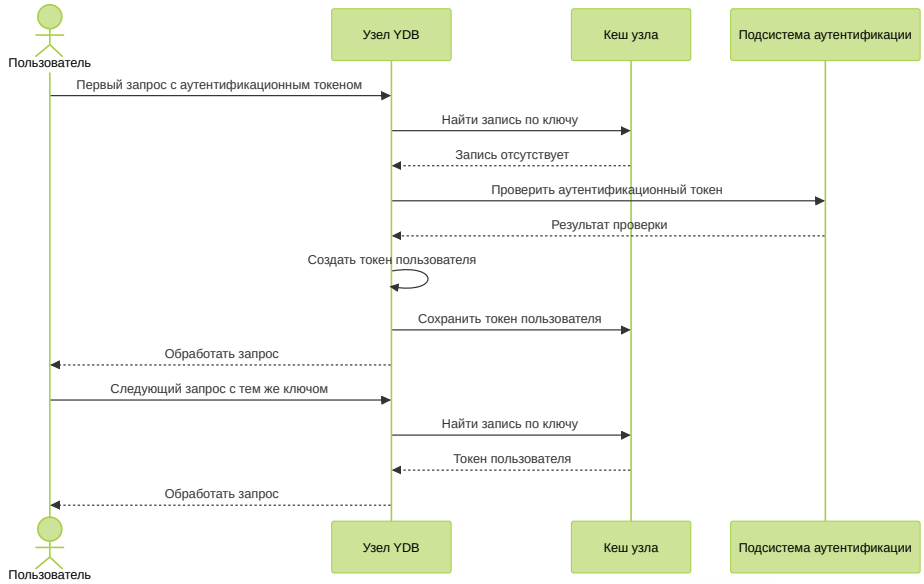
При использовании режимов, предусматривающих обращение клиента YDB к IAM, дополнительно может быть задан URL IAM, предоставляющий API выдачи токенов. По умолчанию в существующих SDK и CLI производится попытка обращения к API IAM Yandex.Cloud, размещенному на `iam.api.cloud.yandex.net:443`.

Кеширование результатов аутентификации

В процессе аутентификации пользовательская сессия получает аутентификационный токен, который передаётся с каждым запросом к кластеру YDB. Поскольку YDB — распределённая система, запросы обрабатываются на разных узлах. Получив запрос, каждый узел независимо проверяет аутентификационный токен и при успешной проверке создаёт токен пользователя.

Аутентификационный токен и токен пользователя — разные сущности. Аутентификационный токен передаётся клиентом. Токен пользователя создаётся внутри узла по результатам проверки, содержит SID пользователя и групп и используется при авторизации. Он хранится только на конкретном узле и не передаётся клиенту.

Узел сохраняет токен пользователя в кеше. Ключ записи включает аутентификационный токен и контекст проверки, например базу данных, атрибуты и запрошенные права доступа. Последующий запрос использует кешированную запись только при совпадении ключа.



Время жизни токена пользователя и другие аспекты кеширования настраиваются в секции `auth_config` конфигурации YDB.

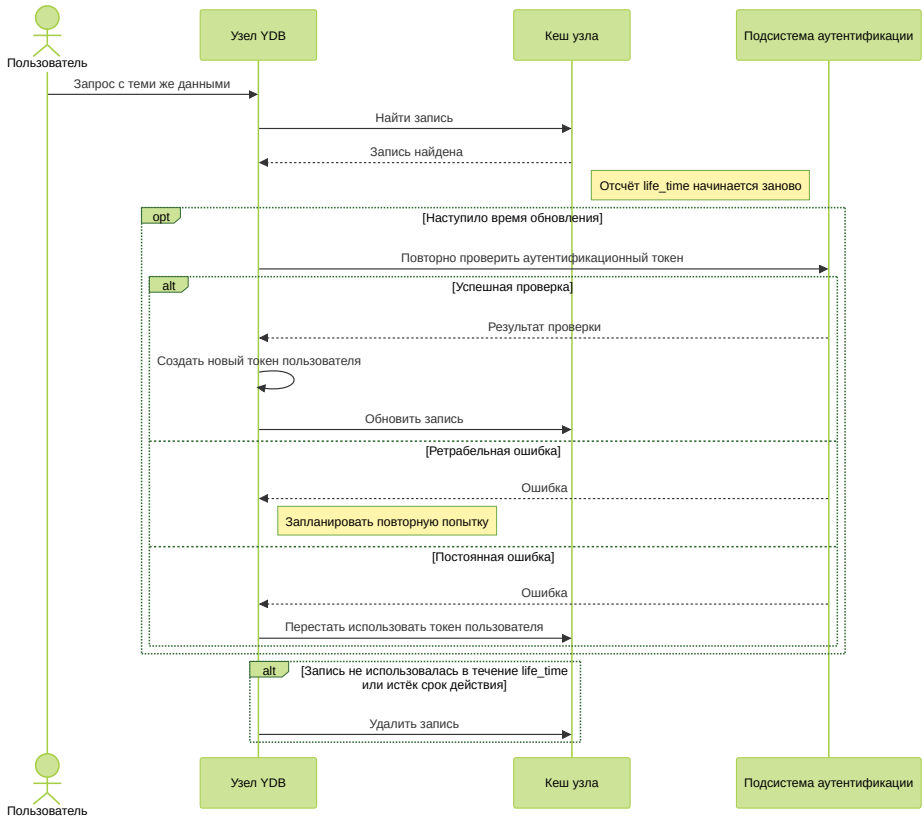
Время жизни записи в кеше

Запись с токеном пользователя удаляется из кеша, если выполняется одно из условий:

- Запись не использовалась в течение `auth_config.life_time`. Время последнего использования записи обновляет только запрос, совпадающий по ключу кеша (аутентификационному токену и контексту проверки).
- Истёк срок действия записи. Для токенов входа по логину и паролю и токенов внешнего провайдера идентификации срок записи определяется сроком действия аутентификационного токена. Для запросов с подписью ключом доступа используется `auth_config.as_signature_expire_time`, для остальных типов — `auth_config.expire_time`.

Ретрабельной считается ошибка, после которой проверку можно повторить без изменения запроса. Тип ошибки определяет соответствующая подсистема аутентификации.

> **Важно**
>
> При ретрабельной ошибке обновления узел продолжает использовать сохранённый токен пользователя. Поэтому удаление аккаунта или изменение принадлежности пользователя к группам может примениться на конкретном узле только после успешного обновления либо удаления записи из кеша.
>
> ACL не кешируются в токене пользователя, поэтому изменения ACL применяются при следующей проверке прав.



Обновление токенов пользователя

Для обновляемых аутентификационных токенов узел периодически повторяет проверку и заменяет токен пользователя в кеше. В зависимости от способа аутентификации проверка выполняется локально или требует обращения к внешней системе. После успешной проверки обновляются SID пользователя и групп.

Параметр `auth_config.refresh_time` задаёт максимальный интервал до следующей проверки. Конкретный момент выбирается в диапазоне от `refresh_time/2` до `refresh_time`.

После ретрабельной ошибки повторные попытки продолжаются до успешного обновления, постоянной ошибки или удаления записи из кеша. Интервалы между попытками задаются параметрами `min_error_refresh_time` и `max_error_refresh_time`. Для токенов входа по логину и паролю и токенов внешнего провайдера идентификации попытки также прекращаются после истечения срока действия аутентификационного токена.

Возможные причины ошибок при проверке и обновлении кешированных результатов включают, но не ограничиваются следующими:

- сетевые ошибки и тайм-ауты при обращении к внешней системе аутентификации;
- недоступность или перегрузка внешней системы аутентификации;
- недоступность локального состояния безопасности, необходимого для проверки;
- перегрузка узла YDB или истечение времени обработки запроса.

Авторизация

Основные понятия

Авторизация в YDB основана на понятиях:

- Объект доступа
- Субъект доступа
- Права доступа
- Список доступов
- Владелец
- Пользователь
- Группа

Независимо от метода аутентификации, авторизация всегда выполняется на серверной стороне YDB на основе хранящейся в ней информации об объектах и правах доступа. Права доступа определяют набор доступных для выполнения операций.

Авторизация выполняется на каждое действие пользователя: его права не кешируются, так как могут быть отозваны или предоставлены в любой момент времени.

Пользователь

Пользователи в YDB могут создаваться в разных источниках:

- локальные пользователи в базах данных YDB;
- внешние пользователи из сторонних служб доступа к каталогам.

Для создания, изменения и удаления локальных пользователей YDB есть команды:

- CREATE USER
- ALTER USER
- DROP USER

Примечание

Область действия команд `CREATE USER`, `ALTER USER`, `DROP USER` не распространяется на внешние каталоги пользователей.
Учитывайте это, если к YDB подключаются пользователи со сторонней аутентификацией (например, LDAP).
Например, команда `CREATE USER` не создаст пользователя в LDAP-каталоге.
Подробнее про взаимодействие YDB с LDAP-каталогом.

Примечание

Отдельно выделяется пользователь `root` с максимальными правами. Он создаётся при первоначальном развёртывании кластера, в ходе которой ему нужно сразу установить пароль. В дальнейшем использование данной учетной записи не рекомендуется и следует завести пользователей с ограниченными правами.

Подробнее про первоначальное развертывание:

- Ansible
- Kubernetes
- Вручную
- Начальная настройка безопасности кластера

SID

YDB позволяет работать с пользователями из разных каталогов и систем, и они отличаются SID с использованием суффикса.

Суффикс `@<auth-domain>` идентифицирует «источник пользователя», внутри которого гарантируется уникальность всех логинов или идентификаторов пользователей. Например, в случае аутентификации LDAP SID'ы пользователей будут `user1@ldap` и `user2@ldap`.

У локальных пользователей пустой auth-domain. Если SID пользователя не содержит суффикса, то имеется в виду локальный пользователь, созданный и существующий непосредственно в кластере YDB.

Группа

Любого пользователя можно включить в ту или иную группу доступа или исключить из неё. Как только пользователь включается в группу, он получает все права на объекты базы данных, которые предоставлялись группе доступа.
С помощью групп доступа YDB можно реализовать бизнес-роли пользовательских приложений, заранее настроив требуемые права доступа на нужные объекты.

Примечание

Группа доступа может быть пустой, когда в неё не входит ни один пользователь.

Группы доступа могут быть вложенными.

Для создания, изменения и удаления групп есть следующие виды YQL запросов:

- CREATE GROUP
- ALTER GROUP
- DROP GROUP

Права доступа

Права доступа в YDB привязаны не к субъекту, а к объекту доступа.

У каждого объекта доступа есть список прав — ACL (Access Control List) — он хранит все предоставленные субъектам доступа (пользователям и группам) права на объект.

По умолчанию, права наследуются от родителей потомкам по дереву объектов доступа.

Для управления правами служат следующие виды YQL запросов:

- GRANT.
- REVOKE.

Для управления правами служат следующие CLI-команды:

- chown
- grant
- revoke
- set
- clear
- clear-inheritance
- set-inheritance

Для просмотра ACL объекта доступа служат следующие CLI-команды:

- describe
- list

Владелец объекта

У каждого объекта доступа есть владелец. Им по умолчанию становится субъект доступа, создавший объект доступа.

> **Примечание**
>
> Для владельца не проверяются списки прав на данный объект доступа.
>
> Он имеет полный набор прав на объект.

Владелец объекта есть в том числе у кластера в целом и каждой базы данных.

Сменить владельца можно с помощью CLI команды `chown`.

Просматривать владельца объекта можно с помощью CLI команды `describe`.

Списки уровней доступа

В дополнение к спискам прав, управляющим доступом к конкретным схемным объектам, YDB использует списки уровней доступа для определения иерархических уровней доступа к общекластерным операциям.

Для операций, в которых одновременно проверяются списки прав и списки уровней доступа, оба механизма применяются совместно: действие доступно только если обе проверки его разрешают, и недоступно, если хотя бы одна проверка не пройдена. Для остальных операций применяется только соответствующий механизм проверки.

Иерархия уровней доступа

Списки уровней доступа образуют иерархию, которая используется в YDB UI, viewer и во многих других общекластерных действиях (порядок от меньших привилегий к большим):

- `database_allowed_sids` (`Database`) - доступ к операциям в контексте конкретной базы;
- `viewer_allowed_sids` (`Viewer`) - доступ к просмотру общекластерного состояния;
- `monitoring_allowed_sids` (`Monitoring`) - доступ к операционным действиям в YDB UI;
- `administration_allowed_sids` (`Administration`) - административные действия с кластером и базами.

Более высокий уровень автоматически включает все более низкие, поэтому субъекту достаточно присутствовать только в одном списке. Например, наличие в `administration_allowed_sids` автоматически даёт привилегии `monitoring`, `viewer` и `database`.
Подробности по каждому уровню — в разделе Описание уровней доступа.

Дополнительно существуют два отдельных списка уровней доступа для специфических операций:

- `bootstrap_allowed_sids` — разрешает операции начальной инициализации кластера;
- `register_dynamic_node_allowed_sids` — разрешает регистрацию узлов в кластере.

Описание уровней доступа

Списки уровней доступа настраиваются в конфигурации безопасности и определяют привилегии для:

- **Database** (наличие в `database_allowed_sids`) — доступ только в контексте конкретной базы данных. Можно открыть YDB UI и работать с данными этой базы, но нельзя выполнять общекластерные запросы (например, просматривать список узлов кластера). Запросы без указания базы запрещены.
- **Viewer** (наличие в `viewer_allowed_sids`) — доступ только на чтение для общекластерного состояния: можно просматривать страницы YDB UI и диагностическую информацию, но нельзя запускать действия, изменяющие состояние системы.
- **Monitoring** (наличие в `monitoring_allowed_sids`) — доступ к операционным действиям в YDB UI, включая действия, которые могут менять состояние системы. Например, запуск резервного копирования, восстановление базы или выполнение YQL-запросов через YDB UI.
- **Administration** (наличие в `administration_allowed_sids`) — даёт право выполнять административные действия с базами данных или кластером. Полный административный доступ к кластеру и его базам данных. Также используется для изменения конфигурации, схемных операций, требующих административных прав, и других административных проверок.
- **Register node** (наличие в `register_dynamic_node_allowed_sids`) — отдельный (неиерархический) уровень для регистрации динамических узлов в кластере. Не даёт автоматически прав

`database` / `viewer` / `monitoring` / `administration`. По техническим причинам, если список задан (не пуст), он должен включать `root@builtin`.

- **Bootstrap** (наличие в `bootstrap_allowed_sids`) — отдельный (неиерархический) уровень только для операций начальной инициализации кластера. Используется в неинициализированном состоянии, когда подсистема аутентификации ещё не функционирует. Начальная инициализация разрешена, если субъект входит в `bootstrap_allowed_sids` или `administration_allowed_sids`, при этом сам по себе `bootstrap` не выдаёт полные административные привилегии.

Начальная настройка безопасности кластера

Начальная настройка безопасности выполняется автоматически при первом запуске кластера YDB.

При выполнении начальной настройки безопасности YDB добавляет в систему суперпользователя, а также реализует набор ролей безопасности для удобного управления пользователями.

> **Примечание**
>
> Информацию об изменении и пропуске начальной настройки безопасности см. в следующих частях:
>
> - Отмена начальной настройки безопасности;
> - Кастомизация начальной настройки безопасности.

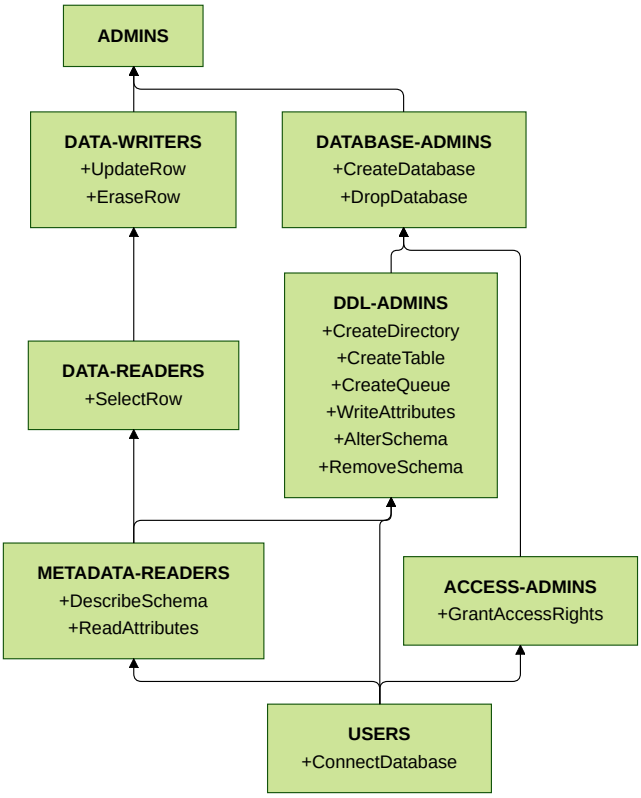
Роли

Роль	Описание
`ADMINS`	Неограниченные права на всю схему кластера.
`DATABASE-ADMINS`	Права на управление базами, схемой и доступами на схеме, без доступа к данным.
`ACCESS-ADMINS`	Права на управление доступами на схеме, без доступа к данным.
`DDL-ADMINS`	Права на управление схемой, без доступа к данным.
`DATA-WRITERS`	Права на доступ к схемным объектам, и на чтение и изменение данных.
`DATA-READERS`	Права на доступ к схемным объектам, и на чтение данных.
`METADATA-READERS`	Права на доступ к схемным объектам, без доступа к данным.
`USERS`	Права на доступ к базам, общая группа всех пользователей.

Группы

Роли реализуются с помощью иерархии пользовательских групп и необходимых наборов прав для этих групп. Возможности групп задаются с помощью прав, выданных на корне схемы кластера.

Группы включаются друг в друга и таким образом по цепочке наследуют соответствующие разрешения:



Например, пользователям группы `DATA-WRITERS` разрешено:

- смотреть схему — `METADATA-READERS`;
- читать данные — `DATA-READERS`;
- и менять их — `DATA-WRITERS`.

Пользователям группы `DDL-ADMINS` разрешено:

- смотреть схему — `METADATA-READERS`;
- и менять её — `DDL-ADMINS`.

Пользователи группы `ADMINS` со схемой и данными могут делать всё.

Суперпользователь

Суперпользователь входит в группы `ADMINS` и `USERS` и обладает полными правами на схему кластера.

По умолчанию суперпользователь называется `root` и имеет пустой пароль.

Группа всех пользователей

Группа `USERS` также определяется как общая группа всех локальных пользователей. При последующем создании локальных пользователей они будут автоматически становиться членами группы `USERS`.

Об управлении группами и пользователями читайте в разделе Авторизация.

Кастомизация начальной настройки безопасности

Вы можете изменить начальную конфигурацию безопасности, добавив собственных пользователей, группы и права доступа.

Для изменения пользователей, групп и прав доступа при начальной настройке безопасности, укажите их в параметрах `default_users`, `default_groups` и `default_access` в секции настроек безопасности в конфигурации кластера (секция `security_config`).

Отмена начальной настройки безопасности

Для отмены начальной настройки безопасности, параметр `security_config.disable_builtin_security` должен быть выставлен в `true`.

Аудитный лог

Аудитный лог — это поток записей, фиксирующих работу кластера YDB. В отличие от технических логов, которые помогают в обнаружении сбоев и устранении неполадок, аудитный лог предоставляет данные значимые для безопасности. Он служит источником информации, отвечая на вопросы: кто что сделал, когда и откуда.

Одна запись в аудитном логе может выглядеть так:

```
{"@timestamp":"2025-11-03T18:07:39.056211Z","@log_type":"audit","operation":"ExecuteQueryRequest","databas
e":"/my_dir/db1","status":"SUCCESS","subject":"serviceaccount@as","remote_address":"ipv6:[xxxx:xxx:xxx:xxx:x:
xxxx:xxx:xxxx]"}
```

Примеры типичных событий аудитного лога:

- доступ к данным через DML-запросы;
- операции управления схемой или конфигурацией;
- изменения прав или настроек контроля доступа;
- административные действия пользователей.

Секция `audit_config` в конфигурации кластера определяет, какие аудит-логи собираются, как они сериализуются и куда доставляются. Подробнее см. в разделе Конфигурация аудитного лога.

Ключевые понятия

Аудитные события

Аудитное событие — это запись в аудитном логе, которая фиксирует одно действие, важное для безопасности. Каждое событие содержит атрибуты, описывающие разные аспекты события. Общие атрибуты перечислены в разделе Общие атрибуты.

Источники аудитных событий

Источник аудитных событий — это служба или подсистема YDB, которая может генерировать аудитные события. Каждый источник имеет уникальный идентификатор (UID) и может предоставлять дополнительные атрибуты, характерные для этого источника. Некоторые источники требуют дополнительной настройки, например включения функциональных флагов, чтобы начать генерировать события. Подробнее см. Обзор источников аудитных событий.

Классы логирования

Аудитные события группируются в *классы логирования*, которые отражают основные категории операций. Вы можете включать или отключать запись для каждого класса в конфигурации и при необходимости настраивать каждый класс отдельно. Доступные классы логирования:

Класс логирования	Описание
`ClusterAdmin`	Запросы на администрирование кластера.
`DatabaseAdmin`	Запросы на администрирование баз данных.
`Login`	Запросы на вход.
`NodeRegistration`	Регистрация нод.
`Ddl`	DDL-запросы.
`Dml`	DML-запросы.
`Operations`	Асинхронные операции RPC, для отслеживания результата которых требуется опрос.
`ExportImport`	Операции экспорта и импорта данных.
`Acl`	Операции с ACL (списками контроля доступа).
`AuditHeartbeat`	Синтетические heartbeat-сообщения, подтверждающие работу аудитного логирования.
`Default`	Настройки по умолчанию для любого компонента без собственной секции конфигурации.

На данный момент не все источники аудитных событий группируют события по классам. Чаще всего для их записи будет достаточно базовой конфигурации. Подробности см. в разделе Обзор источников аудитных событий.

Фазы логирования

Некоторые источники аудитных событий разделяют обработку запроса на этапы. *Фазы логирования* указывают те этапы, на которых создаются события аудитного лога. Указание фаз логирования полезно, когда требуется детальная видимость выполнения запроса и нужно фиксировать события до и после критических этапов обработки. Доступные фазы логирования:

Фаза логирования	Описание
`Received`	Запрос получен, выполнены начальные проверки и аутентификация. Атрибут `status` устанавливается в значение `IN-PROCESS`. Эта фаза отключена по умолчанию; чтобы включить её, добавьте `Received` в `log_class_config.log_phase`.
`Completed`	Запрос полностью завершён. Атрибут `status` принимает значение `SUCCESS` или `ERROR`. Эта фаза включена по умолчанию, если `log_class_config.log_phase` не задан.

Направления аудитного лога

Направление аудитного лога — получатель потока аудитного лога.

На данный момент вы можете настроить следующие направления для аудитного лога:

- файл на каждой ноде кластера YDB;
- агент, который доставляет метрики Unified Agent;
- стандартный поток ошибок `stderr`.

Вы можете использовать любое из перечисленных направлений или их комбинации. Подробнее см. в разделе Конфигурация аудитного лога.

Если поток перенаправлен в файл, доступ к аудитному логу определяется правами файловой системы. Сохранение аудитного лога в файл рекомендуется для продуктивных инсталляций.

Для тестовых инсталляций направляйте аудитный лог в стандартный поток ошибок (`stderr`). Дальнейшая обработка потока зависит от настроек логирования кластера YDB.

Обзор источников аудитных событий

В таблице ниже приведены встроенные источники аудитных событий. Используйте её, чтобы определить, какой источник генерирует нужные события и как их включить.

Источник / UID	Что фиксирует	Требования к настройке
Schemeshard `schemeshard`	Операции со схемой, изменения ACL и действия по управлению пользователями.	Входит в базовую конфигурацию аудита.
gRPC-сервисы `grpc-proxy`	Внешние gRPC-запросы YDB.	Включите соответствующие классы логирования и при необходимости фазы логирования.
gRPC-соединение `grpc-conn`	События подключения и отключения клиентов.	Включите функциональный флаг `enable_grpc_audit`.
gRPC-аутентификация `grpc-login`	Попытки аутентификации в gRPC.	Включите класс `Login` в `log_class_config`.
Сервис мониторинга `monitoring`	HTTP-запросы, обрабатываемые эндпоинтом мониторинга.	Включите класс `ClusterAdmin` в `log_class_config`.
Heartbeat `audit`	Синтетические heartbeat-события, подтверждающие работоспособность аудитного логирования.	Включите класс `AuditHeartbeat` в `log_class_config` и при необходимости настройте параметры heartbeat.
BlobStorage Controller `bsc`	Изменения конфигурации BlobStorage Controller из консоли.	Входит в базовую конфигурацию аудита.
Distconf `distconf`	Обновления распределённой конфигурации.	Входит в базовую конфигурацию аудита.
Web login `web-login`	Взаимодействия с виджетом аутентификации веб-консоли.	Входит в базовую конфигурацию аудита.
Console `console`	Операции жизненного цикла баз данных и изменения динамической конфигурации.	Входит в базовую конфигурацию аудита.

Атрибуты аудитных событий

Атрибуты делятся на две группы:

- общие атрибуты. Присутствуют в большинстве *источников аудитных событий*. Они всегда несут одинаковый смысл;
- атрибуты, специфичные для *источника аудитных событий*.

В этом разделе представлен справочник по атрибутам событий аудитного лога: как общим, так и специфичных для источников. Для каждого источника также указаны его UID, регистрируемые операции и требования к настройке.

Общие атрибуты

В таблице ниже перечислены общие атрибуты.

Атрибут	Описание
`subject`	SID источника события, если обязательная аутентификация включена, иначе `{none}`.
`sanitized_token`	Частично маскированный токен аутентификации, использованный для выполнения запроса. Позволяет связывать события и при этом скрывать исходные учётные данные. Если аутентификация не выполнялась, значение будет `{none}`.
`operation`	Название операции (например, `ALTER DATABASE`, `CREATE TABLE`).
`component`	Уникальный идентификатор *источника аудитных событий*.
`status`	Статус завершения операции. Возможные значения: • `SUCCESS`: операция завершена успешно. • `ERROR`: операция завершилась с ошибкой. • `IN-PROCESS`: операция выполняется.
`reason`	Сообщение об ошибке.

`request_id`	Уникальный идентификатор запроса, вызвавшего операцию. По `request_id` можно отличать события разных операций и связывать их в единый аудитный контекст.
`remote_address`	IP-адрес клиента, отправившего запрос.
`detailed_status`	Статус, который передаёт *источник аудитных событий* YDB.
`database`	Путь базы данных (например, `/my_dir/db`).
`cloud_id`	Идентификатор облака базы данных YDB.
`folder_id`	Идентификатор каталога кластера или базы данных YDB.
`resource_id`	Идентификатор ресурса базы данных YDB.

Schemeshard

UID: `schemeshard`.
Регистрируемые операции: операции со схемой, инициированные DDL-запросами, изменения ACL и действия управления пользователями.
Как включить: требуется только базовая конфигурация аудита.

Таблица ниже перечисляет дополнительные атрибуты, специфичные для источника `Schemeshard`.

Атрибут	Описание
Общие атрибуты Schemeshard	**>**
`tx_id`	Уникальный идентификатор транзакции. Этот идентификатор помогает различать события разных операций. *Обязательный.*
`paths`	Список путей в базе данных, которые изменяет операция (например, `[/my_dir/db/table-a, /my_dir/db/table-b]`). *Необязательный.*
Атрибуты владения и прав доступа	**>**
`new_owner`	SID нового владельца объекта при передаче владения.
`acl_add`	Список добавленных разрешений в краткой записи (например, `[+R:someuser]`).
`acl_remove`	Список отозванных разрешений в краткой записи (например, `[-R:someuser]`).
Пользовательские атрибуты	**>**
`user_attrs_add`	Список пользовательских атрибутов, добавленных при создании объектов или обновлении атрибутов (например, `[attr_name1: A, attr_name2: B]`).
`user_attrs_remove`	Список пользовательских атрибутов, удалённых при создании объектов или обновлении атрибутов (например, `[attr_name1, attr_name2]`).
Атрибуты входа/аутентификации	**>**
`login_user`	Имя пользователя, зафиксированное при операциях входа.
`login_group`	Имя группы, зафиксированное при операциях входа.
`login_member`	Изменения членства.
`login_user_change`	Изменения, применённые к настройкам пользователя.
`login_user_level`	Уровень привилегий пользователя, записанный в событиях аудита. Это поле принимает значение `admin`.
Атрибуты операций импорта/экспорта	**>**
`id`	Уникальный идентификатор операции экспорта или импорта.
`uid`	Заданная пользователем метка операции.
`start_time`	Время начала операции в формате ISO 8601.
`end_time`	Время завершения операции в формате ISO 8601.
`last_login`	Время последнего успешного входа пользователя в формате ISO 8601.
Атрибуты экспорта	**>**
`export_type`	Направление экспорта. Возможные значения: `yt`, `s3`.
`export_item_count`	Количество экспортированных элементов.
`export_yt_prefix`	Префикс пути для экспорта в YT.
`export_s3_bucket`	Имя S3-бакета, используемого для экспорта.

`export_s3_prefix`	Префикс пути для экспорта в S3.
Атрибуты импорта	**>**
`import_type`	Тип источника импорта. Всегда `s3`.
`import_item_count`	Количество импортированных элементов.
`import_s3_bucket`	Имя S3-бакета, используемого для импорта.
`import_s3_prefix`	Префикс источника в S3.

gRPC-сервисы

UID: `grpc-proxy`.
Регистрируемые операции: все внешние (не внутренние) gRPC-запросы.
Как включить: требуется указать классы логирования в конфигурации аудита.
Классы логирования: зависят от типа RPC-запроса: `Ddl`, `Dml`, `Operations`, `ClusterAdmin`, `DatabaseAdmin` и другие классы.
Фазы логирования: `Received`, `Completed`.

Таблица ниже перечисляет дополнительные атрибуты, специфичные для источника `gRPC services`.

Атрибут	Описание
Общие gRPC-атрибуты	**>**
`grpc_method`	Имя RPC-метода. *Необязательный.*
`request`	Санитизированное представление входящего запроса. *Необязательный.*
`start_time`	Время начала операции в формате ISO 8601. *Обязательный.*
`end_time`	Время завершения операции в формате ISO 8601. *Необязательный.*
Атрибуты транзакций	**>**
`tx_id`	Идентификатор транзакции.
`begin_tx`	Флаг со значением `1`, если запрос запускает новую транзакцию.
`commit_tx`	Показывает, завершается ли транзакция этим запросом. Возможные значения: `true`, `false`.
Поля запроса	**>**
`query_text`	Санитизированный текст YQL-запроса.
`prepared_query_id`	Идентификатор подготовленного запроса.
`program_text`	MiniKQL-программа, отправленная с запросом.
`schema_changes`	Описание изменений схемы, запрошенных в операции.
`table`	Полный путь таблицы.
`row_count`	Количество строк, обработанных пакетной вставкой данных.
`tablet_id`	Идентификатор таблетки.

gRPC-соединение

UID: `grpc-conn`.
Регистрируемые операции: изменения состояния соединения (подключение и отключение).
Как включить: активируйте функциональный флаг `enable_grpc_audit`.

Этот источник использует только общие атрибуты.

gRPC-аутентификация

UID: `grpc-login`.
Регистрируемые операции: аутентификация в gRPC.
Как включить: требуется указать классы логирования в конфигурации аудита.
Классы логирования: `Login`.
Фазы логирования: `Completed`.

Таблица ниже перечисляет дополнительные атрибуты, специфичные для источника `gRPC authentication`.

Атрибут	Описание
`login_user`	Имя пользователя. *Обязательный.*
`login_user_level`	Уровень привилегий пользователя, записанный в событиях аудита. Это поле принимает значение `admin`. *Необязательный.*

Сервис мониторинга

UID: `monitoring`.
Регистрируемые операции: HTTP-запросы, обрабатываемые сервисом мониторинга.
Как включить: требуется указать классы логирования в конфигурации аудита.
Классы логирования: `ClusterAdmin`.
Фазы логирования: `Received`, `Completed`.

Таблица ниже перечисляет дополнительные атрибуты, специфичные для источника `Monitoring service`.

Атрибут	Описание
`method`	HTTP-метод запроса, например `POST`, `GET`. *Обязательный.*
`url`	Путь запроса без параметров строки запроса. *Обязательный.*
`params`	Параметры строки запроса в исходном виде. *Необязательный.*
`body`	Тело запроса (усечено до 2 МБ с добавлением суффикса `TRUNCATED_BY_YDB`). *Необязательный.*

Heartbeat

UID: `audit`.
Регистрируемые операции: периодические heartbeat-сообщения аудита.
Как включить: укажите классы логирования в конфигурации аудита.
Классы логирования: `AuditHeartbeat`.
Фазы логирования: `Completed`.

Таблица ниже перечисляет дополнительные атрибуты, специфичные для источника `Heartbeat`.

Атрибут	Описание
`node_id`	Идентификатор ноды, на которой произошло событие. *Обязательный.*

BlobStorage Controller

UID: `bsc`.
Регистрируемые операции: запросы замены конфигурации (`TEvControllerReplaceConfigRequest`), отправляемые консолью.
Как включить: требуется только базовая конфигурация аудита.

Таблица ниже перечисляет дополнительные атрибуты, специфичные для источника `BlobStorage Controller`.

Атрибут	Описание
`old_config`	Снапшот предыдущей конфигурации BlobStorage Controller в формате YAML. *Необязательный.*
`new_config`	Снапшот конфигурации, которая заменила предыдущую. *Необязательный.*

Distconf

UID: `distconf`.
Регистрируемые операции: изменения распределённой конфигурации.
Как включить: требуется только базовая конфигурация аудита.

Таблица ниже перечисляет дополнительные атрибуты, специфичные для источника `Distconf`.

Атрибут	Описание
`old_config`	Снапшот конфигурации, которая была активна до принятия распределённого обновления. Distconf сериализует его в YAML. *Обязательный.*
`new_config`	Снапшот конфигурации, который Distconf зафиксировал после изменения. *Обязательный.*

Web login

UID: `web-login`.
Регистрируемые операции: отслеживает взаимодействия с виджетом аутентификации веб-консоли YDB.
Как включить: требуется только базовая конфигурация аудита.

Этот источник использует только общие атрибуты.

Console

UID: `console`.
Регистрируемые операции: операции жизненного цикла баз данных и изменения динамической конфигурации.
Как включить: требуется только базовая конфигурация аудита.

Таблица ниже перечисляет дополнительные атрибуты, специфичные для источника `Console`.

Атрибут	Описание
`old_config`	Снапшот конфигурации, которая действовала до применения запроса из консоли. *Необязательный.*

`new_config`	Снапшот конфигурации, применённой консолью. *Необязательный.*

Конфигурация аудитного лога

Включение аудитного лога

Аудитное логирование работает на уровне всего кластера. Для *базовой конфигурации* добавьте раздел `audit_config` в конфигурацию кластера и укажите одно или несколько направлений потока (`file_backend` , `unified_agent_backend` , `stderr_backend`):

```
audit_config:
  file_backend:
    format: audit_log_format
    file_path: "path_to_log_file"
  unified_agent_backend:
    format: audit_log_format
    log_name: session_meta_log_name
  stderr_backend:
    format: audit_log_format
```

Параметры audit_config

Все поля являются необязательными.

Ключ	Описание
`stderr_backend`	Перенаправляет аудитный лог в стандартный поток ошибок (`stderr`). Для тестовых инсталляций направляйте аудитный лог в стандартный поток ошибок (`stderr`). Дальнейшая обработка потока зависит от настроек логирования кластера YDB. См. структуру настроек backend.
`file_backend`	Записывает аудитный лог в файл на каждой ноде кластера. Если поток перенаправлен в файл, доступ к аудитному логу определяется правами файловой системы. Сохранение аудитного лога в файл рекомендуется для продуктовых инсталляций. См. структуру настроек backend.
`unified_agent_backend`	Отправляет аудитный лог в Unified Agent. Дополнительно необходимо задать секцию `uaclient_config` в конфигурации кластера. См. структуру настроек backend.
`log_class_config`	Массив правил аудита для разных классов логирования. См. конфигурацию классов логирования.
`heartbeat`	Необязательная конфигурация heartbeat. См. настройки heartbeat.

Конфигурация направлений

Каждое направление поддерживает следующие поля:

Поле	Описание
`format`	Формат аудитного лога. Значение по умолчанию — `JSON` . Подробнее см. раздел Формат логов. *Необязательный.*
`file_path`	Путь к файлу, в который будет записываться аудитный лог. Если путь или файл отсутствуют, они будут созданы на каждой ноде при запуске кластера. Если файл существует, данные будут дозаписываться. Только для `file_backend` . *Обязательный.*
`log_name`	Метаданные сессии, передаваемые вместе с сообщением. По ним можно перенаправлять поток логов в один или несколько дочерних каналов с условием `_log_name: "session_meta_log_name"` . Только для `unified_agent_backend` . *Необязательный.*
`log_json_envelope`	Шаблон JSON, который оборачивает каждую запись лога. Шаблон должен содержать плейсхолдер `%message%` , заменяемый сериализованной аудитной записью. См. раздел Формат конверта. *Необязательный.*

Формат логов

Поле `format` определяет формат сериализации аудитных событий. Поддерживаются следующие форматы:

Формат	Описание
`JSON`	Каждое аудитное событие сериализуется в однострочный JSON-объект с префиксом из временной метки в формате ISO 8601. Пример: `<time>: {"k1": "v1", "k2": "v2", ...}` `k1` , `k2` , …, `kn` *— атрибуты аудитного лога;* `v1` , `v2` , …, `vn` *— их значения.*
`TXT`	Каждое аудитное событие сериализуется в однострочную строку `key=value` с префиксом из временной метки в формате ISO 8601. Пример: `<time>: k1=v1, k2=v2, ...` `k1` , `k2` , …, `kn` *— атрибуты аудитного лога;* `v1` , `v2` , …, `vn` *— их значения.*
`JSON_LOG_COMPATIBLE`	Каждое аудитное событие сериализуется в однострочный JSON-объект, совместимый с целями, совместно используемыми с отладочными логами. Объект содержит поле `@timestamp` с временной меткой ISO 8601 и поле `@log_type` со значением `audit` . Пример: `{"@timestamp": "<ISO 8601 time>", "@log_type": "audit", "k1": "v1", "k2": "v2", ...}` `@timestamp` *хранит временную метку ISO 8601;* `k1` , `k2` , …, `kn` *— атрибуты аудитного лога;* `v1` , `v2` , …, `vn` *— их значения.*

Формат конверта

Backend может оборачивать аудитные события пользовательским конвертом перед отправкой, если указать поле `log_json_envelope`. Шаблон должен содержать плейсхолдер `%message%`, который заменяется сериализованной аудитной записью в выбранном формате.

Например, следующая конфигурация выводит аудитные события в `stderr` в формате JSON, обёрнутые пользовательским конвертом:

```
audit_config:
  stderr_backend:
    format: JSON
    log_json_envelope: '{"audit": %message%, "source": "ydb-audit-log"}'
```

Подробнее о выводе см. в разделе Пример с JSON-конвертом.

Настройка классов логирования

Каждый элемент `log_class_config` поддерживает следующие поля:

Поле	Описание
`log_class`	Имя настраиваемого класса. Использует значения из списка классов логирования. В списке `log_class_config` не должно быть двух классов с одинаковым именем. *Обязательный.*
`enable_logging`	Включает генерацию аудитных событий для выбранного класса. По умолчанию отключено. *Необязательный.*
`exclude_account_type`	Массив типов аккаунтов (`Anonymous`, `User`, `Service`, `ServiceImpersonatedFromUser`), для которых события исключаются, даже если логирование включено. *Необязательный.*
`log_phase`	Массив фаз обработки запроса, которые нужно логировать. См. раздел Фазы логирования. *Необязательный.*

Настройки heartbeat

Heartbeat-события помогают контролировать состояние подсистемы аудитного логирования. Они позволяют настраивать оповещения об отсутствии аудитных событий без ложных срабатываний в периоды без активности.

Параметр `heartbeat.interval_seconds` задаёт частоту записи heartbeat-событий. Значение `0` отключает сообщения heartbeat.

Примеры конфигураций

Простая конфигурация. Ниже приведена простая конфигурация, сохраняющая аудитный лог в текстовом виде в файле с форматом `TXT`.

```
audit_config:
  file_backend:
    format: TXT
    file_path: "/var/log/ydb-audit.log"
```

Расширенная конфигурация. Ниже приведён расширенный пример конфигурации аудитного лога:

- YDB отправляет аудитные логи в Unified Agent в формате `TXT` с меткой `audit`, а также выводит его в `stderr` в формате `JSON`;
- настройки `Default` включают логирование всех классов в фазе `Completed`;
- дополнительно `ClusterAdmin` настроен для логирования фазы `Received`, а `DatabaseAdmin` — для исключения событий от анонимных пользователей.

```
audit_config:
  unified_agent_backend:
    format: TXT
    log_name: audit
  stderr_backend:
    format: JSON
  log_class_config:
    - log_class: ClusterAdmin
      enable_logging: true
      log_phase: [Received, Completed]
    - log_class: DatabaseAdmin
      enable_logging: true
      log_phase: [Completed]
      exclude_account_type: [Anonymous]
    - log_class: Default
      enable_logging: true
  heartbeat:
    interval_seconds: 60
```

Примеры

Следующие вкладки демонстрируют одно и то же событие аудитного лога, записанное с разной конфигурацией направления.

JSON

Формат `JSON` создаёт записи следующего вида:

```
2023-03-14T10:41:36.485788Z: {"paths":"[/my_dir/db1/some_dir]","tx_id":"281474976775658","database":"/my_dir/db1","remote_address":"ipv6:[xxxx:xxx:xxx:xxx:x:xxxx:xxx:xxxx]:xxxxx","status":"SUCCESS","subject":"{none}","sanitized_token":"{none}","detailed_status":"StatusAccepted","operation":"MODIFY ACL","component":"schemeshard","acl_add":"[+(ConnDB):subject:-]"}
2023-03-13T20:07:30.927210Z: {"reason":"Check failed: path: '/my_dir/db1/some_dir', error: path exist, request accepts it (id: [OwnerId: 72075186224037889, LocalPathId: 3], type: EPathTypeDir, state: EPathStateNoChanges)","paths":"[/my_dir/db1/some_dir]","tx_id":"844424930216970","database":"/my_dir/db1","remote_address":"ipv6:[xxxx:xxx:xxx:xxx:x:xxxx:xxx:xxxx]:xxxxx","status":"SUCCESS","subject":"{none}","sanitized_token":"{none}","detailed_status":"StatusAlreadyExists","operation":"CREATE DIRECTORY","component":"schemeshard"}
2025-11-03T18:07:39.056211Z: {"@log_type":"audit","begin_tx":1,"commit_tx":1,"component":"grpc-proxy","database":"/my_dir/db1","detailed_status":"SUCCESS","end_time":"2025-11-03T18:07:39.056204Z","grpc_method":"Ydb.Query.V1.QueryService/ExecuteQuery","operation":"ExecuteQueryRequest","query_text":"SELECT * FROM `my_row_table`;","remote_address":"ipv6:[xxxx:xxx:xxx:xxx:x:xxxx:xxx:xxxx]","sanitized_token":"xxxxxxxx.**","start_time":"2025-11-03T18:07:39.054863Z","status":"SUCCESS","subject":"serviceaccount@as"}
2025-11-03T17:41:44.203214Z: {"component":"monitoring","remote_address":"ipv6:[xxxx:xxx:xxx:xxx:x:xxxx:xxx:xxxx]","operation":"HTTP REQUEST","method":"POST","url":"/viewer/query","params":"base64=false&schema=multipart","body":"{\"query\":\"SELECT * FROM `my_row_table`;\",\"database\":\"/local\",\"action\":\"execute-query\",\"syntax\":\"yql_v1\"}","status":"IN-PROCESS","reason":"Execute"}
```

TXT

Формат `TXT` создаёт записи следующего вида:

```
2023-03-14T10:41:36.485788Z: component=schemeshard, tx_id=281474976775658, remote_address=ipv6:[xxxx:xxx:xxx:xxx:x:xxxx:xxx:xxxx]:xxxxx, subject={none}, database=/my_dir/db1, operation=MODIFY ACL, paths=[/my_dir/db1/some_dir], status=SUCCESS, detailed_status=StatusSuccess, acl_add=[+(ConnDB):subject:-]
2023-03-13T20:07:30.927210Z: component=schemeshard, tx_id=281474976775657, remote_address=ipv6:[xxxx:xxx:xxx:xxx:x:xxxx:xxx:xxxx]:xxxxx, subject={none}, database=/my_dir/db1, operation=CREATE DIRECTORY, paths=[/my_dir/db1/some_dir], status=SUCCESS, detailed_status=StatusAlreadyExists, reason=Check failed: path: '/my_dir/db1/some_dir', error: path exist, request accepts it (id: [OwnerId: 72075186224037889, LocalPathId: 3], type: EPathTypeDir, state: EPathStateNoChanges)
2025-11-03T18:07:39.056211Z: component=grpc-proxy, tx_id=281474976775656, remote_address=ipv6:[xxxx:xxx:xxx:xxx:x:xxxx:xxx:xxxx]:xxxxx, subject=serviceaccount@as, database=/my_dir/db1, operation=ExecuteQueryRequest, query_text=SELECT * FROM `my_row_table`; status=SUCCESS, detailed_status=StatusSuccess, begin_tx=1, commit_tx=1, end_time=2025-11-03T18:07:39.056204Z, grpc_method=Ydb.Query.V1.QueryService/ExecuteQuery, sanitized_token=xxxxxxxx.**, start_time=2025-11-03T18:07:39.054863Z
2025-11-03T17:41:44.203214Z: component=monitoring, remote_address=ipv6:[xxxx:xxx:xxx:xxx:x:xxxx:xxx:xxxx], operation=HTTP REQUEST, method=POST, url=/viewer/query, params=base64=false&schema=multipart, body={"query":"SELECT * FROM `my_row_table`;","database":"/local","action":"execute-query","syntax":"yql_v1"}, status=IN-PROCESS, reason=Execute
```

JSON_LOG_COMPATIBLE

Формат `JSON_LOG_COMPATIBLE` создаёт записи следующего вида:

```
{"@timestamp":"2023-03-14T10:41:36.485788Z","@log_type":"audit","paths":"[/my_dir/db1/some_dir]","tx_id":"281474976775658","database":"/my_dir/db1","remote_address":"ipv6:[xxxx:xxx:xxx:xxx:x:xxxx:xxx:xxxx]:xxxxx","status":"SUCCESS","subject":"{none}","detailed_status":"StatusAccepted","operation":"MODIFY ACL","component":"schemeshard","acl_add":"[+(ConnDB):subject:-]"}
{"@timestamp":"2023-03-13T20:07:30.927210Z","@log_type":"audit","reason":"Check failed: path: '/my_dir/db1/some_dir', error: path exist, request accepts it (id: [OwnerId: 72075186224037889, LocalPathId: 3], type: EPathTypeDir, state: EPathStateNoChanges)","paths":"[/my_dir/db1/some_dir]","tx_id":"844424930216970","database":"/my_dir/db1","remote_address":"ipv6:[xxxx:xxx:xxx:xxx:x:xxxx:xxx:xxxx]:xxxxx","status":"SUCCESS","subject":"{none}","detailed_status":"StatusAlreadyExists","operation":"CREATE DIRECTORY","component":"schemeshard"}
{"@timestamp":"2025-11-03T18:07:39.056211Z","@log_type":"audit","begin_tx":1,"commit_tx":1,"component":"grpc-proxy","database":"/my_dir/db1","detailed_status":"SUCCESS","end_time":"2025-11-03T18:07:39.056204Z","grpc_method":"Ydb.Query.V1.QueryService/ExecuteQuery","operation":"ExecuteQueryRequest","query_text":"SELECT * FROM `my_row_table`;","remote_address":"ipv6:[xxxx:xxx:xxx:xxx:x:xxxx:xxx:xxxx]","start_time":"2025-11-03T18:07:39.054863Z","status":"SUCCESS","subject":"serviceaccount@as","sanitized_token":"xxxxxxxx.**"}
{"@timestamp":"2025-11-03T17:41:44.203214Z","@log_type":"audit","component":"monitoring","remote_address":"ipv6:[xxxx:xxx:xxx:xxx:x:xxxx:xxx:xxxx]","operation":"HTTP REQUEST","method":"POST","url":"/viewer/query","params":"base64=false&schema=multipart","body":"{\"query\":\"SELECT * FROM `my_row_table`;\",\"database\":\"/local\",\"action\":\"execute-query\",\"syntax\":\"yql_v1\"}","status":"IN-PROCESS","reason":"Execute"}
```

JSON-конверт

Шаблон JSON-конверта `{"audit": %message%, "source": "ydb-audit-log"}` создаёт записи следующего вида:

```
{"audit":"2023-03-14T10:41:36.485788Z: {\"paths\":\"[/my_dir/db1/some_dir]\",\"tx_id\":\"281474976775658\",\"database\":\"/my_dir/db1\",\"remote_address\":\"ipv6:[xxxx:xxx:xxx:xxx:x:xxxx:xxx:xxxx]:xxxxx\",\"status\":\"SUCCESS\",\"subject\":\"{none}\",\"detailed_status\":\"StatusAccepted\",\"operation\":\"MODIFY ACL\",\"component\":\"schemeshard\",\"acl_add\":\"[+(ConnDB):subject:-]\"}\n","source":"ydb-audit-log"}
{"audit":"2023-03-13T20:07:30.927210Z: {\"reason\":\"Check failed: path: '/my_dir/db1/some_dir', error: path exist, request accepts it (id: [OwnerId: 72075186224037889, LocalPathId: 3], type: EPathTypeDir, state: EPathStateNoChanges)\",\"paths\":\"[/my_dir/db1/some_dir]\",\"tx_id\":\"844424930216970\",\"database\":\"/my_dir/db1\",\"remote_address\":\"ipv6:[xxxx:xxx:xxx:xxx:x:xxxx:xxx:xxxx]:xxxxx\",\"status\":\"SUCCESS\",\"subject\":\"{none}\",\"detailed_status\":\"StatusAlreadyExists\",\"operation\":\"CREATE DIRECTORY\",\"component\":\"schemeshard\"}\n","source":"ydb-audit-log"}
{"audit":"2025-11-03T18:07:39.056211Z: {\"@log_type\":\"audit\",\"begin_tx\":1,\"commit_tx\":1,\"component\":\"grpc-proxy\",\"database\":\"/my_dir/db1\",\"detailed_status\":\"SUCCESS\",\"end_time\":\"2025-11-03T18:07:39.056204Z\",\"grpc_method\":\"Ydb.Query.V1.QueryService/ExecuteQuery\",\"operation\":\"ExecuteQueryRequest\",\"query_text\":\"SELECT * FROM `my_row_table`;\",\"remote_address\":\"ipv6:[xxxx:xxx:xxx:xxx:x:xxxx:xxx:xxxx]\",\"sanitized_token\":\"xxxxxxxx.**\",\"start_time\":\"2025-11-03T18:07:39.054863Z\",\"status\":\"SUCCESS\",\"subject\":\"serviceaccount@as\"}\n","source":"ydb-audit-log"}
{"audit":"2025-11-03T17:41:44.203214Z: {\"component\":\"monitoring\",\"remote_address\":\"ipv6:[xxxx:xxx:xxx:xxx:x:xxxx:xxx:xxxx]\",\"operation\":\"HTTP REQUEST\",\"method\":\"POST\",\"url\":\"/viewer/query\",\"params\":\"base64=false&schema=multipart\",\"body\":\"{\\\"query\\\":\\\"SELECT * FROM `my_row_table`;\\\",\\\"data
```

```
base\\\":\\\"/local\\\",\\\"action\\\":\\\"execute-query\\\",\\\"syntax\\\":\\\"yql_v1\\\"}\",\"status\":\"IN
-PROCESS\",\"reason\":\"Execute\"}\n","source":"ydb-audit-log"}
```


Форматированный JSON

Та же запись аудитного лога, представленная для удобства чтения:


```
{
  "paths": "[/my_dir/db1/some_dir]",
  "tx_id": "281474976775658",
  "database": "/my_dir/db1",
  "remote_address": "ipv6:[xxxx:xxx:xxx:xxx:x:xxxx:xxx:xxxx]:xxxxx",
  "status": "SUCCESS",
  "subject": "{none}",
  "detailed_status": "StatusAccepted",
  "operation": "MODIFY ACL",
  "component": "schemeshard",
  "acl_add": "[+(ConnDB):subject:-]"
}
```

Шифрование в YDB

YDB предоставляет два основных подхода к шифрованию пользовательских данных:

- Шифрование данных при хранении
- Шифрование данных при передаче

Рекомендуется использовать их оба во всех производственных средах, где обрабатываются чувствительные данные.

Шифрование данных при хранении

YDB поддерживает прозрачное шифрование данных на уровне прокси распределённого хранилища с использованием алгоритма ChaCha8. YDB включает две реализации этого алгоритма, которые переключаются в зависимости от доступности набора инструкций AVX-512F.

По умолчанию шифрование данных при хранении отключено. Инструкции по его включению можно найти в разделе Конфигурация Blob Storage.

Более подробную информацию о реализации можно найти в ydb/core/blobstorage/dsproxy/dsproxy_encrypt.cpp и ydb/core/blobstorage/crypto.

Шифрование данных при передаче

Так как YDB является распределённой системой, обычно работающей на кластере, часто расположенным в нескольких датацентрах или зонах доступности, пользовательские данные регулярно передаются по сети. Могут использоваться различные протоколы, каждый из которых может быть настроен для работы с использованием TLS. Ниже приведён список протоколов, поддерживаемых YDB:

- Интерконнект — специализированный протокол для общения между узлами YDB.
- YDB в роли сервера:
  - gRPC — для внешнего взаимодействия с клиентскими приложениями, разработанными для нативной работы с YDB через SDK или CLI.
  - Протокол Kafka — для внешнего взаимодействия с клиентскими приложениями, изначально разработанными для работы с Apache Kafka.
  - HTTP — для работы с YDB UI, публикации метрик и других вспомогательных конечных эндпоинтов.
- YDB в роли клиента:
  - LDAP — для аутентификации пользователей.
  - Федеративные запросы — функциональность, позволяющая YDB выполнять запросы к различным внешним источникам данных. Запросы к некоторым источникам отправляются напрямую из процесса `ydbd`, в то время как другие проксируются через отдельный процесс-коннектор.
  - Трассировочные данные отправляются во внешний сборщик через gRPC.
- В асинхронной репликации между двумя базами данных YDB одна из них выступает в роли клиента по отношению к другой.
- В трансфере между двумя базами данных YDB одна из них выступает в роли клиента по отношению к другой.

По умолчанию шифрование данных при передаче отключено и должно быть включено отдельно для каждого протокола. Они могут использовать общий набор TLS-сертификатов или отдельные. Инструкции по включению TLS можно найти в разделе tls.

Краткая запись управления доступом

При описании или логировании разрешений, которые выдаются пользователям (например при записи аудитных логов), может использоваться специальная краткая запись управления доступом. Запись может немного меняться, в зависимости от списка разрешений и их наследования у дочерних объектов.

Формат записи

Каждая запись начинается со знака `+` и состоит из 2 или 3 атрибутов, перечисленных через символ `:`.

Эти атрибуты:

- список разрешений. Обязательный.
- SID субъекта предоставленных разрешений. Обязательный.
- тип наследования. Необязательный.

Примеры

```
+R:subject:O
+W:subject
+(SR|UR):subject
+(SR|ConnDB):subject:OC+
```

Список разрешений

Для записи каждого разрешения используется короткое сокращение.

Группы разрешений

Группы разрешений — это объединения нескольких разрешений. По возможности в краткой записи будет указано одна из групп.
Например, `+R:subject` — разрешение на чтение.

Группа	Описание
`L`	(list) перечисление. Состоит из разрешений на чтение аттрибутов ACL и описание объектов.
`R`	(read) чтение. Состоит из разрешений на чтение из таблиц, топиков и перечисления.
`W`	(write) запись. Состоит из разрешений на обновление и удаление записей из таблиц, запись атрибутов ACL, создание подкаталогов, создание таблиц, очередей, изменение и удаление объектов, изменении пользовательских атрибутов
`UL`	(use legacy) использование (устаревшее). Состоит из разрешений за чтение, запись и предоставления прав доступа
`U`	(use) использование. Состоит из разрешений за чтение, запись, предоставления прав доступа и отправку запросов к БД
`M`	(manage) управление. Состоит из разрешений на создание и удаление БД
`FL`	(full legacy) все права (устаревшее). Состоит из разрешений на использование (устаревшее) и управление
`F`	(full) все права. Состоит из разрешений на использование и управление

Простые разрешения

Если нет другой возможности, в сокращенной записи будет приведен список разрешений в круглых скобках через символ вертикальной черты `|`.
Например, `+(SR|UR):subject` — разрешение на чтение и обновление записей в таблицы.

Разрешение	Описание
`SR`	(select row) чтение из таблиц
`UR`	(update row) обновление записей таблиц
`ER`	(erase row) удаление записей из таблиц
`RA`	(read attributes) чтение атрибутов ACL
`WA`	(write attributes) запись атрибутов ACL
`CD`	(create directory) создание подкаталога
`CT`	(create table) создание таблиц
`CQ`	(create queue) создание очередей
`RS`	(remove schema) удаление объектов
`DS`	(describe schema) описание объектов, содержимое каталогов
`AS`	(alter schema) изменение объектов
`CDB`	(create database) создание БД
`DDB`	(drop database) удаление БД
`GAR`	(grant access rights) предоставление прав доступа (только из списка собственных)
`WUA`	(write user attributes) изменение пользовательских атрибутов
`ConnDB`	(connect database) подключение и отправка запросов в БД

Типы наследования

Для описания передачи наследования дочерним объектам могут использоваться один или несколько флагов наследования.

Флаг	Описание
`-`	без наследования
`O`	эта запись будут наследоваться дочерними объектами
`C`	эта запись будет наследоваться дочерними контейнерами
`+`	эта запись будет использоваться только для наследования и не будут использоваться для проверки доступа

Разработка YDB

Данный раздел содержит материалы для разработчиков и контрибьюторов YDB.

- Сборка и тестирование с использованием Ya Make
- Работа над изменениями
- Управление релизами
- Разработка акторов
- Архитектура распределённого хранилища
- Тестирование с помощью нагружающих акторов

Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

Процесс разработки: работа над изменениями кода YDB

Этот раздел содержит пошаговый сценарий, который поможет вам выполнить необходимые шаги по настройке и узнать, как внести изменения в проект YDB. Этому сценарию не обязательно строго следовать, вы можете разработать свой собственный подход на основе предоставленной информации.

Настройка окружения

Учетная запись на GitHub

Чтобы предлагать какие-либо изменения в исходном коде YDB, необходима учетная запись на GitHub. Зарегистрируйтесь на GitHub, если вы еще этого не сделали.

Пара ключей SSH

- Для подключения к GitHub вы можете использовать ssh или token. Рекомендуемый метод — ssh-ключи.
- Если у вас ещё нет ключей, создайте новые. Полные инструкции находятся на этой странице GitHub.

Удаленная разработка

Если вы разрабатываете на удалённом компьютере, вы можете использовать ключ со своего ноутбука. Настройте переадресацию SSH-агента по инструкции GitHub.

Git CLI

У вас должна быть установлена утилита командной строки git для запуска команд из консоли. Посетите страницу Downloads официального веб-сайта для получения инструкций по установке.

Чтобы установить ее в Linux/Ubuntu, запустите:

```
sudo apt-get update
sudo apt-get install git
```

Зависимости сборки

На компьютере разработчика должны быть установлены некоторые библиотеки.

Чтобы установить их под Linux/Ubuntu, запустите:

```
sudo apt-get update
sudo apt-get install libidn11-dev libaio-dev libc6-dev
```

GitHub CLI (необязательно)

Использование GitHub CLI позволяет создавать Pull Request'ы и управлять репозиторием из командной строки. Вы также можете использовать GitHub UI для таких действий.

Установите GitHub CLI, как описано на домашней странице. Для Linux Ubuntu вы можете перейти непосредственно к https://github.com/cli/cli/blob/trunk/docs/install_linux.md#debian-ubuntu-linux-raspberry-pi-os-apt.

Выполните настройку аутентификации:

```
gh auth login
```

Вам будет задано несколько вопросов в интерактивном режиме, ответьте на них следующим образом:

Вопрос	Ответ
What account do you want to log into?	**GitHub.com**
What is your preferred protocol for Git operations?	**SSH**
Upload your SSH public key to your GitHub account?	Выберите файл с открытым ключом (расширение `.pub`) из тех, что были созданы на шаге "Создать пару ключей SSH", например **/home/user/.ssh/id_ed25519.pub**
Title for your SSH key	**GitHub CLI** (оставьте значение по умолчанию)
How would you like to authenticate GitHub CLI	**Paste your authentication token**

После последнего ответа вам будет предложено ввести токен, который вы можете сгенерировать в пользовательском интерфейсе GitHub:

```
Tip: you can generate a Personal Access Token here https://github.com/settings/tokens
The minimum required scopes are 'repo', 'read:org', 'admin:public_key'.
? Paste your authentication token:
```

Откройте настройки токенов в GitHub, нажмите "Generate new token" / "Classic", поставьте галочки в четырех полях:

- Поле **`workflow`**
- Три других, как указано в подсказке: "repo", "admin:public_key" and "read:org" (в разделе "admin:org")

И скопипастите показанный токен, чтобы завершить настройку GitHub CLI.

Форк и клонирование репозитория

Официальным репозиторием YDB является https://github.com/ydb-platform/ydb, расположенный под учетной записью организации YDB `ydb-platform`.

Чтобы работать над изменениями в YDB, вы должны создать форк репозитория в вашем аккаунте GitHub. Нажмите на кнопку `Fork` на странице официального репозитория YDB.

После того, как ваш форк создан, создайте локальный git репозиторий с двумя remote:

- `official`: официальный репозиторий YDB, с ветками main и stable
- `fork`: ваш форк YDB для разработки

```
mkdir -p ~/ydbwork
cd ~/ydbwork
git clone -o official git@github.com:ydb-platform/ydb.git
```

```
cd ydb
git remote add fork git@github.com:{your_github_user_name}/ydb.git
```

После завершения у вас будет форк репозитория YDB Git, клонированный в `~/ydbwork/ydb`.

Форк репозитория - это мгновенное действие, однако клонирование на локальный компьютер требует некоторого времени для передачи около 650 МБ данных репозитория по сети.

Затем установите поведение по умолчанию для команды `git push`:

```
git config push.default current
git config push.autoSetupRemote true
```

Таким образом, команда `git push {remote}` будет автоматически устанавливать upstream `{remote}` для текущей ветки и последующие команды `git push` будут отправлять только текущую ветку.

Если вы собираетесь использовать GitHub CLI, установите `ydb-platform/ydb` репозиторием по умолчанию для GitHub CLI:

```
gh repo set-default ydb-platform/ydb
```

Настройка авторства коммитов

Запустите следующую команду, чтобы указать свое имя и адрес электронной почты для коммитов, отправляемых с помощью Git (замените имя пользователя и email на ваши):

```
git config --global user.name "Marco Polo"
git config --global user.email "marco@ydb.tech"
```

Работа над изменением

Чтобы начать работу над изменением, убедитесь, что шаги, указанные в разделе Настройка окружения выше, выполнены.

Обновление транка

Обычно для начала работы над изменением требуется последняя версия официального репозитория. Синхронизируйте вашу локальную ветку `main`, выполнив следующую команду:

Если ваша текущая локальная ветка `main`:

```
git pull --ff-only official main
```

Если ваша текущая локальная ветка не `main`:

```
cd ~/ydbwork/ydb
git fetch official main:main
```

Эта команда обновляет локальную ветку `main` без checkout.

Создайте ветку разработки

Создайте ветку разработки с помощью Git (замените "feature42" на название вашей новой ветки):

```
git checkout -b feature42
```

Внесите изменения и коммиты

Редактируйте файлы локально, используйте стандартные команды Git для добавления файлов, проверки статуса, выполнения коммитов и отправки изменений в ваш fork репозиторий:

```
git add .
git status
```

```
git commit -m "Implemented feature 42"
git push fork
```

Последующие push не требуют upstream или имени ветки:

```
git push
```

Создайте Pull Request в официальный репозиторий

Когда изменения будут завершены и протестированы локально (см. Ya Build and Test), создайте Pull Request.

GitHub UI

Откройте страницу вашей ветки на GitHub.com (`https://github.com/{your_github_user_name}/ydb/tree/{branch_name}`), нажмите `Contribute` и затем `Open Pull Request`.
Также можно использовать ссылку в выводе команды `git push`, чтобы создать Pull Request:

```
...
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for '{branch_name}' on GitHub by visiting:
remote:      https://github.com/{your_github_user_name}/test/pull/new/{branch_name}
...
```

GitHub CLI

Установите и сконфигурируйте GitHub CLI.

```
cd ~/ydbwork/ydb
```

```
gh pr create --title "Feature 42 implemented"
```

После ответа на некоторые вопросы Pull Request будет создан, и вы получите ссылку на его страницу на GitHub.com.

Заполните описание к Pull Request'у

При создании Pull Request'а описание будет заполнено текстом из шаблона, который нужно отредактировать:

1. **Changelog Entry.** В этот блок следует добавить описание изменения для конечных пользователей системы (см. требования). Содержимое этого блока будет опубликовано в списке изменений, если PR будет замержен.
2. **Description for reviewers.** В этот блок можно добавить ссылку на задачу и любую дополнительную информацию, которая будет полезна для ревью вашего изменения. Содержимое этого блока не попадёт в список изменений.

Требования к Changelog Entry

Сообщение в Changelog Entry должно отвечать следующим требованиям:

- должно быть написано на английском языке;
- опираться на термины, которые используются в глоссарии;
- описывать, что в работе системы изменилось для конечного пользователя.

Предварительные проверки

Перед мержем изменений выполняются прекоммитные проверки Pull Request'а.

Для изменений в коде YDB прекоммитные проверки собирают артефакты, и запускают описанные в файлах `ya.make` тесты. Сборка/тесты выполняются на специальном merge коммите, который мержит ваши изменения с текущей веткой `main`.

Вы можете увидеть статус проверок на странице Pull Request'а. Также, ключевая информация о ходе сборки/тестов YDB и текущий статус публикуются в комментариях к PR.

Если вы не являетесь членом команды YDB, проверки не будут запущены до тех пор, пока член команды не рассмотрит ваши изменения, и не одобрит PR для тестов, присвоив метку `ok-to-test`.

Проверки перезапускаются каждый раз, когда пушатся новые изменения, предыдущая проверка прерывается, если она еще не завершена. Каждая итерация проверок создает собственный комментарий на странице PR, поэтому там сохраняется история проверок.

Участники команды разработки YDB могут рестартовать проверки на новом merge коммите без пуша изменений. Для этого необходимо добавить к PR метку `rebase-and-check`.

Результаты тестирования

Вы можете кликнуть по количеству тестов в разных разделах комментария с результатами тестирования, чтобы перейти к простому HTML-отчету о тестировании. В этом отчете вы можете увидеть, какие тесты были пройдены неудачно/успешно, и получить доступ к их логам.

История тестов

Каждый раз, когда тесты запускаются YDB CI, их результаты загружаются в приложение Test History. В комментарии к результатам тестирования есть ссылка "Test history", ведущая на страницу с соответствующим прогоном в этом приложении.

В "Test history" члены команды YDB могут просматривать тестовые прогоны, выполнять поиск тестов, просматривать логи и сравнивать их между различными тестовыми прогонами. Если какой-либо тест завершается сбоем на некоторой прекоммитной проверке, в его истории можно увидеть, был ли этот сбой вызван данным изменением, или тест был сломан ранее.

Review и merge

Pull Request может быть замержен после получения Approve от члена команды YDB. Для коммуникации используются комментарии. После подтверждения участник команды YDB нажимает на кнопку "Merge".

Обновление изменения

Если в вашем репозитории открыт Pull Request для какой-либо ветки разработки, он будет обновляться каждый раз, когда вы выполняете push в эту ветку, с перезапуском проверок.

Rebase changes

При возникновении конфликтов, вы можете сделать rebase своих изменений поверх текущего транка из официального репозитория. Чтобы сделать это, обновите транк на вашей локальной машине, и запустите команду rebase:

```
# Предполагается, что ваша активная ветка является вашей веткой разработки
git fetch official main:main
git rebase main
```

Перенос патчей в стабильную ветку

Когда есть необходимость перенести патч в стабильную ветку, отведите ветку от ветки stable:

```
git fetch official
git checkout -b "cherry-pick-fix42" official/stable-24-1
```

Затем используйте cherry-pick для переноса патча и сделайте push ветки в ваш fork:

```
git cherry-pick {fixes_commit_hash}
git push fork
```

И создайте Pull Request из вашей ветки с патчем в стабильную ветку. Это делается аналогично открытию PR в ветку `main`, но необходимо убедиться что в качестве target выбрана нужная стабильная ветка.

Если вы используете GitHub CLI, target ветка указывается в опции `-B`:

```
gh pr create --title "Title" -B stable-24-1
```

Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

Сборка и тестирование YDB с использованием Ya Make

Ya Make - это система сборки и тестирования, исторически используемая для разработки YDB. Изначально разработанная для C++, теперь она поддерживает ряд языков программирования, включая Java, Go и Python.

Язык конфигурации сборки **Ya Make** является основным для YDB, с файлом `ya.make` в каждом каталоге, представляющем таргеты Ya Make.

Настройте среду разработки, как описано в статье Работа над изменениями - Настройка окружения для работы с `Ya Make`.

Выполнение ваших команд

В корневом каталоге репозитория YDB расположен скрипт `ya` для запуска команд `Ya Make` из командной строки. Его можно добавить в переменную окружения PATH, чтобы он запускался без указания полного пути. Для Linux/Bash и репозитория GitHub, склонированного в `~/ydbwork/ydb`, можно использовать следующую команду:

```
echo "alias ya='~/ydbwork/ydb/ya'" >> ~/.bashrc
source ~/.bashrc
```

Запуск `ya` без параметров выводит справку:

```
$ ya
Yet another build tool.

Usage: ya [--precise] [--profile] [--error-file ERROR_FILE] [--keep-tmp] [--no-logs] [--no-report] [--no-tmp-
dir] [--print-path] [--version] [-v] [--diag] [--help] <SUBCOMMAND> [OPTION]...

Options:
...

Available subcommands:
...
```

Подробная справка по любой подкоманде может быть получена её запуском с флагом `--help`, например:

```
$ ya make --help
Build and run tests
To see more help use -hh/-hhh

Usage:
  ya make [OPTION]... [TARGET]...

Examples:
  ya make -r               Build current directory in release mode
  ya make -t -j16 library  Build and test library with 16 threads
  ya make --checkout -j0   Checkout absent directories without build

Options:
...
```

Скрипт `ya` загружает при запуске необходимые артефакты, зависящие от платформы, и кэширует их локально. Периодически скрипт обновляется ссылками на новые версии артефактов.

Настройка IDE

При использовании IDE для разработки доступна команда `ya ide`, которая помогает создать проект с преднастроенными инструментами. Поддерживаются следующие IDE: goland, idea, pycharm, venv, vscode (multilanguage, clangd, go, py).

Перейдите в каталог в исходном коде, который должен быть корневым для вашего проекта. Запустите команду `ya ide`, указав имя IDE, и целевой каталог для записи конфигурации проекта IDE в параметре `-P`. Например, для работы с изменениями библиотеки YQL в vscode вы можете запустить следующую команду:

```
cd ~/ydbwork/ydb/library/yql
ya ide vscode -P=~/ydbwork/vscode/yqllib
```

Теперь можно открыть `~/ydbwork/vscode/yqllib/ide.code-workspace` из vscode.

Сборка таргета

В `Ya Make` есть 3 основных типа таргетов: программа, тест и библиотека. Чтобы собрать таргет, запустите `ya make` с именем каталога. Например, для сборки YDB CLI запустите:

```
cd ~/ydbwork/ydb
ya make ydb/apps/ydb
```

Вы также можете запустить `ya make` из каталога таргета без параметров:

```
cd ~/ydbwork/ydb/ydb/apps/ydb
ya make
```

Запуск тестов

Запуск команды `ya test` в каком-либо каталоге создаст все бинарные файлы тестов, расположенные внутри его подкаталогов, и запустит тесты.

Например, для запуска small тестов YDB Core запустите:

```
cd ~/ydbwork/ydb
ya test ydb/core
```

Для запуска medium и large тестов добавьте опции `-tt` и `-ttt` к вызову `ya test` соответственно.

Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

Логирование

Назначение

В YDB осуществляется **структурированное логирование** происходящих событий.
Структурированность подразумевает, что в логе (журнале работы) текстовое описание событий отделено от параметров этих событий, а сами значения параметров хранятся и обрабатываются независимо друг от друга. Например, если произошла ошибка чтения файла, то её описание — это текст вида "Ошибка чтения файла", а параметры — это путь к файлу, код системной ошибки, текстовое описание системной ошибки и т.п.

Разделение описания событий и их параметров впоследствии может быть использовано для эффективного поиска сообщений, сбора статистики, оптимизированного хранения журнала и прочих задач.

Базовые средства записи сообщений в лог

В большинстве случаев при записи сообщений в лог выполняются следующие условия:

1. Исходный код содержится в некотором файле `.cpp` (логирование из заголовочных файлов поддерживается, но рассматривается далее).
2. В одном файле `.cpp` все сообщения пишутся от имени одного и того же компонента. Поэтому в начале файла (но после всех директив `#include`) должен быть определён макрос `YDB_LOG_THIS_FILE_COMPONENT`, который задаёт код компонента для всего файла.

> **Примечание**
>
> Логирование в YDB предполагает, что система состоит из большого количества внутренних компонентов, за каждым из которых закреплён его уникальный код. С полным списком компонентов и их кодов можно ознакомиться на GitHub.
>
> При записи сообщения в лог должен быть явно или неявно указан код компонента, от имени которого происходит запись этого сообщения. Впоследствии имя компонента всегда выводится в лог и может быть использовано для поиска или фильтрации содержимого лога.
>
> Кроме того, для каждого компонента могут быть настроены свои параметры логирования.

Пример:

```
#define YDB_LOG_THIS_FILE_COMPONENT NKikimrServices::STATESTORAGE
```

> **Важно**
>
> Система сборки может объединить несколько файлов `.cpp` в одну единицу компиляции с помощью `JOIN_SRCS`, который подключает их через директивы `#include`. Поэтому в конце каждого исходного файла настоятельно рекомендуется отменять определение макроса:
>
> ```
> #undef YDB_LOG_THIS_FILE_COMPONENT
> ```
>
> В противном случае определение макроса `YDB_LOG_THIS_FILE_COMPONENT` может распространиться на несколько файлов исходного кода.

3. Логирование происходит в ходе работы актора, и для работы с контекстом (в частности, отправки сообщений актору логирования) доступна переменная `NActors::TlsActivationContext`.

> **Важно**
>
> Если хотя бы одно из трёх перечисленных условий не выполняется, то следует воспользоваться расширенными средствами логирования.

Далее в этом файле `.cpp` для логирования сообщений могут использоваться следующие макросы:

Макрос	Уровень сообщения
`YDB_LOG_EMERG(message, ...values...)`	Возможен сбой системы (например, отказ кластера).
`YDB_LOG_ALERT(message, ...values...)`	Возможна деградация системы, компоненты системы могут выйти из строя.
`YDB_LOG_CRIT(message, ...values...)`	Критическое состояние.
`YDB_LOG_ERROR(message, ...values...)`	Некритическая ошибка.
`YDB_LOG_WARN(message, ...values...)`	Предупреждение, на которое следует отреагировать и исправить, если оно не временное.
`YDB_LOG_NOTICE(message, ...values...)`	Произошло событие, существенное для системы или пользователя.
`YDB_LOG_INFO(message, ...values...)`	Отладочная информация для сбора статистики.
`YDB_LOG_DEBUG(message, ...values...)`	Отладочная информация для разработчиков.
`YDB_LOG_TRACE(message, ...values...)`	Очень подробная отладочная информация.

В аргументах вызова перечисленных макросов указываются следующие параметры:

- `message` — текстовое сообщение;

- `...values...` — опциональные параметры. Каждый параметр указывается в виде пары `{name, value}`, где `name` — текстовая строка с названием параметра, а `value` — значение параметра.

Примеры записи сообщений в журнал:

1. Сообщение без параметров:

```
YDB_LOG_INFO("Module started");
```

2. Сообщение с параметрами:

```
YDB_LOG_ERROR("Unable to open file",
    {"sourceFilePath", filename},
    {"errorCode", err});
```

Динамическое определение уровня сообщения

В некоторых случаях уровень сообщения определяется динамически на этапе исполнения. Тогда следует использовать макрос, который принимает уровень сообщения в виде аргумента. Синтаксис макроса:

```
YDB_LOG(prio, message, ...values...);
```

Пример использования:

```
auto prio = NActors::NLog::PRI_ERROR;
...
YDB_LOG(prio, "Unable to open file",
    {"sourceFilePath", filename},
    {"errorCode", err});
```

Расширенные средства записи сообщений в лог

Ядро логирования

Наиболее низкоуровневым средством для записи сообщений в журнал является макрос `YDB_LOG_CTX_COMP`. Он проверяет необходимость логирования (согласно настройкам), формирует текстовое сообщение, прикрепляет к нему опциональные параметры и затем отправляет всю эту информацию актору логирования. Далее, актор логирования обрабатывает полученное сообщение согласно своим текущим настройкам (например, записывает в файл или syslog).

> **Примечание**
>
> Все остальные средства логирования (в том числе и базовые средства логирования) построены на основе макроса `YDB_LOG_CTX_COMP`.

Полный синтаксис макроса логирования выглядит следующим образом:

```
YDB_LOG_CTX_COMP(ctx, prio, comp, message, ...values...)
```

В аргументах вызова макроса указываются следующие параметры:

- `ctx` — контекст исполнения актора (необходим для отправки сообщения актору логирования);
- `prio` — уровень логирования сообщения (соответствует уровням логирования);
- `comp` — идентификатор компонента;
- `message` — текстовое сообщение;
- `...values...` — опциональные параметры. Каждый параметр указывается в виде пары `{name, value}`, где `name` — текстовая строка с названием параметра, а `value` — значение параметра.

В примерах ниже `EXAMPLE_COMP_CODE` обозначает код компонента логирования. В рабочем коде вместо него укажите существующий компонент, например `NKikimrServices::STATESTORAGE`.

Примеры записи сообщения в журнал

1. Сообщение без параметров:

```
YDB_LOG_CTX_COMP(ctx, PRI_INFO, EXAMPLE_COMP_CODE, "Module started");
```

2. Сообщение с параметрами:

```
YDB_LOG_CTX_COMP(ctx, PRI_ERROR, EXAMPLE_COMP_CODE, "Unable to open file",
    {"sourceFilePath", filename},
    {"errorCode", err});
```

В данном примере происходит запись сообщения с текстом `Unable to open file` и двумя параметрами: `sourceFilePath` (значение берётся из переменной `filename`) и `errorCode` (значение берётся из переменной `err`). Для передачи сообщения актору логирования используется контекст исполнения `ctx`, а в качестве кода компонента передаётся `EXAMPLE_COMP_CODE`.

Макрос `YDB_LOG_CTX_COMP` подразумевает динамическое определение уровня сообщения и его передачу в виде аргумента. По аналогии с базовыми средствами логирования, существует ряд макросов, которые не требуют передачи уровня сообщения в виде аргумента.

Макросы без указания уровня сообщения в виде аргумента

Макрос	Уровень сообщения
`YDB_LOG_EMERG_CTX_COMP`	Возможен сбой системы (например, отказ кластера).
`YDB_LOG_ALERT_CTX_COMP`	Возможна деградация системы, компоненты системы могут выйти из строя.
`YDB_LOG_CRIT_CTX_COMP`	Критическое состояние.
`YDB_LOG_ERROR_CTX_COMP`	Некритическая ошибка.
`YDB_LOG_WARN_CTX_COMP`	Предупреждение, на которое следует отреагировать и исправить, если оно не временное.
`YDB_LOG_NOTICE_CTX_COMP`	Произошло событие, существенное для системы или пользователя.
`YDB_LOG_INFO_CTX_COMP`	Отладочная информация для сбора статистики.
`YDB_LOG_DEBUG_CTX_COMP`	Отладочная информация для разработчиков.
`YDB_LOG_TRACE_CTX_COMP`	Очень подробная отладочная информация.

Перечисленные в таблице макросы не требуют указания аргумента `prio`.

Пример:

```
YDB_LOG_ERROR_CTX_COMP(ctx, EXAMPLE_COMP_CODE, "Unable to open file",
    {"sourceFilePath", filename},
    {"errorCode", err});
```

На основе макроса `YDB_LOG_CTX_COMP` работает множество других макросов, рассматриваемых далее. Их названия строятся по следующей схеме:

1. Имя макроса всегда начинается с `YDB_LOG`.
2. Если макрос не требует указания уровня сообщения, то к имени макроса через символ подчеркивания добавляется название уровня сообщения.
3. Если макрос требует указания контекста исполнения, то к имени макроса добавляется `_CTX`.
4. Если макрос требует указания компонента, то к имени макроса добавляется `_COMP`.

Использование стандартного контекста исполнения

В большинстве случаев для передачи сообщений актору логирования используется контекст исполнения, доступный через глобальную переменную `NActors::TlsActivationContext`. Для того, чтобы исходные тексты не были визуально перегружены обращениями к этой глобальной переменной, введён дополнительный макрос `YDB_LOG_COMP`, который всегда использует `NActors::TlsActivationContext` и не требует указания параметра `CTX`.

Пример:

```
YDB_LOG_COMP(PRI_ERROR, EXAMPLE_COMP_CODE, "Unable to open file",
    {"sourceFilePath", filename},
    {"errorCode", err});
```

Примечание

Построение имени макроса `YDB_LOG_COMP` полностью укладывается в базовый принцип именования макросов, рассмотренный в ядре логирования.

В этом примере используется стандартный контекст исполнения, а логирование происходит от имени компонента с кодом `EXAMPLE_COMP_CODE`.

Макросы без указания уровня сообщения в виде аргумента

Макрос	Уровень сообщения
`YDB_LOG_EMERG_COMP`	Возможен сбой системы (например, отказ кластера).
`YDB_LOG_ALERT_COMP`	Возможна деградация системы, компоненты системы могут выйти из строя.
`YDB_LOG_CRIT_COMP`	Критическое состояние.
`YDB_LOG_ERROR_COMP`	Некритическая ошибка.
`YDB_LOG_WARN_COMP`	Предупреждение, на которое следует отреагировать и исправить, если оно не временное.
`YDB_LOG_NOTICE_COMP`	Произошло событие, существенное для системы или пользователя.
`YDB_LOG_INFO_COMP`	Отладочная информация для сбора статистики.
`YDB_LOG_DEBUG_COMP`	Отладочная информация для разработчиков.
`YDB_LOG_TRACE_COMP`	Очень подробная отладочная информация.

Пример:

```
YDB_LOG_ERROR_COMP(EXAMPLE_COMP_CODE, "Unable to open file",
    {"sourceFilePath", filename},
    {"errorCode", err});
```

Использование компонента по умолчанию

Используется тот же механизм, что и в базовых средствах логирования:

1. В начале файла (но после всех директив `#include`) должен быть определён макрос `YDB_LOG_THIS_FILE_COMPONENT`. Он задаёт код компонента для всего файла.

> **Важно**
>
> Система сборки может объединить несколько файлов `.cpp` в одну единицу компиляции с помощью `JOIN_SRCS`, который подключает их через директивы `#include`. Поэтому в конце каждого исходного файла настоятельно рекомендуется отменять определение макроса:
>
> ```
> #undef YDB_LOG_THIS_FILE_COMPONENT
> ```
>
> В противном случае определение макроса `YDB_LOG_THIS_FILE_COMPONENT` может распространиться на несколько файлов исходного кода.

2. Далее в файле должны использоваться макросы логирования, не требующие указания кода компонента. Они аналогичны уже рассмотренным ранее, но их имя не содержит в себе строку `_COMP`.

Пример:

```
#define YDB_LOG_THIS_FILE_COMPONENT EXAMPLE_COMP_CODE
...
YDB_LOG_CTX(ctx, PRI_ERROR, "Unable to open file",
    {"sourceFilePath", filename},
    {"errorCode", err});
```

Макросы без указания уровня сообщения в виде аргумента

Макрос	Уровень сообщения
`YDB_LOG_EMERG_CTX`	Возможен сбой системы (например, отказ кластера).
`YDB_LOG_ALERT_CTX`	Возможна деградация системы, компоненты системы могут выйти из строя.
`YDB_LOG_CRIT_CTX`	Критическое состояние.
`YDB_LOG_ERROR_CTX`	Некритическая ошибка.
`YDB_LOG_WARN_CTX`	Предупреждение, на которое следует отреагировать и исправить, если оно не временное.
`YDB_LOG_NOTICE_CTX`	Произошло событие, существенное для системы или пользователя.
`YDB_LOG_INFO_CTX`	Отладочная информация для сбора статистики.
`YDB_LOG_DEBUG_CTX`	Отладочная информация для разработчиков.
`YDB_LOG_TRACE_CTX`	Очень подробная отладочная информация.

Пример:

```
#define YDB_LOG_THIS_FILE_COMPONENT EXAMPLE_COMP_CODE
...
YDB_LOG_ERROR_CTX(ctx, "Unable to open file",
    {"sourceFilePath", filename},
    {"errorCode", err});
```

Логирование в заголовочных файлах

> **Важно**
>
> Крайне не рекомендуется определять макрос `YDB_LOG_THIS_FILE_COMPONENT` в заголовочных файлах, поскольку это может приводить к следующим последствиям:
>
> 1. Сложная логика определения кода компонента (не очевидно, из какого заголовочного файла берётся определение `YDB_LOG_THIS_FILE_COMPONENT`).
> 2. Ошибки при компиляции (в разных файлах макрос `YDB_LOG_THIS_FILE_COMPONENT` может быть определён по-разному).

Если в заголовочном файле необходимо логирование, то следует использовать макросы, требующие явного указания кода компонента:

1. `YDB_LOG_CTX_COMP` — требует явного указания уровня сообщения, контекста исполнения и кода компонента.
2. `YDB_LOG_XXXX_CTX_COMP` — не требует указания уровня сообщения.
3. `YDB_LOG_COMP` — использует стандартный контекст исполнения.
4. `YDB_LOG_XXXX_COMP` — использует стандартный контекст исполнения и не требует указания уровня сообщения.

Построение сообщений

Переиспользуемые наборы прикрепляемых параметров

Если один и тот же набор параметров прикрепляется к различным сообщениям, то во избежание дублирования кода можно заранее создать этот набор параметров, а затем его переиспользовать при логировании различных событий. Для этого предназначен макрос `YDB_LOG_CREATE_MESSAGE`. В качестве аргументов он принимает набор параметров (в виде пар `{name, value}`), а в качестве результата возвращает объект, который впоследствии может быть использован при отправке сообщений в журнал.

Пример

```
#define YDB_LOG_THIS_FILE_COMPONENT EXAMPLE_COMP_CODE
...
void MyFunction(const std::string& filename) {
    ...
    // Create message parameters
    auto context = YDB_LOG_CREATE_MESSAGE(
        {"sourceFilePath", filename});
    ...
    if (err != 0) {
        YDB_LOG_ERROR("MyFunction failed",
            context,                            // Use message parameters
            {"errorCode", err});
        return;
    }
    ...
    YDB_LOG_NOTICE("MyFunction succeeded",
        context);                               // Use message parameters
}
```

Результат вызова макроса `YDB_LOG_CREATE_MESSAGE` является экземпляром класса `TStructuredMessage`. Этот экземпляр хранит набор входящих в него параметров, их имена, типы и значения. Экземпляры класса `TStructuredMessage` можно хранить в локальных и глобальных переменных, копировать, передавать и т.д.

> **Примечание**
>
> Можно считать, что класс `TStructuredMessage` является специализированным контейнером для хранения пар `{name, value}`, прикрепляемых к сообщениям журнала.

Поддерживаются операции по изменению ранее созданного набора параметров:

1. Добавление и обновление.

Пример

```
YDB_LOG_UPDATE_MESSAGE(context,
    {"socket", socketNum});
```

2. Удаление параметров.

Пример

```
context.RemoveValue("socket");
```

Вложенные наборы параметров

Возможно создание вложенных наборов прикрепляемых параметров. Для этого при логировании сообщения (или создании другого набора параметров) в паре `{name, value}` в качестве значения должен быть указан ранее созданный набор значений (т.е. экземпляр класса `TStructuredMessage`). Тогда в итоговое сообщение (или создаваемый набор значений) будут автоматически добавлены все параметры из набора `value`, но к именам этих параметров будет добавлен префикс, указанный в `name`, с разделителем `.`.

Пример

```
#define YDB_LOG_THIS_FILE_COMPONENT EXAMPLE_COMP_CODE
...
void MyFunction() {
    // Create message parameters
    auto context = YDB_LOG_CREATE_MESSAGE(
        {"operationName", "read"},
        {"sourceFilePath", filename});
    ...
    if (err != 0) {
        YDB_LOG_ERROR("MyFunction failed",
            {"details", context},                       // Use message parameters
```

```
            {"errorCode", err});
        return;
    }
    ...
}
```

В этом примере к сообщению с текстом `MyFunction failed` будут прикреплены параметры `details.operationName`, `details.sourceFilePath` и `errorCode`.

Контексты логирования

Формирование и переиспользование наборов прикрепляемых параметров неудобно в следующих случаях:

1. Эти наборы нужно указывать при логировании большого количества сообщений.
2. Эти наборы нужно передавать во все вызываемые функции, чтобы там указывать при логировании сообщений.

Можно упростить исходный код, если использовать **контекст логирования** — набор параметров, которые будут автоматически прикрепляться ко всем сообщениям, отправляемым в журнал в данном потоке исполнения. Для этого нужно вызвать макрос `YDB_LOG_CREATE_CONTEXT` и передать ему пары `{name, value}` (так же, как при отправке сообщений в журнал или создании переиспользуемых наборов), а также ранее созданные объекты `TStructuredMessage`. Добавление перечисленных в `YDB_LOG_CREATE_CONTEXT` параметров происходит при логировании в блоке кода, где вызван этот макрос, а также во вложенных блоках кода и всех вызываемых функциях.

Пример использования контекста логирования

```
#define YDB_LOG_THIS_FILE_COMPONENT EXAMPLE_COMP_CODE
...
void MyFunction() {
    YDB_LOG_CREATE_CONTEXT(
        {"sourceFilePath", filename});
    ...
    if (errorCode != 0) {
        YDB_LOG_ERROR("MyFunction failed",     // К сообщению будут прикреплены параметры sourceFilePath
и errorCode
            {"errorCode", errorCode});
        return;
    }
    ...
    YDB_LOG_NOTICE("MyFunction succeeded");    // К сообщению будет прикреплён параметр sourceFilePath
}
```

Параметр `sourceFilePath` будет прикрепляться ко всем сообщениям, логируемым в данном потоке исполнения в функции `MyFunction` и вызываемых ею функциях вплоть до выхода из функции `MyFunction`.

Контексты логирования организованы в виде стека. Если контекст логирования настроен, то он хранится в верхушке стека. У каждого потока исполнения свой стек контекстов логирования.

При вызове `YDB_LOG_CREATE_CONTEXT` происходят действия:

1. Создаётся новый набор параметров (как и при вызове `YDB_LOG_CREATE_MESSAGE`), содержащий все параметры из уже существующего контекста, а также параметры, непосредственно указанные в макросе `YDB_LOG_CREATE_CONTEXT`.
2. Новый набор параметров помещается на верхушку стека и, таким образом, становится текущим контекстом логирования в данном потоке исполнения.
3. При записи сообщения в журнал (вызове макроса `YDB_LOG_CTX_COMP` или т.п.) к сообщению автоматически прикрепляются параметры из верхушки стека контекстов логирования, а поверх этого набора записываются параметры, указанные в макросе логирования.
4. Верхушка стека будет автоматически удалена при выходе из текущего блока, и таким образом произойдёт возврат к предыдущему настроенному контексту.

Чтобы добавить параметры в текущий контекст или обновить их значения, необходимо использовать макрос `YDB_LOG_UPDATE_CONTEXT`:

```
YDB_LOG_UPDATE_CONTEXT({"requestId", requestId});
```

В макросе `YDB_LOG_UPDATE_CONTEXT` может быть указано сразу несколько параметров и их значений. Если указанные параметры уже существуют в контексте, то их значения будут обновлены. Если указанные параметры отсутствуют в контексте, то их значения будут добавлены в контекст.

Чтобы удалить параметры из текущего контекста, необходимо передать их имена в `YDB_LOG_REMOVE_CONTEXT`:

```
YDB_LOG_REMOVE_CONTEXT("requestId");
```

Макросы `YDB_LOG_UPDATE_CONTEXT` и `YDB_LOG_REMOVE_CONTEXT` могут вызываться сколько угодно раз. Они изменяют только текущий контекст (то есть верхушку стека контекстов), но не создают новых контекстов и никак не затрагивают другие контексты, расположенные ниже по стеку контекстов.

Рекомендации по наполнению логов

Безопасность

Запрещается записывать в лог пароли, токены доступа, ключи шифрования, строки подключения и персональные данные. Если значение необходимо для диагностики, то перед передачей в макрос необходимо удалить чувствительную часть, замаскировать её или заменить необратимым хешем.

Стиль кодирования

Рекомендуется следующий стиль кодирования при использовании макросов логирования:

1. Вся обязательная информация (контекст исполнения, код компонента, уровень сообщения и текст сообщения) располагается на одной строке.
2. Каждая пара `{name, value}` располагается на отдельной строке.

Пример записи сообщения с несколькими параметрами:

```
    YDB_LOG_DEBUG("Handle TEvNodeWardenNotifyConfigMismatch",
        {"clusterStateGeneration", ClusterStateGeneration},
        {"msgGeneration", msgGeneration},
        {"clusterStateGuid", ClusterStateGuid},
        {"msgGuid", msgGuid});
```

Стиль текстовых сообщений

Рекомендуется следующий стиль написания текстов сообщений:

1. Текст является осмысленным (и желательно корректным) предложением на английском языке, говорящим о некотором событии внутри системы, при этом:
   - текст начинается с заглавной буквы;
   - если текст представляет собой отдельное предложение, то точка в конце сообщения не ставится;
   - если текст состоит из нескольких предложений, то они разделяются точками, но после последнего предложения точка не ставится.
2. Текст может содержать имена классов и функций. Если актор логирует факт получения некоторого сообщения, то хорошей практикой является логирование с текстом `Handle <имя класса-события>`.
3. Сообщения являются фиксированным текстом без динамически формируемых фрагментов.
4. Вся динамическая информация, известная только на момент исполнения, должна помещаться в виде прикрепляемых параметров.

Для согласования текстов сообщений и названий параметров существует простое эмпирическое правило:

> Если взять текст сообщения и к нему добавить значения параметров в виде `(name=value, name=value, ... )`, то должно получиться осмысленное, законченное, однозначное и понятное человеку текстовое сообщение.

Пример согласованных текста сообщения и названий параметров

```
YDB_LOG_ERROR_CTX(ctx, "Unable to open file",
    {"sourceFilePath", filename},
    {"errorCode", err});
```

Предполагаемое описание события выглядит так (параметры сообщения перечисляются в алфавитном порядке):

```
Unable to open file (errorCode = ..., sourceFilePath = ...)
```

Общие рекомендации по именованию отдельных параметров

Стиль именования отдельных параметров строится на двух базовых принципах:

1. В контексте конкретного сообщения значение параметра должно интерпретироваться просто и однозначно.

Хорошо

```
YDB_LOG_ERROR("Response timeout elapsed",
    {"nodeHostName", hostName},
    {"requestNum", requestNum},
    {"waitAtPosixTime", startTime},
    {"timeoutMs", timeout});
```

Плохо

```
YDB_LOG_ERROR("Response timeout elapsed",
    {"node", hostName},
    {"request", requestNum},
    {"wait", startTime},
    {"timeout", timeout});
```

Если использовать упомянутое ранее правило согласования текстов сообщений и названий параметров, то получится предложение `Response timeout elapsed (node=<строка>, request=<число>, wait=<число>, timeout=<число>)`, где будет непонятно: значение `node` — это имя хоста или некоторое внутреннее название-идентификатор узла? Значение `request` — это порядковый номер или числовой идентификатор? Значение `wait` — что это такое и как интерпретировать это значение? В каких единицах задано значение `wait`?

2. Семантически близкие параметры должны иметь одинаковые названия, поскольку различное именование одних и тех же понятий затрудняет поиск записей в логах. Например, нежелательно в одних сообщениях идентификатор транзакции обозначить как `txId`, а в других — как `transactionId`.

Хорошо

```
YDB_LOG_INFO("Started transaction",
    {"txId", transactionId});
```

```
...
YDB_LOG_INFO("Transaction modifies table",
    {"queryId", queryId},
    {"txId", transactionId});
...
YDB_LOG_INFO("Query commits transaction",
    {"queryId", queryId},
    {"txId", transactionId});
...
```

Плохо

```
YDB_LOG_INFO("Started transaction",
    {"id", transactionId});
...
YDB_LOG_INFO("Transaction modifies table",
    {"id", queryId},
    {"transactionId", transactionId});
...
YDB_LOG_INFO("Query commits transaction",
    {"id", queryId},
    {"txId", transactionId});
...
```

В этом примере идентификатор транзакции записывается в параметры с разными именами (`id` , `transactionId` и `txId`), и в то же время параметр с названием `id` каждый раз хранит идентификаторы совершенно разных сущностей.

3. Названия параметров пишутся в `camelCase` .

Частные рекомендации по именованию отдельных параметров

Далее перечисляются рекомендации, направленные на выполнение двух указанных выше принципов:

1. Нежелательно в качестве названий параметров использовать отдельные общеупотребимые слова (`id` , `item` , `value` , `path` , `child` , `parent` , `min` , `max` , и т.п.) — это осложняет поиск записей в логах. Например, слово `id` обозначает идентификатор, но при этом совершенно непонятно, к какой сущности он относится, хотя в контексте конкретного сообщения смысл этого параметра может быть предельно ясен.

Хорошо

```
YDB_LOG_INFO("Started transaction",
    {"txId", ...});
...
YDB_LOG_INFO("Received query",
    {"queryId", ...});
...
YDB_LOG_INFO("Sent request to actor",
    {"actorId", ...});
```

Плохо

```
YDB_LOG_INFO("Started transaction",
    {"id", ...});
...
YDB_LOG_INFO("Received query",
    {"id", ...});
...
YDB_LOG_INFO("Sent request to actor",
    {"id", ...});
```

2. Названия параметров должны состоять из нескольких слов, причём главным должно быть последнее слово, а каждое предшествующее должно уточнять смысл последующего. Например:

- `actorId` — идентификатор актора;
- `operationId` — идентификатор операции;
- `operationName` — название операции;
- `operationSessionId` — идентификатор сессии, в рамках которой выполняется операция.

Хорошо

```
YDB_LOG_INFO("Copy data to another shard",
    {"srcShardId", ...},
    {"dstShardId", ...});
```

Плохо

```
YDB_LOG_INFO("Copy data to another shard",
    {"srcId", ...},
    {"dstShard", ...});
```

Здесь непонятно ни идентификатором какой сущности является параметр `srcId` , ни как трактовать значение `dstShard` (это идентификатор, название или что-либо ещё).

3. Правила обозначения отдельных сущностей и коллекций:

- отдельное упоминание имени сущности в единственном числе предполагает, что значение параметра содержит описание этой сущности. Например, `table` — это описание таблицы (однако, в данном случае лучше добавлять суффикс, то есть `tableId` явно указывает, что параметр содержит идентификатор таблицы, `tableName` — название таблицы, `tableDesc` — описание таблицы и т.д.);
- отдельное упоминание имени сущности во множественном числе предполагает, что значение параметра содержит описание коллекции сущностей. Например,
  `tables` — это описание нескольких таблиц (но не их количество);
- отдельное упоминание имени сущности во множественном числе с добавлением `Count` предполагает, что значение параметра содержит количество элементов в коллекции. Например, `tablesCount` — количество таблиц;
- нежелательно для обозначения количества элементов в коллекции к названию параметра вместо `Count` добавлять `Size`. Например, по названию параметра `totalFilesSize` — непонятно, речь идёт о количестве файлов или их суммарном размере.

Хорошо

```
YDB_LOG_INFO("Backup volumes progress",
    {"currentVolumeNum", ...},
    {"volumesCount", ...},
    {"volumesSizeBytes", ...});
```

Плохо

```
YDB_LOG_INFO("Backup volumes progress",
    {"currentVolume", ...},
    {"volumes", ...});
```

4. Если в названии параметра используется общепринятая в YDB аббревиатура или термин, то её следует писать в том регистре, в котором принято. Примеры названия таких параметров:

- `VDiskId` — идентификатор VDisk (пишется не как `vdiskId`);
- `PDiskId` — идентификатор PDisk (пишется не как `pdiskId`);
- и т.д.

5. Допустимо использовать популярные сокращения (`idx` вместо `index`, `msg` вместо `message`, `tx` вместо `transaction` и т.д.).
6. Если параметр содержит числовую величину, то желательно к названию параметра добавлять суффикс:

- `Id`, если число является идентификатором (но не как `ID`);
- название единиц измерения (например, для объёмов памяти — `Byte`, `KByte`, `MByte`, для времени — `Ms`, `Sec`, `Min`, для указания процентных долей чего-либо — `Percent` и так далее). Если используется нестандартная единица измерения, то желательно использовать предлог `In` (например, `bufferSizeInBlocks`);

Хорошо

```
YDB_LOG_INFO("Table dump progress",
    {"tableId", ...},
    {"dumpedRecordsCount", ...},
    {"dumpedSizeBytes", ...},
    {"totalSizeBytes", ...});
```

Плохо

```
YDB_LOG_INFO("Table dump progress",
    {"table", ...},
    {"dumpedRecords", ...},
    {"dumpedSize", ...},
    {"totalSize", ...});
```

7. Если числовое значение параметра человеку удобнее анализировать не в десятичном виде, то лучше это значение указывать как символьную строку и дополнять общепринятыми символами (например, у шестнадцатеричного представления должен быть префикс `0x`, а у двоичного — префикс '0b' и т.д.). Хорошей практикой является дополнение названия параметра суффиксом `Hex` или т.п.

Хорошо

```
unsigned flags;
...
YDB_LOG_INFO("Invalid access rights to file",
    {"filename", ...},
    {"aclFlags", flags},
    {"aclFlagsOct", ToOct(flags)});         // Make flags as "0XXXXX" string
```

Плохо

```
unsigned flags;
...
YDB_LOG_INFO("Invalid access rights to file",
    {"filename", ...},
    {"aclFlags", flags});
```

8. Если параметр содержит текстовую информацию, то желательно к названию параметра добавлять суффикс, позволяющий легче интерпретировать значение параметра (например, `Name`, `Desc`, `Json`, `Base64` и т.п.).
9. Для именования параметров можно применять правила именования локальных переменных (например, для логических признаков следует в начало названия параметра добавлять `is`, `has` или т. п.).

Общепринятые названия параметров

Существуют общепринятые названия параметров, которые желательно использовать.

Название	Содержимое параметра
`actorClassName`	Название класса, в котором реализован актор.
`actorStateName`	Состояние актора.
`backtrace`	Стек возникновения исключения (результат функции `TBackTrace::FromCurrentException().PrintToString()`).
`ev`	Описание сообщения, которое обрабатывает актор.
`evType`	Тип сообщения, которое обрабатывает актор.
`exception`	Описание произошедшего исключения C++.
`selfId`	Идентификатор актора, который обрабатывает сообщение.

Методология обогащения логов контекстной информацией

Предлагается следующая методология для обогащения событий, которые пишет в журнал конкретный актор.

Шаг 1. В акторе должна быть реализована функция, которая формирует текущий контекст логирования в виде набора параметров, прикрепляемых к логируемым сообщениям.

Пример реализации

```
TStructuredMessage GetLogContext() const {
    return YDB_LOG_CREATE_MESSAGE(
        {"actorClassName", "TFileReader"},
        {"selfId", SelfId()},
        {"sourceFilePath", filename});
}
```

Вместо имени `GetLogContext` может использоваться `GetLogPrefix` или т.п. В принципе, это может быть даже не функция, а поле класса, создаваемое в конструкторе и изменяемое в ходе работы актора.

Шаг 2. Настройка контекста в "точках входа". У каждого актора есть небольшой набор функций, которые могут быть вызваны ядром акторной системы. Сюда входят:

1. Функция `Bootstrap`.
2. Функции обработки сообщений (ищутся по макросу `STATEFN` или т.п.).
3. Прочие виртуальные функции, которые переопределяет актор (они легко ищутся по ключевому слову `override`).

Далее в первой строке каждой такой функции с помощью макроса `YDB_LOG_CREATE_CONTEXT` должен настраиваться контекст логирования.

Пример настройки контекста

```
STATEFN(StateWork) {
    YDB_LOG_CREATE_CONTEXT(GetLogContext());
    switch (ev->GetTypeRewrite()) {
        ...
    }
}
```

Manage YDB releases

На базе исходного кода из репозитория YDB разрабатываются два продукта с независимыми релизными циклами:

- Сервер YDB
- Интерфейс командной строки YDB (CLI)

Релизный цикл сервера YDB

Данный документ описывает релизный цикл, начиная с мажорного релиза 25.1.

> **Примечание**
>
> С мажорного релиза `25.1` действует схема релизных веток, в которой релизные теги создаются в минорных ветках `stable-XX-Y-Z` (в них можно мерджить только исправления ошибок), а в мажорную ветку `stable-XX-Y` по согласованию можно мерджить новые фичи. Подробнее — в разделе Общая схема работы с ветками. Как устроен номер версии сервера — в статье Версионирование YDB. Актуальный список версий — на Загрузка YDB Open-Source Database.

Если у вас возникли вопросы по этому документу, обращайтесь к команде YDB.

Номера и расписание релизов

Версия сервера YDB состоит из четырех чисел, разделенных точками:

1. Две последние цифры календарного года релиза
2. Порядковый номер мажорного релиза в текущем году
3. Порядковый номер минорного релиза в этом мажорном релизе
4. Порядковый номер патча (релизного тега) в минорном релизе

Таким образом:

- Мажорная версия - это комбинация первых двух чисел (например, `25.1`)
- Минорная версия - это комбинация первых трёх чисел (например, `25.1.2`)
- Полная версия - это комбинация всех четырёх чисел (например, `25.1.2.1`)

Как устроена структура номера версии и какие компоненты могут опускаться в зависимости от контекста, см. в статье Версионирование YDB.

В течение года обычно выпускается 4 мажорных релиза сервера YDB: `YY.1` – первый, а `YY.4` - последний в году `YY`. Количество минорных релизов и патчей не является постоянным и может варьироваться от одного мажорного релиза к другому.

Совместимость

Совместимость версий YDB - это гарантии, что кластер сможет работать, даже если на его узлах выполняются две смежные мажорные версии исполняемого файла сервера YDB. Подробнее о процедуре обновления кластера вы можете прочитать в статье Обновление YDB.

Для обеспечения такой совместимости мажорные релизы выпускаются парами:

- В нечетных версиях добавляется новая функциональность, отключенная с помощью фича-флагов.
- В четных версиях эта функциональность включается по умолчанию.

Например, версия `25.1` поставляется с отключенной новой функциональностью, и может быть постепенно развернута на кластере, работающем под управлением `24.4`, без остановки работы кластера. Как только на всех узлах кластера будет запущена `25.1`, его можно будет далее обновить до `25.2`, чтобы использовать новые функциональные возможности.

Релизные ветки и теги

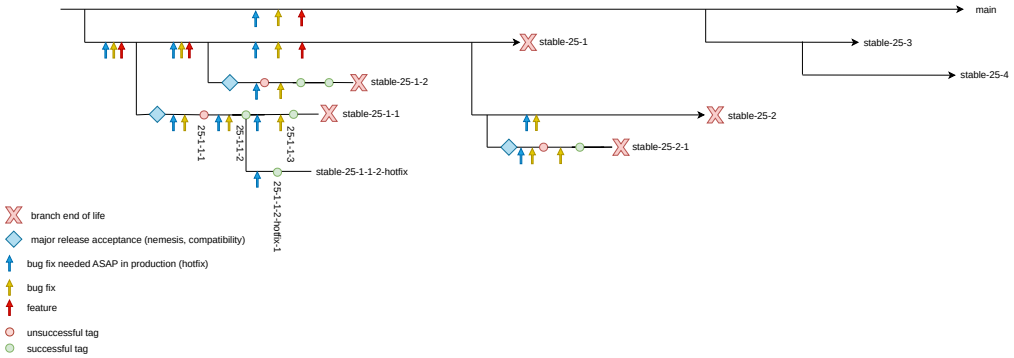
Виды коммитов

- **Фича**. К фичам относятся любые добавляющие новую функциональность или улучшающие существующую изменения, не связанные с исправлением ошибок.
- **Исправление ошибки**. Изменение, направленное на устранение конкретной ошибки.
- **Исправление критической ошибки**. Срочное исправление серьезной проблемы, которое требуется немедленно выкатить в продакшн. Без срочного исправления критических ошибок высока вероятность наступления серьезных негативных последствий.

Типы релизных веток

- **Мажорная ветка** - ветка, в которой хранится исходный код соответствующей мажорной версии. Именуются `stable-XX-Y` (например, `stable-24-1` или `stable-25-1`). В эту ветку можно мерджить новые фичи и исправления ошибок.
- **Минорная ветка** - ветка, из которой собираются релизы YDB. Именуется `stable-XX-Y-Z` (например, `stable-25-1-2`). Коммит, из которого собирается релиз помечается соответствующим релизным тегом, который имеет формат `XX.Y.Z.A`. Новая минорная ветка отводится от мажорной ветки после стабилизации предыдущего минорного релиза. В минорную ветку можно мерджить только исправления ошибок.
- **Хотфиксная ветка** - ветка для срочного исправления критических ошибок в конкретном релизном теге. Именуется `stable-XX-Y-Z-A-hotfix` (где `XX.Y.Z.A` - имя релизного тега), например, `stable-24-1-1-2-hotfix`. Такие ветки отводятся только от релизных тегов при необходимости сделать хотфикс. Из коммита хотфикса создается релизный тег, который именуется `stable-XX-Y-Z-A-hotfix-N` (где `stable-XX-Y-Z-A-hotfix` - имя хотфиксной ветки, N - порядковый номер хотфикса). При необходимости сделать хотфикс поверх ранее сделанного хотфикса, исправление коммитится в ту же хотфиксную ветку и из него создается новый релизный тег. В хотфиксные ветки можно мержить только исправления критических ошибок, которые требуется немедленно выкатить в продакшн.

Общая схема работы с ветками



Цикл выпуска для нечетного мажорного релиза начинается с отведения от `main` мажорной ветки участником команды YDB. Название релизной ветки начинается с префикса `stable-`, за которым следует мажорная версия с точками, замененными на дефис (например, `stable-25-1`).

Цикл выпуска четного мажорного релиза (например, `stable-25-2`) начинается с отведения ветки от ветки предыдущего нечетного мажорного релиза.

От мажорной ветки отводится минорная ветка. Все выпуски минорных версий для нечетных и четных мажорных релизов, проходят цикл тестирования, в ходе которого выпускается ряд патчей. Каждый патч фиксируется назначением релизного тега с полным номером версии. Таким образом, в минорной ветке `stable-25-1-1` могут быть теги `25.1.1.1`, `25.1.1.2` и т.д. Как только тег успешно прошел необходимое тестирование, мы считаем его стабильным, регистрируем релиз на GitHub, добавляем на страницы загрузки и в список изменений. Для мажорной версии может быть более одного стабильного релиза.

Тестирование

Каждая минорная версия проходит приемочное тестирование — комплексный процесс проверки соответствия требованиям качества. Тестирование включает в себя оценку производительности по стандартам (TPC-C, TPC-H), проверку совместимости между версиями и другие критически важные тесты. Последующее тестирование минорной версии включает выкладку на внутренние кластера и является итеративным. Каждая итерация начинается с назначения релизного тега на коммит минорной ветки. Например, тег 25.1.1.3 отмечает 3-ю итерацию тестирования для минорной ветки 25.1.1. Основываясь на выявленных проблемах, релизная команда YDB решает, можно ли считать тег стабильным, или необходимо запустить новую итерацию тестирования.

Стабильный релиз

Если итерация тестирования подтверждает качество минорного релиза, релизная команда YDB готовит перечень изменений, и публикует релиз на страницах Releases на GitHub и Загрузки в документации, объявляя его тем самым стабильным.

Цикл выпуска YDB CLI (интерфейс командной строки)

Номера выпусков и расписание

Версия YDB CLI состоит из трех чисел, разделенных точкой:

1. Порядковый номер мажорного релиза (в настоящее время "2")
2. Порядковый номер минорного релиза в этом мажорном релизе
3. Порядковый номер патча (релизного тега) в минорном релизе

Например, версия `2.8.0` обозначате вторую мажорную, восьмую минорную версию, без патчей.

В отличие от сервера YDB, для CLI не существует фиксированного расписания выпуска релизов. Новые минорные версии публикуются по мере готовности значимых улучшений или новой функциональности. При первоначальном выпуске минорной версии номер патча равен 0.

Процесс выпуска YDB CLI значительно упрощен по сравнению с сервером YDB, что позволяет выпускать релизы чаще.

Релизные теги

Теги для YDB CLI назначаются в транке (ветка `main`) участником релизной команды YDB после запуска тестов для некоторой ревизии. Чтобы отличаться от тегов сервера YDB, теги CLI YDB содержат префикс `CLI_` перед номером версии, например CLI_2.8.0.

Стабильный релиз

Чтобы объявить тег YDB CLI стабильным, участник релизной команды YDB готовит перечень изменений, и публикует релиз на странице GitHub Releases и в разделе Загрузки документации.

Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

Участие в разработке документации YDB

YDB следует подходу «Документация как код», что означает, что документация YDB разрабатывается с использованием аналогичных методов и инструментов, как и основной исходный код проекта на C++.

Исходный код документации состоит из файлов Markdown и конфигурационных файлов YAML, расположенных в папке ydb/docs основного репозитория YDB на GitHub. Компилятором этого исходного кода является инструмент с открытым исходным кодом под названием Diplodoc. Подробности о его диалекте Markdown, параметрах конфигурации, расширениях и многом другом можно найти в его документации.

Процесс предложения изменений в исходном коде документации в основном аналогичен изменению любого другого исходного кода YDB, поэтому большая часть Процесс разработки: работа над изменениями кода YDB применима. Основные дополнительные особенности:

- Для pull-запросов к документации выполняются дополнительные прекоммитные проверки. Одна из этих проверок оставляет комментарий со ссылкой на онлайн-предпросмотр изменений или список ошибок.
- Процесс ревью кода включает дополнительные шаги. См. Процесс ревью документации YDB и Руководство по стилю документации YDB.
- Для небольших изменений, таких как исправление опечатки, можно использовать функцию «Edit this file» в веб-интерфейсе GitHub. Каждая страница документации имеет ссылку «Edit on GitHub» (представлена значком карандаша в правом верхнем углу), которая ведёт к исходному коду страницы в веб-интерфейсе GitHub.

После мёрджа pull-запроса с документацией в ветку `main` CI/CD-пайплайн автоматически публикует её на сайте YDB. Документация также автоматически публикуется для стабильных версий сервера YDB из веток `git` с именами `stable-*`, где разрабатываются эти версии. Если код C++ и документация для функции были закоммичены отдельно, и между этими коммитами была создана новая стабильная ветка, может потребоваться бекпортирование некоторых изменений в стабильную ветку. То же самое относится к исправлению опечаток и другим «багфиксам» в содержании документации. Подробнее о процессе релиза YDB см. Manage YDB releases.

См. также

- Структура документации YDB
- Жанры документации YDB
- Документирование функциональности под фича-флагами
- Документация GitHub
- Документация Git
- Добавление медиа в публичные материалы
- Документирование новых команд CLI

Процесс ревью документации YDB

Развивая общий обзор из Участие в разработке документации YDB, эта статья подробнее рассматривает, что происходит на этапе ревью pull-запроса с документацией.

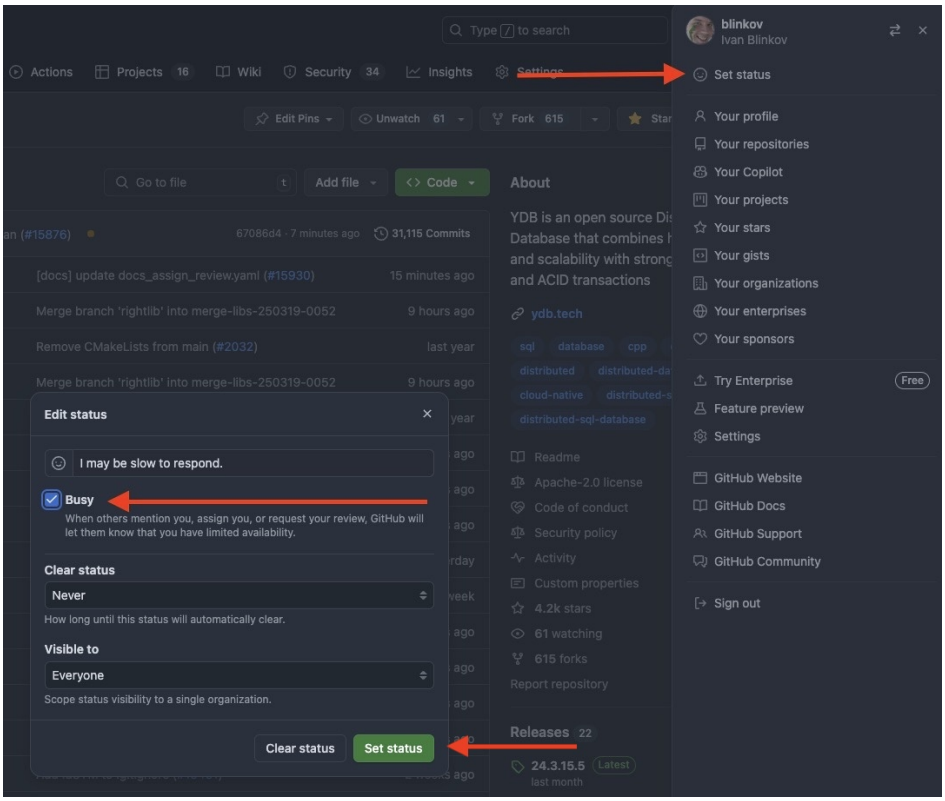
Роли

- **Автор** — человек, предлагающий изменения в документации.
- **Основной ревьюер** — человек, тщательно проверяющий предложенные изменения согласно контрольному списку.
- **Финальный ревьюер** — человек, повторно проверяющий изменения с использованием того же контрольного списка и утверждающий pull-запрос на GitHub.

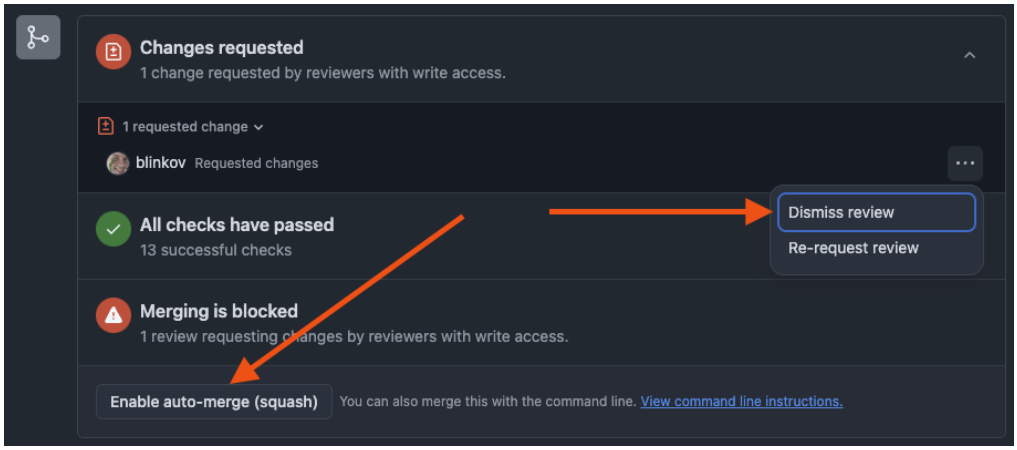
Процесс

1. **Автор** открывает pull-запрос на GitHub с предложенными изменениями в папке ydb/docs. Следование руководству по стилю с самого начала делает процесс ревью более гладким.
2. **Автор** обеспечивает, чтобы pull-запрос был в состоянии, готовом к ревью, соответствуя всем следующим критериям:
   2.1. Pull-запрос *не* помечен как черновик.
   2.2. Предложенные изменения успешно собираются, и автоматизация оставила комментарий со ссылкой на предпросмотр (вместо ошибок). Предпросмотр показывает изменённый контент как ожидается.
3. *(опционально)* **Автор** делится ссылкой на pull-запрос в чате сообщества или чате, связанном с документацией, для привлечения дополнительного внимания к нему.
4. **Основной ревьюер** назначается автоматически или выбирает pull-запрос из входящего списка с помощью кнопки «assign yourself» в правом сайдбаре, а затем предоставляет начальную обратную связь и набор предложений по улучшению.

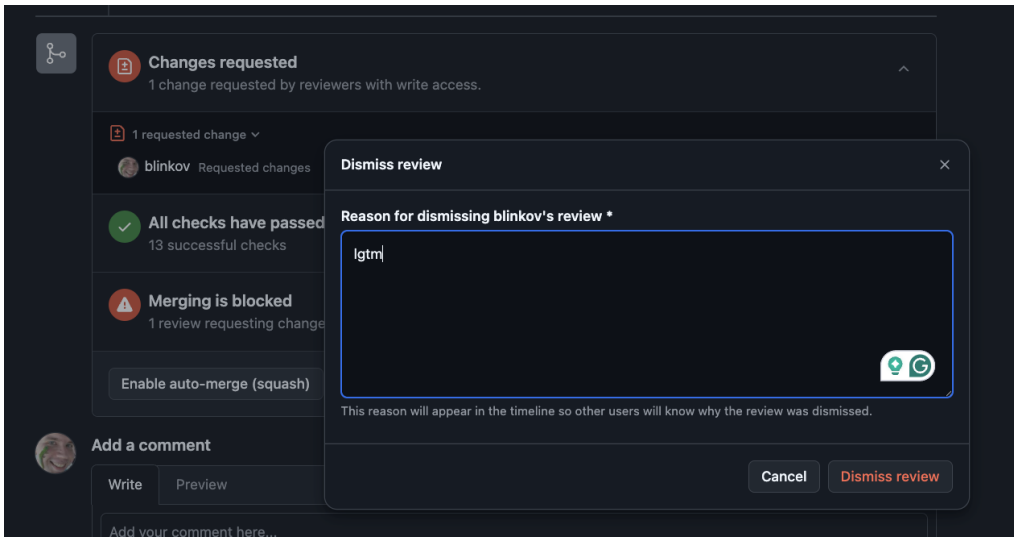
   Временное исключение себя из автоматического назначения, если вы являетесь основным ревьюером



5. **Автор** и **основной ревьюер** совместно итеративно работают, пока предложенные изменения не пройдут контрольный список. **Основной ревьюер** предоставляет отзывы через комментарии к pull-запросу, а **автор** их учитывает. Ожидаемое время ответа для каждой итерации ревью составляет два рабочих дня, до нескольких недель в случае форс-мажора. Также **основной ревьюер** периодически пересматривает назначенные на себя pull-запросы и уточняет с авторами статус, если несколько рабочих дней в них ничего не происходит (в комментариях на GitHub, плюс по возможности в личной переписке).
6. Как только **основной ревьюер** подтверждает, что pull-запрос соответствует требованиям контрольного списка, он:
   - Включает auto-merge для pull-запроса. Если целевая ветка pull-запроса отличается от `main`, функция auto-merge будет недоступна: в этом случае **основной ревьюер** должен вручную нажать кнопку «Squash and merge» после одобрения **финальным ревьюером** на этапе 7.



- Отклоняет свой устаревший отзыв с комментарием «lgtm».



- **Основной ревьюер** передаёт процесс **финальному ревьюеру** для дополнительной проверки свежим взглядом. Как правило, это происходит личным сообщением, но т.к. весь этап 6 делается одновременно — по самому pull-запросу это событие всё равно видно по предыдущим подпунктам (включенному auto-merged и/или отклонённому отзыву **основного ревьюера**).

7. В зависимости от вердикта **финального ревьюера**:
   - Если **финальный ревьюер** нажимает кнопку Approve, pull-запрос начинает соответствовать одному из обязательных условий для мёрджа. Таким образом, если сборка всё ещё проходит, auto-merge GitHub, вероятно, автоматически вмёрджит pull-запрос. В противном случае любые проблемы должны быть устранены вручную.
   - Если **финальный ревьюер** предоставляет дополнительную обратную связь или предложения, процесс возвращается к шагу 5.
8. Если auto-merge не был включен на этапе 6, **основной ревьюер** нажимает кнопку «Squash and merge».
9. После попадания контента в ветку `main`, он будет автоматически опубликован на официальный сайт YDB посредством CI/CD.
10. Документация YDB многоязычная, и от **авторов** ожидается предоставление синхронизированных изменений для всех поддерживаемых языков (в настоящее время английский и русский), если это применимо. Если **автор** не знает все необходимые языки, допустимо использование LLM или машинного перевода. На каком этапе делать перевод зависит от сложности:

- Для простых изменений обычно лучше перевести в начале и пройти процесс ревью с одним pull-запросом, содержащим изменения на всех языках.
- Для сложных изменений, вероятно требующих нескольких итераций ревью, допустимо сначала завершить процесс ревью на одном языке, а затем начать отдельный pull-запрос с переводом после утверждения первого.

Контрольный список

Если PR добавляет новый контент в ветку main

- [ ] Текст понятен целевой аудитории статьи.
- [ ] Текст технически точен.
- [ ] Текст грамматически правильный, без пунктуационных, орфографических или типографских ошибок.
- [ ] Терминология консистентна. Первое упоминание каждого термина, используемого в статье, является ссылкой на его объяснение в Глоссарий YDB или известном доверенном источнике, таком как Википедия.
- [ ] Каждая новая статья правильно размещена в структуре документации.
- [ ] Каждая статья следует одному жанру и соответствует своему месту в структуре документации.
- [ ] Каждая новая статья включает ссылки на все релевантные существующие страницы документации, либо по ходу текста, либо в конце в разделе «См. также».
- [ ] Релевантные существующие статьи обновлены ссылками на новые статьи.
- [ ] Все новые статьи перечислены в YAML файлах с оглавлением и `index.md` их папки.
- [ ] Все переименованные или перемещённые статьи отражены в redirects.yaml.
- [ ] Тон повествования и стиль статьи соответствуют остальной документации или, как минимум, остаются консистентными внутри статьи.

> **Совет**
>
> Этот контрольный список является сжатой версией Руководство по стилю документации YDB и служит напоминанием при каждом ревью. Его можно копировать и вставлять в описание pull-запроса и отмечать пункты по мере продвижения процесса ревью. Обращайтесь к полному руководству по стилю для изначального понимания этих пунктов и дополнительных деталей.

Если PR является обратным переносом в стабильную ветку

Содержимое документации публикуется на сайте из нескольких Git-веток независимо. Аналогично разработке исходного кода, ветка `main` содержит документацию для будущих релизов, а документация для каждого конкретного выпуска YDB публикуется из стабильной ветки. Например, версии YDB v25.1 соответствует ветка `stable-25-1`. Если необходимо перенести документацию из ветки `main` в стабильную ветку через cherry-pick, создаётся отдельный PR с целевой веткой `stable-***`. Процесс проверки таких PR отличается, так как содержимое уже прошло ревью при слиянии в ветку `main`. В таком случае используйте следующий чек-лист:

- [ ] В PR для обратного переноса указаны идентификаторы или ссылки на один или несколько PR, в которых данный контент был добавлен в ветку `main`.
- [ ] Содержимое PR для обратного переноса совпадает с оригинальными PR в ветку `main`. Если структура документации была изменена, содержимое перенесено корректно.
- [ ] PR корректно собирается, предварительный просмотр формируется без ошибок и предупреждений, контент отображается в правильных местах.
- [ ] Отсутствуют артефакты слияния, такие как дублирующийся контент, Git-маркеры типа `>>>>>>>>` и другие аналогичные проблемы.
- [ ] Функциональность, описанная в документации, доступна в соответствующем релизе YDB. В случае сомнений уточните у автора или ответственного за эту функциональность.

Чем не является ревью документации

Тестирование

Ревью документации не является заменой тестирования. Если документация включает инструкции, **автор** несёт ответственность за обеспечение их правильности, реализацию автоматизированных тестов для поддержания точности со временем и т.д.

Основной ревьюер и/или **финальный ревьюер** могут проверить корректность описанных шагов в инструкции. Но это не является обязательной частью их роли.

Ревью технического дизайна

Ревью документации не является ревью технического дизайна функциональности. Документация обычно пишется для по большей части реализованной функциональности, поэтому значительные изменения в поведении продукта на этом этапе редко возможны. Однако **основной ревьюер** и/или **финальный ревьюер** могут подсветить любые неконсистентности, странное поведение или проблемы с удобством использования. Устранить их сразу же, если это возможно, или учесть их для будущих итераций развития описываемой функциональности является ответственностью **автора**.

См. также

- Документация GitHub
- Документация Git

Руководство по стилю документации YDB

Руководство по стилю документации YDB разработано, чтобы помочь авторам создавать понятную, последовательную и удобную для разработчиков документацию.

Основные принципы

- **Понимание целевой аудиторией:**
  - Перед началом работы над новой статьёй или улучшением существующей уделите время на определение её целевой аудитории. Учитывайте их конкретную роль (разработчики приложений, администраторы кластеров, инженеры по безопасности и т.д.), технический опыт и знакомство с темой статьи и YDB в целом.
  - Убедитесь, что текст понятен целевой аудитории статьи.
- **Чёткий язык:**
  - Используйте простой, понятный язык, который напрямую передаёт мысли автора.
  - Избегайте сложных фраз, которые могут быть трудны для неносителей языка.
  - Избегайте жаргона и сленга, если они не являются общепринятыми в отрасли.
- **Последовательная терминология:**
  - Для каждого необычного термина в статье убедитесь, что первое использование содержит ссылку на объяснение:
    - Для терминов, специфичных для YDB, ссылайтесь на Глоссарий YDB. Если в глоссарии перечислено несколько синонимов термина, используйте основной из заголовка глоссария. Если запись в глоссарии отсутствует, добавьте её.
    - Для общих терминов, которые могут быть незнакомы некоторым читателям, ссылайтесь на их страницу в Википедии или аналогичном общеизвестном источнике. Избегайте ссылок на блоги, социальные сети или случайные сторонние статьи.
  - Если в тексте есть аббревиатура, при первом появлении в статье напишите её полностью и явно покажите аббревиатуру в скобках. Например, Structured Query Language (SQL).
- **Структурированное содержание:**
  - Каждая статья должна иметь только *одну* цель для достижения читателем. Если текст преследует несколько целей, вероятно, его нужно разделить на несколько статей. Если возможно, явно укажите эту цель в начале статьи и продемонстрируйте её достижение в конце.
  - Каждая статья должна следовать только одному жанру. Если это не так, то вероятно её нужно разделить на несколько статей.
  - Организуйте статьи с помощью заголовков, подзаголовков, списков, заметок, таблиц и блоков кода, чтобы помочь читателям быстро ориентироваться в информации.
  - Следуйте общей структуре документации при создании новых статей.

Тон повествования

- **Разговорный.** Стремитесь к тону, который ощущается как объяснение концепций опытным другом в доступной манере.
- **Дружелюбный и уважительный.** Сохраняйте баланс профессионализма и дружественной теплоты.
- **Инклюзивный язык.** Пишите нейтрально и уважительно, избегая предвзятого или исключающего языка.
- **Контекстно-зависимый.** Прдстраивайте тон текста в зависимости от типа контента. Для практических руководств будьте более ободряющими; для справочного материала будьте более точными и краткими.
- **Активный залог.** Предпочитайте активные конструкции для обеспечения ясности и непосредственности, облегчая следование инструкциям.

Специфика языка

Убедитесь, что текст следует правилам своего языка без опечаток, грамматических, пунктуационных или орфографических ошибок. Кроме того, следуйте этим специфичным для YDB правилам.

Только для английского

- Используйте последовательную запятую где это уместно.
- Используйте заголовочный регистр для заголовков, предпочтительно следуя правилам Chicago Manual of Style.
  В качестве кавычек в тексте используйте простые двойные кавычки (").

Только для русского

- Используйте `ё` где это уместно.
- Используйте кавычки-ёлочки (`«` и `»`).
- Используйте `–` (en dash) для тире.
- Используйте точку с запятой для маркированных и нумерованных списков где это уместно, т.е. для всех элементов, кроме последнего, если элементы не содержат нескольких предложений.
- *Не* используйте заголовочный регистр для заголовков.

Форматирование и структура

- **Чёткая иерархия.** Используйте заголовки и подзаголовки для определения разделов и облегчения просмотра документа.
- **Списки и маркеры.** Используйте маркеры или нумерованные списки для разбивки шагов, функций или рекомендаций.
- **Визуальные выделения.** Если есть важное предупреждение, информация или совет, выделите их с помощью тега `{% note info %} ... {% endnote %}`. Для небольших выделений по тексту используйте **жирный** или *курсив*, но в меру.
- **Код и примеры.** При включении кода используйте правильное выделение синтаксиса, форматирование и комментарии для обеспечения читаемости. Показывайте примеры вывода для запросов и CLI-команд. Используйте `блоки кода` для всего, что может появиться в консоли или IDE, но не для визуальных выделений.
- **Именование файлов.** Используйте дефисы вместо подчёркиваний для пробелов в именах новых файлов и папок (например, `new-folder/new-file.md` вместо `new_folder/new_file.md`). Тем не менее, подчёркивания допустимы в ключевых словах, которые должны писаться именно с подчёркиваниями, например в названиях настроек.
- **Ссылки.**
  - Предоставляйте ссылки на релеватные ресурсы или дополнительную документацию.
  - Ссылки внутри документации всегда должны включать суффикс `.md` и быть относительными (то есть без префикса `https://ydb.tech`); в противном случае проверка согласованности ссылок не работает, и со временем они начнут вести к ошибкам 404. Внутренние ссылки не должны содержать параметры вида `?version=...`, `?revision=...` или аналогичные.

    - Вместо повторения заголовка целевой статьи для внутренних ссылок используйте `[{#T}](path/to/an/article.md)`.
- **Без копирования.** Если ссылок не достаточно и часть информации должна отображаться в документации более одного раза, создайте отдельный файл Markdown в папке `_includes`, затем добавьте его во все необходимые места через `{% include [...](_includes/path/to/a/file.md) %}` вместо дублирования контента копированием и вставкой.
- **Стиль синтаксиса Markdown.** Документация YDB использует автоматический линтер для файлов Markdown. Обратитесь к .yfmlint для актуального списка применяемых правил.
- **Использование переменных.** Документация YDB имеет конфигурационный файл presets.yaml, который перечисляет переменные для предотвращения опечаток или условного скрытия контента. Используйте эти переменные где это уместно, особенно `YDB` вместо `YDB`.
- **Диаграммы.** Используйте встроенные диаграммы Mermaid, если это возможно. Если используется сторонний инструмент для диаграмм, сохраните исходный файл в ту же папку `_assets` рядом с изображением для будущих правок. Убедитесь, что диаграммы хорошо выглядят как в светлой, так и в тёмной теме.
- **Правильное размещение статей.** Новые статьи должны быть правильно размещены в соответствии со структурой документации.
- **Консистентность жанра.** Статьи не должны смешивать несколько жанров, и жанр должен соответствовать месту статьи в структуре документации.

Интеграция документации

- **Перекрёстные ссылки.** Добавляйте ссылки на все релевантные существующие страницы документации, либо по ходу текста, либо в разделе «См. также» в конце.
- **Обратные ссылки.** Обновляйте релевантные существующие статьи ссылками на новые статьи.
- **Включение в индекс.** Упоминайте все статьи в оглавлении (`toc_i.yaml` или `toc_p.yaml`) и `index.md` их папки.
- **Ссылки на исходный код.** Ссылайтесь напрямую на исходные файлы на GitHub когда это уместно. Если цель ссылки может значительно измениться со временем, используйте ссылку на конкретный коммит или стабильную ветку.
- **Ссылки на глоссарий:**
    - Когда термин, специфичный для YDB, упоминается в статье впервые, сделайте его кликабельной ссылкой на соответствующую запись глоссария.
    - Если есть отдельная подробная статья по той же теме, что и термин глоссария, добавьте ссылку на неё из описания термина глоссария.

Использование и гибкость

- **Рекомендации, а не жёсткие правила.** Это руководство по стилю предлагает рекомендации для улучшения ясности и последовательности, но допускает гибкость, когда отклонения приносят пользу контенту.
- **Дополнительные ресурсы.** Авторам рекомендуется консультироваться с внешними руководствами по стилю (например, The Chicago Manual of Style, Merriam-Webster) при неоднозначных ситуациях.
- **Обновления.** Это руководство эволюционирует со временем вместе с остальным контентом. Если вы часто вносите вклад в документацию YDB, периодически возвращайтесь сюда, чтобы ознакомиться с обновлениями.

Чего следует избегать

- Жаргона и сленга.
- Использования «мы».
- Шуток и другого эмоционального контента.
- Чрезмерно длинных предложений.
- Нерелевантных тем и отсылок, таких как культура, религия или политика.
- Излишне подробных деталей реализации (кроме Разработка YDB, Список изменений YDB Server и Видеозаписи).
- Повторного использования стороннего контента без письменного разрешения или совместимой открытой лицензии, явно разрешающей это.
- HTML внутри Markdown, кроме как для обхода отсутствующих визуальных стилей.

См. также

- Участие в разработке документации YDB
- Процесс ревью документации YDB
- Структура документации YDB
- Жанры документации YDB

Структура документации YDB

Эта статья дополняет Руководство по стилю документации YDB. Она объясняет текущие директории верхнего уровня документации и какой контент принадлежит каждой из них. Как правило, большинство разделов верхнего уровня фокусируются либо на конкретной целевой аудитории (если названы «Для ...»), либо на конкретном жанре.

- Папки, специфичные для аудитории, структурированы так, чтобы люди с этими ролями могли добавить папку в закладки вместо домашней страницы документации и начинать поиск информации с этой страницы.
- Папки, специфичные для жанра, в основном разработаны для поиска через встроенный поиск документации, сторонние поисковые системы или LLM.

Введение новых папок верхнего уровня возможно, но требует тщательного рассмотрения для минимизации загромождения сайдбара и неоднозначности в отношении того, где должны размещаться статьи.

Общие правила

- Документация YDB многоязычная, и структура файлов для каждого языка должна быть согласованной для корректной работы переключателя языков. Содержимое файлов также должно быть максимально близким между языками. Единственным исключением является папка «Публичные материалы», которая намеренно содержит разный контент для разных языков. Все остальные расхождения между языками считаются техническим долгом.
- Поддерживайте консистентность между структурой файлов в репозитории, URL-адресами в адресной строке и папками в боковой панели. Исторически они были независимыми, но опыт показал, что несогласованности приводят к путанице и проблемам с навигацией. Документация постепенно переходит к единой структуре.
- Если имя файла или папки содержит несколько слов, используйте `-` вместо `_` в качестве разделителя, если имя не является ключевым словом, включающим подчёркивания (например, имена разделов конфигурации).
- При переименовании или перемещении любой статьи обязательно добавьте перенаправление со старого URL на новый в redirects.yaml.

Список папок верхнего уровня

- **Начало работы.** Руководство для начинающих, объясняющее, как настроить одноузловой сервер YDB и выполнить на нём первые запросы.
- **Концепции.** Высокоуровневый теоретический обзор YDB как технологии, охватывающий его функции, терминологию и архитектуру. Предназначен для широкой аудитории с минимальными предварительными знаниями, включая руководство компаний и лиц, принимающих решения.
- **Администрирование кластеров.** Папка для инженеров, отвечающих за настройку и эксплуатацию кластеров YDB. Большинство контента состоит из практических руководств для конкретных задач, связанных с кластером. Поскольку YDB поддерживает несколько вариантов развёртывания, руководства, различающиеся в зависимости от метода развёртывания, размещаются в соответствующих подпапках (Ansible, Kubernetes или вручную), каждая из которых следует согласованной внутренней структуре. Здесь также включена теоретическая информация, специфичная для этой роли.
- **Для разработчиков приложений.** Папка для разработчиков приложений, работающих с YDB. В основном состоит из практических руководств и теории.
- **Для инженеров безопасности.** Папка для инженеров по безопасности, отвечающих за защиту и аудит кластеров YDB и приложений, взаимодействующих с ними. Содержит в основном практические руководства и теорию, специфичную для роли.
- **Для контрибьюторов.** Папка для участников основной команды разработки самого YDB и внешних контрибьюторов. Она объясняет различные процессы вокруг разработки YDB и предоставляет более глубокое понимание того, как работают некоторые компоненты. В основном теория с несколькими практическими руководствами.
- **Справка.** Подробный раздел со справкой по различным аспектам YDB. Подразумевается, что читатели будут попадать сюда в основном через поиск по мере необходимости или ссылки из других разделов. Основная цель — полнота, чтобы любая тема могла быть найдена через точные совпадения ключевых слов или их описания. Три основных сценария использования этого раздела:
  - Поиск значения незнакомых ключевых слов, функций, настроек, аргументов и т.д.
  - Поиск правильного синтаксиса для запросов, работы с SDK или конфигурационных файлов.
  - Использование как цели для внутренних ссылок, когда другие статьи документации упоминают функциональность без подробного объяснения.
- **Рецепты.** Мини-руководства, объясняющие конкретные задачи с YDB, часто с примерами и фрагментами кода. Эта папка существует в основном по историческим причинам, так как большая часть её содержимого могла бы быть размещена либо в папках «Для ...», либо в «Вопросы и ответы».
- **Диагностика проблем.** Смесь теории о потенциальных проблемах, возможных при работе кластерами YDB и взаимодействующими с ними приложениями, а также практических руководств для диагностики и их устранения.
- **Вопросы и ответы.** Раздел в стиле StackOverflow с часто задаваемыми вопросами. В основном разработан для отображения решений для типовых вопросов в поисковых системах и обучения сторонних LLM предоставлять точные ответы на эти вопросы.
- **Публичные материалы.** Коллекция ссылок на видео и статьи о YDB. Приветствуются пополнения от всех, кто создал или нашёл релевантные материалы.
- **Загрузки.** Коллекция ссылок для скачивания исполняемых файлов YDB.
- **Список изменений.** Заметки о выпуске для каждой новой версии сервера YDB и других связанных приложений.

Жанры документации YDB

Эта статья дополняет Руководство по стилю документации YDB, описывая основные жанры статей, используемые в документации YDB. Понимание этих жанров помогает контрибьюторам размещать новый контент в соответствующем разделе и поддерживать консистентную структуру материалов.

Теория

Основная цель для читателя: построить прочный фундамент знаний и прийти к пониманию основных концепций, архитектуры и принципов, на которых работает YDB.

Теоретическая документация:

- вводит ключевые концепции и терминологию;
- объясняет, как система работает как с точки зрения пользователя, так и изнутри;
- предоставляет высокоуровневые обзоры компонентов системы;
- помогает пользователям понять «почему» за архитектурными решениями;
- может быть нацелен либо на широкую аудиторию с минимальными предварительными знаниями, либо на конкретную роль;
- может включать диаграммы и другие визуализации для помощи в передаче информации.

Теоретическая документация в основном находится в разделе Концепции YDB, но также появляется и в папках, специфичных для ролей, когда теоретическая информация релевантна только для конкретной аудитории.

Руководство

Основная цель для читателя: выполнить конкретную практическую задачу или реализовать определённое решение с помощью YDB, следуя инструкциям.

Руководства — это практические пошаговые инструкции, которые помогают пользователям достичь конкретной цели с помощью YDB. Каждая статья в этом жанре:

- предоставляет чёткую цель и последовательные инструкции для её достижения;
- включает конкретные примеры и команды;
- фокусируется на практической реализации;
- освещает конкретные сценарии использования;
- может включать скриншоты для иллюстрации шагов.

Руководства в основном находятся в папках, специфичных для ролей, таких как Администрирование кластеров, YDB для разработчиков приложений и YDB для инженеров по безопасности, а также в разделе Диагностика проблем.

Справочник

Основная цель для читателя: найти дополнительную информацию по конкретной узкой теме, связанной с YDB.

Справочная документация предоставляет всестороннюю, подробную информацию о компонентах YDB, запросах, API, параметрах конфигурации, CLI-командах, страницах UI и многом другом. Этот жанр:

- стремится к полноте и точности;
- служит ресурсом для поиска конкретных деталей;
- документирует все доступные опции, параметры и настройки;
- организован для быстрого поиска информации через поисковую систему или LLM;
- включает синтаксис, типы данных, параметры, возвращаемые значения, значения по умолчанию, конфигурацию и примеры.

Справочная документация разработана для поиска по мере необходимости и является самым подробным уровнем документации. Она особенно полезна, когда пользователям нужна конкретная информация о функциях, настройках или ключевых словах. Этот контент в основном находится в разделе Справка по YDB.

FAQ

Основная цель для читателя: быстро найти ответы на распространённые вопросы, возникающие при работе с YDB.

Документация часто задаваемых вопросов (FAQ) отвечает на общие вопросы о YDB в формате прямых вопросов и ответов на них. Этот жанр:

- обращается к конкретным, часто задаваемым вопросам;
- предоставляет краткие, сфокусированные ответы;
- организован по темам или категориям;
- помогает пользователям быстро находить решения типовых проблем;
- оптимизирован для обнаружения поисковыми системами или LLM.

Контент FAQ в основном находится в разделе Вопросы и ответы про YDB и разработан для помощи пользователям, которые ищут конкретные решения для типовых ситуаций.

Рецепт

Основная цель для читателя: реализовать конкретное, сфокусированное решение для типовой проблемы или сценария использования с помощью YDB.

Рецепты — это короткие, сфокусированные мини-руководства, которые демонстрируют, как выполнять конкретные задачи с помощью YDB. Этот жанр:

- предоставляет краткие решения для конкретных проблем;
- включает фрагменты кода и примеры;
- акцентирует внимание на практической реализации;
- более узко сфокусирован, чем полноценные руководства;
- часто следует формату проблема-решение.

Рецепты в основном находятся в разделе Рецепты по работе с YDB, хотя похожий контент может также появляться в папках, специфичных для роли.

Заметки о выпуске

Основная цель для читателя: быть в курсе новых функций, улучшений, исправлений ошибок и критических изменений в выпусках YDB.

Заметки о выпуске документируют изменения, улучшения и исправления в каждой новой версии YDB. Этот жанр:

- перечисляет новые функции и улучшения;
- документирует исправления ошибок и решённые проблемы;
- выделяет критические изменения и устаревшие функции;
- предоставляет инструкции по обновлению при необходимости;
- организован хронологически по номеру версии.

Заметки о выпуске находятся в разделе Список изменений YDB Server и помогают пользователям понять, что изменилось между версиями, и решить, обновляться ли.

Коллекция ссылок

Основная цель для читателя: открыть для себя дополнительные ресурсы, учебные материалы и внешний контент, связанный с YDB.

Коллекции ссылок предоставляют курированные списки ресурсов, связанных с YDB. Этот жанр:

- агрегирует релевантные сторонние или внутренние ресурсы;
- предоставляет краткие описания каждого связанного ресурса;
- может быть организован по темам, форматам или степени релевантности;
- помогает пользователям открывать для себя дополнительные учебные материалы;
- может включать видео, статьи, загрузки и другой контент.

Коллекции ссылок в основном находятся в разделах Видеозаписи и Загрузки YDB и выступают входными точками к внешним ресурсам о YDB.

Добавление медиа в публичные материалы

Эта статья посвящена размещению медиа на странице «Публичные материалы»

Видео

Название

Первым шагом при публикации видеоматериалов является выбор заголовка. Как правило, это заголовок второго уровня, а название полностью идентично названию видео с той площадки, откуда пришла ссылка первоисточника.

Тег

У каждого видео должен быть свой тег, отражающий категорию и краткое содержание. Все виды тегов расписаны в папке репозитория `/public-materials/_includes/tags.md` и на странице. Тег ставится следующим после названия.



Краткое описание

Следующий шаг - небольшое описание в один абзац, отражающее лишь самые основные моменты видео. Необязательно отсматривать видео полностью самостоятельно, можно воспользоваться функцией саммаризации видео с помощью ИИ, однако помните, что ИИ могут совершать ошибки и лучше перепроверять саммари самостоятельно.

Целевая аудитория

Обязательно следующий абзац должен быть про целевую аудиторию публикуемого видео. Учитывайте их конкретную роль (разработчики приложений, администраторы кластеров, инженеры по безопасности и т.д.), а если вы не можете сориентироваться самостоятельно, спросите у автора доклада. В случае, если автор недоступен, можно снова спросить совета у ИИ. Тот, скорее всего, не ошибется.

Прикрепление ссылок

Ссылки на видео оформляются по стандартным md-правилам. `@[название_ресурса](ссылка)`. Например:

- `@[youtube](https://youtu.be/Dy0VtzQatag?)`
- `@[rutube](6840af8411a8be4e7da9f82cb4a25103)`
- `@[](https://runtime.strm.yandex.ru/player/episode/vplekzpruxb4htfcgqwm?autoplay=0&mute=1&color=white&rel=0)`
- `@[vk](https://vk.com/video_ext.php?oid=-34475478&id=456239479&hd=2&autoplay=1)`

Для получения ссылки с Youtube достаточно нажать **копировать URL видео**, и использовать полученную ссылку.

С Rutube после получения ссылки видео в формате `https://rutube.ru/video/6840af8411a8be4e7da9f82cb4a25103/?r=plwd` достаточно оставить только часть с ID видео, как было показано в примере выше.

С Yandex достаточно скопировать ссылку на видео, а название хостинга в квадратных скобках указывать необязательно.

При работе со ссылками VK-видео необходимо учитывать, что воспроизводимый формат превью получается только при формате ссылки вида `https://vk.com/video_ext.php?oid=-84793390&id=456239888&hd=2&autoplay=0`
Тогда как при копировании ссылки на видео вк, формат будет выглядеть так:
`https://vkvideo.ru/video-84793390_456239888`.

В данном случае необходимо первую числовую комбинацию до нижнего слеша вставить в поле `oid=`, а вторую комбинацию - в поле `id=`.

> **Примечание**
>
> При наличии видео на нескольких ресурсах, оформлять ссылки с этих ресурсов необходимо с помощью табов `{% list tabs %}`

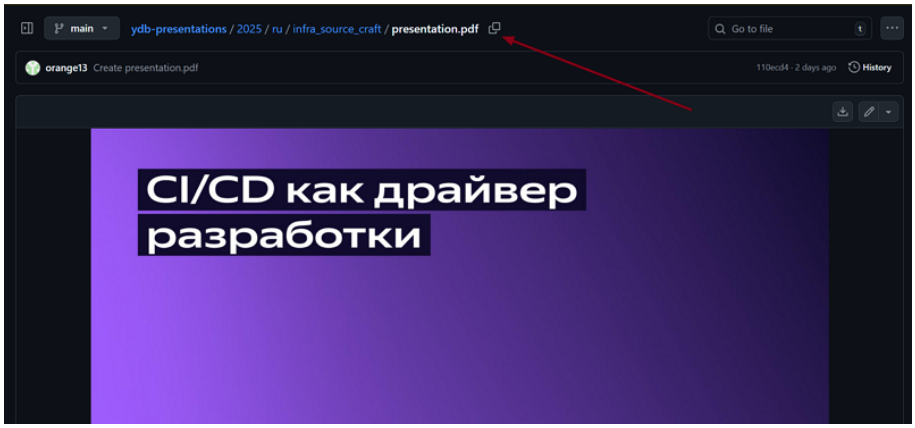
Добавление слайдов

К некоторым видеоматериалам (например, к докладам с конференций) необходимо прикладывать слайды презентаций, помещённые после ссылки на видео во вкладку **Слайды**.

Для корректного отображения презентацию необходимо загружать в виде ссылки
`https://presentations.ydb.tech/2025/ru/infra_source_craft/presentation.pdf`.

Для этого нужно:

- Загрузить презентацию в pdf - формате в репозиторий.
- Извлечь ссылку на слайды:



Получится ссылка `2025/ru/infra_source_craft/presentation.pdf`.

- Добавить в начало этой ссылки `https://presentations.ydb.tech/`.

Документирование новых команд CLI

Эта статья описывает процесс создания документации для новых команд YDB CLI. Она содержит шаблон и рекомендации, основанные на существующих статьях о командах CLI.

Общие принципы

Перед началом работы над документацией новой команды:

1. Ознакомьтесь с Руководство по стилю документации YDB для понимания общих правил стиля документации.
2. Убедитесь, что команда реализована и доступна в CLI с опцией `--help`.

Структура статьи

Каждая статья о команде CLI должна следовать следующей структуре:

1. Заголовок первого уровня

Заголовок должен кратко описывать действие команды. Используйте формулировку, которая начинается с существительного или отглагольного существительного.

Примеры:

- `# Создание топика`
- `# Удаление таблицы`
- `# Получение статуса фоновой операции`
- `# Чтение из топика`

2. Краткое описание

Первые 1–3 предложения описывают назначение команды. Если команда связана с другими командами, добавьте ссылки на них.

Пример:

```
С помощью подкоманды `topic create` вы можете создать новый топик.
```

Пример со ссылкой:

```
С помощью команды `topic consumer add` вы можете добавить читателя для [созданного ранее](topic-create.md) то
пика.
```

3. Примечания (опционально)

Если команда имеет важные особенности, ограничения или предупреждения, добавьте их сразу после описания с помощью блока `{% note %}`.

Пример:

```
{% note info %}

При удалении топика также будут удалены все добавленные для него читатели.

{% endnote %}
```

4. Общий вид команды

Покажите синтаксис команды с использованием шаблона `ydb` и правильного форматирования.

Базовый шаблон:

```
Общий вид команды:

ydb [global options...] <command> [options...] [arguments...]

- `global options` — [глобальные параметры](commands/global-options.md).
- `options` — [параметры подкоманды](#options).
- `arguments` — аргументы команды (опишите каждый).
```

Примеры:

Для команды без опций:

```
ydb [global options...] topic drop <topic-path>
```

Для команды с опциями:

```
ydb [global options...] topic create [options...] <topic-path>
```

Для сложной команды:

```
ydb [connection options] topic read <topic-path> [--consumer STR] \
  [--format STR] [--wait] [--limit INT] \
  [--transform STR] [--file STR] [--commit BOOL] \
  [дополнительные параметры...]
```

5. Ссылка на справку

Всегда добавляйте команду для получения справки:

```
Посмотрите описание команды:

ydb <command> --help
```

6. Параметры подкоманды

Если команда имеет параметры, создайте раздел `## Параметры подкоманды {#options}`.

Оформите параметры в виде таблицы:

```
## Параметры подкоманды {#options}

Имя | Описание
|---|---
`--parameter-name VAL` | Описание параметра.
```

Рекомендации по описанию параметров:

- Укажите тип значения, если он важен (например, `VAL`, `STR`, `INT`, `BOOL`).
- Укажите значение по умолчанию, если оно есть.
- Перечислите возможные значения для перечислений.
- Добавьте ссылки на концепции, если параметр связан с ними.
- Используйте HTML-теги для форматирования внутри ячеек таблицы (например, `<br/>`, `<ul>`, `<li>`).

Пример простого параметра:

```
`--consumer VAL` | Имя читателя, которого нужно добавить.
```

Пример параметра с описанием:

```
`--retention-period` | Время хранения данных в топике. Положительное число с указанием единицы измерения.<br/>Поддерживаются следующие единицы:<ul><li>`s` – секунды;</li><li>`m` – минуты;</li><li>`h` – часы;</li><li>`d` – дни.</li></ul>Значение по умолчанию — `18h`.
```

Пример параметра со ссылкой:

```
`--partitions-count`| Количество [партиций](../../concepts/datamodel/topic.md#partitioning) топика.<br/>Значение по умолчанию — `1`.
```

Группировка параметров:

Для команд с большим количеством параметров можно разделить их на группы:

```
## Параметры {#options}

### Обязательные параметры

`<topic-path>`: Путь топика

### Основные опциональные параметры

`-c VAL`, `--consumer VAL`: Имя читателя топика.

### Другие опциональные параметры

| Имя | Описание |
|-----|----------|
| `--idle-timeout VAL` | Таймаут принятия решения о том что топик пуст. |
```

7. Примеры

Всегда добавляйте раздел с примерами использования команды.

Шаблон раздела примеров:

```
## Примеры {#examples}

{% include [ydb-cli-profile](../../_includes/ydb-cli-profile.md) %}

[Описание примера]:
```

Рекомендации по примерам:

- Начинайте каждый пример с краткого описания того, что он демонстрирует.
- Используйте профиль `quickstart` для примеров: `ydb -p quickstart ...`
- Показывайте результаты выполнения команды, если они важны для понимания.
- Группируйте связанные примеры.
- Добавляйте ссылки на связанные команды или статьи в конце раздела, если это уместно.

Пример простого использования:

```
Создайте читателя с именем `my-consumer` для [созданного ранее](topic-create.md) топика `my-topic`:

ydb -p quickstart topic consumer add \
  --consumer my-consumer \
  my-topic
```

Пример с результатом:

```
Убедитесь, что читатель создан:

ydb -p quickstart scheme describe my-topic

Результат:

    RetentionPeriod: 2h
    PartitionsCount: 2
    SupportedCodecs: RAW, GZIP

    Consumers:
    ┌──────────────┬─────────────────┬──────────────────────────┬─────────────┐
    || ConsumerName | SupportedCodecs | ReadFrom                 | Important |
    ├──────────────┼─────────────────┼──────────────────────────┼─────────────┤
    || my-consumer  | RAW, GZIP       | Mon, 15 Aug 2022 16:00:00 MSK | 0         |
    └──────────────┴─────────────────┴──────────────────────────┴─────────────┘
```

Пример с несколькими вариантами использования:

```
* Чтение одного сообщения с выводом в терминал:

ydb -p quickstart topic read topic1 -c c1

* Ожидание появления и чтение одного сообщения с записью его в файл:

ydb -p quickstart topic read topic1 -c c1 -w -f message.bin
```

Полный шаблон статьи

Полный шаблон статьи доступен в отдельном файле: шаблон. Используйте его как основу при создании документации для новой команды CLI.

Специальные случаи

Команды без параметров

Если команда не имеет параметров (кроме глобальных), раздел "Параметры подкоманды" можно опустить.

Пример:

```
# Удаление топика

С помощью подкоманды `topic drop` вы можете удалить [созданный ранее](topic-create.md) топик.

Общий вид команды:

ydb [global options...] topic drop <topic-path>

* `global options` — [глобальные параметры](commands/global-options.md).
* `topic-path` — путь топика.

Посмотрите описание команды удаления топика:

ydb topic drop --help

## Примеры {#examples}
...
```

Команды с режимами работы

Если команда поддерживает несколько режимов работы, опишите их перед разделом параметров.

Пример:

```
Поддерживается три режима работы команды:

1. **Одно сообщение**. Из топика считывается не более одного сообщения.
2. **Пакетный режим**. Сообщения из топика считываются до того момента, пока в топике не закончатся сообщения для обработки.
3. **Потоковый режим**. Сообщения из топика считываются по мере их появления.
```

Предупреждения об устаревших командах

Если команда устарела, добавьте предупреждение в начале статьи:

```
{% include notitle [warning](../../reference/ydb-cli/_includes/deprecated_command_warning.md) %}
```

Именование файлов

- Используйте дефисы вместо подчёркиваний: `topic-create.md`, а не `topic_create.md`.
- Имя файла должно соответствовать структуре команды: `topic-consumer-add.md` для команды `topic consumer add`.
- Для простых команд используйте короткие имена: `version.md`, `table-drop.md`.

Интеграция в документацию

После создания статьи:

1. **Добавьте ссылку в** `_includes/commands.md` — общий список всех команд CLI.
2. **Обновите оглавление** — добавьте статью в соответствующий `toc_i.yaml` или `toc_p.yaml`.
3. **Добавьте перекрёстные ссылки** — обновите связанные статьи, добавив ссылки на новую команду.
4. **Проверьте ссылки** — убедитесь, что все внутренние ссылки корректны и используют относительные пути с суффиксом `.md`.

Проверка перед публикацией

Перед отправкой pull request убедитесь, что:

- [ ] Статья следует структуре, описанной выше.
- [ ] Все примеры протестированы и работают корректно.
- [ ] Используются правильные переменные: `ydb`, `YDB`.
- [ ] Все ссылки корректны и используют относительные пути с `.md`.
- [ ] Статья добавлена в оглавление и список команд.
- [ ] Текст соответствует руководству по стилю.
- [ ] Нет опечаток и грамматических ошибок.

См. также

- Руководство по стилю документации YDB — общее руководство по стилю документации
- Структура документации YDB — структура документации
- Процесс ревью документации YDB — процесс ревью документации
- Примеры существующих статей — изучите похожие команды для вдохновения

Документирование функциональности под фича-флагами

Документирование экспериментальной функциональности под фича-флагами включает условное предупреждение на каждой странице, где она описана. Предупреждение объясняет риски и отображается, пока соответствующий флаг выключен по умолчанию в версии YDB, для которой собирается документация.

Не каждый выключенный флаг означает экспериментальную функциональность. Если флаг управляет стабильной возможностью, которая нужна не всем, опишите её как обычную настройку. Экспериментальный статус и связь возможности с конкретным флагом согласуйте с владельцем функциональности.

Как добавить предупреждение

Добавьте плашку в начало страницы или раздела об экспериментальной функциональности. Если возможность описана на нескольких страницах, плашка нужна на каждой из них. Изменение документации экспериментальной функциональности без такого предупреждения считается неполным.

Для флага из `NKikimrConfig.TFeatureFlags` используйте следующий сниппет:

```
{% if feature_flags.enable_xxx == false %}
{% note warning "Экспериментальная функциональность" %}

Функциональность находится в разработке и по умолчанию выключена флагом [`enable_xxx`](../../reference/configuration/feature_flags.md). Её поведение может измениться без сохранения обратной совместимости, а сама функциональность может так и не стать общедоступной. Включение флага может привести к ошибкам, нестабильной работе кластера или невозможности сменить версию YDB. Не включайте флаг в production-кластерах. [Подробнее](../../reference/configuration/feature_flags.md#experimental).

{% endnote %}
{% endif %}
```

Замените `enable_xxx` на имя реального флага в snake_case — в условии и в тексте предупреждения. Скорректируйте относительные пути ссылок: в примере они указаны относительно этой статьи. Для английской версии страницы используйте такой же флаг и соответствующий текст:

```
{% if feature_flags.enable_xxx == false %}
{% note warning "Experimental feature" %}

This feature is under development and is disabled by default by the [`enable_xxx`](../../reference/configuration/feature_flags.md) flag. Its behavior may change without backward compatibility, and the feature may never become generally available. Enabling the flag may cause errors, cluster instability, or make it impossible to change the YDB version. Do not enable this flag in production clusters. [Learn more](../../reference/configuration/feature_flags.md#experimental).

{% endnote %}
{% endif %}
```

Если доступность зависит от нескольких флагов, условие должно соответствовать проверкам в реализации. Не выбирайте флаг только по сходству его имени с названием возможности.

Как выбрать переменную

Флаги определяются не только в `ydb/core/protos/feature_flags.proto`. Для других источников переменная включает имя типа, чтобы не смешивать одноимённые поля разных сообщений.

Источник	Пример переменной
`NKikimrConfig.TFeatureFlags`	`feature_flags.enable_xxx`
`NKikimrConfig.TTableServiceConfig`	`proto_flags.NKikimrConfig.TTableServiceConfig.enable_stream_write`
Вложенные protobuf-сообщения	`proto_flags.<package>.<message>.<nested_message>.<snake_case_field>`
Зарегистрированные C++-структуры	`compile_time_flags.NActors.NFeatures.TCommonFeatureFlags.probe_spin_cycles`

Имена пакетов и типов сохраняют исходный регистр; имя поля преобразуется в snake_case. Например, `EnableStreamWrite` становится `enable_stream_write`. C++-пространства имён разделяются точками в переменной документации.

Для флага из конфигурации сервиса замените условие сниппета на условие с нужным protobuf-типом. Например:

```
{% if proto_flags.NKikimrConfig.TTableServiceConfig.enable_stream_write == false %}
<!-- Здесь разместите предупреждение для соответствующей экспериментальной возможности. -->
{% endif %}
```

Генератор учитывает зарегистрированные protobuf-источники и их импорты: конфигурации сервисов, PQ, Blob Storage, аутентификации, FQ, NBS и другие сообщения. Реестр находится в `ydb/docs/tools/feature_flags/sources.json`. Если источник не зарегистрирован и не попадает в импорты, добавьте его в реестр. Для нового C++-источника также проверьте фильтры запуска сборки в GitHub workflows.

Наличие поля в сгенерированном каталоге не означает, что оно управляет экспериментальной функциональностью кластера:

- `Ydb.FeatureFlag.Status` — состояние параметра публичного API. Значение `STATUS_UNSPECIFIED` не равнозначно `false`.
- Поля Viewer описывают ответ API, а поля физического плана — данные плана запроса. Их protobuf-default не определяет доступность возможности на кластере.
- Для C++-флагов читаются только литералы `constexpr bool`. Вычисляемое выражение получает значение `null`: генератор не исполняет C++ и не определяет параметры конкретной сборки бинарного файла.

Для этих источников сначала установите смысл значения и связь с пользовательской возможностью. Не переносите на них булевое условие автоматически.

Как работает сборка

Перед сборкой Diplodoc генератор читает исходный код из того же checkout, что и документация, и добавляет значения в `default`-preset файла `ydb/docs/presets.yaml`. Поэтому документация `main` использует значения из `main`, а документация стабильной версии — из соответствующей стабильной ветки.

Значение в переменной	Результат условия `== false`
`false`	Предупреждение отображается.
`true`	Предупреждение скрыто.
`null`, включая зарегистрированный удалённый флаг	Предупреждение скрыто.
Неизвестное имя флага или типа	Проверка имён завершает сборку с ошибкой.

Отсутствие предупреждения само по себе не доказывает готовность функциональности: `null` также может означать неопределённое значение. Статус возможности проверяют по её реализации и документации.

Генератор получает значения по умолчанию из protobuf-схем и поддерживаемых C++-литералов. Они не заменяют проверку конфигурации работающего кластера и не учитывают значения, которые код приложения назначает дополнительно.

Не редактируйте и не коммитьте сгенерированный блок `presets.yaml`. При бекпорте документации переносите сниппет с правильным именем флага; значения будут получены из исходного кода целевой ветки.

Как проверить изменение

Из корня репозитория установите зависимости и запустите генератор:

```
python3 -m pip install -r ydb/docs/tools/feature_flags/requirements.txt
python3 ydb/docs/tools/feature_flags/generate.py --catalog feature-flag-defaults.json
```

Генератор проверит имена флагов в условиях и создаст каталог `feature-flag-defaults.json` с исходными файлами, типами и значениями. В CI этот каталог доступен как артефакт `feature-flag-defaults`.

Проверьте русский и английский предпросмотр из PR. Убедитесь, что имя флага и условие одинаковы, предупреждение соответствует экспериментальному статусу, а ссылки ведут на статью и её раздел `#experimental`. При локальной сборке используйте инструкции в `ydb/docs/README.md`.

Что делать при удалении флага

При удалении protobuf-поля зарезервируйте его исходное имя через `reserved`. Генератор присвоит соответствующей переменной `null`, и предупреждение с условием `== false` перестанет отображаться.

Если имя нельзя сохранить в protobuf — например, удалено сообщение целиком или C++-поле — внесите ссылку в `retired_references` файла `sources.json`. Запись должна указывать пространство переменных, исходный тип и имя поля в snake_case. Пример для удалённого поля конфигурации сервиса:

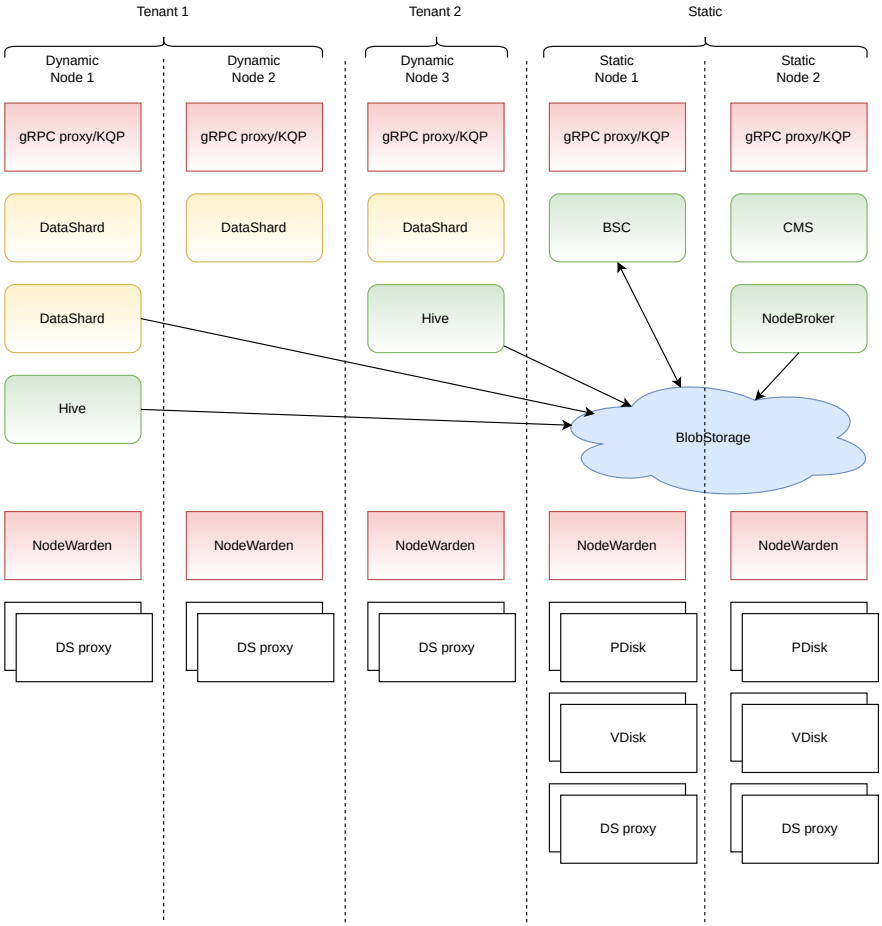
```
{
  "namespace": "proto_flags",
  "message": "NKikimrConfig.TTableServiceConfig",
  "field": "enable_removed_example"
}
```

Для `feature_flags` имя типа в записи не требуется. Для `compile_time_flags` используйте полное C++-имя структуры с разделителем `::`. Активное поле нельзя помечать удалённым. Не используйте этот реестр для обхода ошибки в имени: неизвестные ссылки должны обнаруживаться при сборке.

После удаления флага проверьте актуальность описания самой возможности. Уберите ненужный сниппет при следующей правке страницы. Если возможность по-прежнему экспериментальная, замените условие проверкой актуального флага или сохраните безусловное предупреждение.

Общая схема YDB

Примерная общая схема YDB представлена ниже.



Узлы

Одна инсталляция YDB состоит из *кластера*, который разбит на *узлы*. Узел — это один процесс в системе, как правило, ydbd. Этот узел входит в кластер и может обмениваться данными с другими узлами в этом кластере через *Interconnect*. Каждый *узел* имеет свой идентификатор, который обычно называется NodeId. NodeId — целое число от 1, состоящее из 20 бит. NodeId 0 зарезервирован для внутренних нужд и, как правило, обозначает текущий узел либо отсутствие узла.

На каждом узле выполняется ряд сервисов, реализованных через *акторы*.

Узлы могут быть статическими или динамическими.

Конфигурация статических узлов, то есть их полный перечень с указанием адресов для подключения по Interconnect, хранится в конфигурационном файле и вычитывается единожды при запуске процесса. Набор статических узлов меняется очень редко. Как правило, это происходит при расширении кластеров или при переезде узлов с одних физических машин на другие. Для того чтобы изменить набор статических узлов, необходимо разложить на **всех** узлах обновлённую конфигурацию, после чего выполнить rolling restart всего кластера.

Динамические узлы заранее неизвестны и добавляются в систему по мере запуска новых процессов. Это может быть связано, например, с созданием новых тенантов в инсталляциях YDB как базы данных. При регистрации динамического узла его процесс сначала подключается к одному из статических узлов через gRPC и через специальный сервис, который называется Node Broker, передаёт информацию о себе, получая в ответ NodeId, под которым ему можно будет войти в систему. Механизм назначения узлов во многом похож на DHCP в контексте раздачи IP-адресов.

Таблетки

На каждом узле выполняются специальные микросервисы, которые называются таблетками. Каждая таблетка имеет определённый тип и идентификатор и является singleton'ом, что означает, что в каждый момент времени во всём кластере может работать только одна таблетка с конкретным идентификатором. Таблетка может запускаться на любом из подходящих для неё узлов. Важной характеристикой таблетки является её поколение — Generation — которое увеличивается при каждом следующем запуске. Стоит отметить, что в силу распределённости системы и наличия различного рода проблем, например, сетевого партиционирования, может сложиться ситуация, когда одна и та же таблетка будет фактически выполняться на двух разных узлах одновременно. Однако distributed storage гарантирует, что только одна из них сможет успешно завершить операции, изменяющие её состояние, и при этом поколение, в котором выполнена каждая успешная операция, не будет убывать со временем.

Для системных таблеток кластера узел, на котором запускается таблетка, выбирается с помощью механизма Bootstrapper, реализующего распределённый консенсус. Пользовательскими таблетками управляет специальная таблетка Hive. Hive следит за тем, что все таблетки запущены, распределяет таблетки между узлами и каналы таблеток между группами хранения.

Узнать, на каком узле выполняется таблетка в актуальном поколении, можно через сервис *StateStorage*. Для отправки сообщений в таблетку существует специальный набор библиотек, который называется *tablet pipe*. С его помощью, зная идентификатор целевой таблетки, можно легко послать ей нужное сообщение.

Таблетку можно условно разделить на две части: базовая таблетка и пользовательская логика.

Базовая таблетка представляет собой набор таблиц, каждая из которых может состоять из одной или нескольких колонок ключа произвольного типа, а также произвольного набора колонок данных. Каждая таблица может иметь свою схему, кроме

того, таблицы можно создавать и удалять в ходе работы таблетки. Интерфейс базовой таблетки позволяет выполнять операции чтения и изменения этих таблиц.

Пользовательская логика находится между базовой таблеткой и пользователем и позволяет обрабатывать специфические запросы для данного типа таблеток, надёжно сохраняя изменения в distributed storage. Часто используемый шаблон работы таблетки — хранение всех данных в памяти, их вычитка только на старте и синхронное изменение данных в памяти и в хранилище после успешного коммита.

Как таблетка хранит данные и какие они

Базовая таблетка представляет собой LSM-дерево, в котором находятся все данные её таблицы. Уровнем ниже базовой таблетки находится distributed storage, который, грубо говоря, является KeyValue-хранилищем, в котором лежат блобы. *Блоб* — это бинарный фрагмент размером от 1 байта до 10 мегабайт, который имеет идентификатор фиксированной структуры (обычно он называется *BlobId* и имеет тип TLogoBlobID) и связанные с ним данные. Хранилище иммутабельно, то есть каждому идентификатору соответствует только одно значение, которое не может изменяться со временем. Блоб можно записать, прочитать и затем удалить, когда он станет не нужен.

Подробнее о блобах и распределённом хранилище можно прочитать в разделе Распределённое хранилище YDB.

Для distributed storage блобы являются непрозрачной сущностью. Таблетка может хранить несколько типов блобов. Наиболее часто записываемый блоб — блоб лога (имеется в виду recovery log, журнал восстановления). Лог таблетки устроен в виде списка блобов, в каждом из которых содержится информация о вносимом изменении в таблицы. При запуске таблетка находит последний блоб лога и затем рекурсивно по ссылкам вычитывает все связанные с ним блобы. В логе могут также содержаться ссылки на блобы снапшотов, которые содержат данные нескольких блобов лога после слияния (операция merge в LSM-дереве).

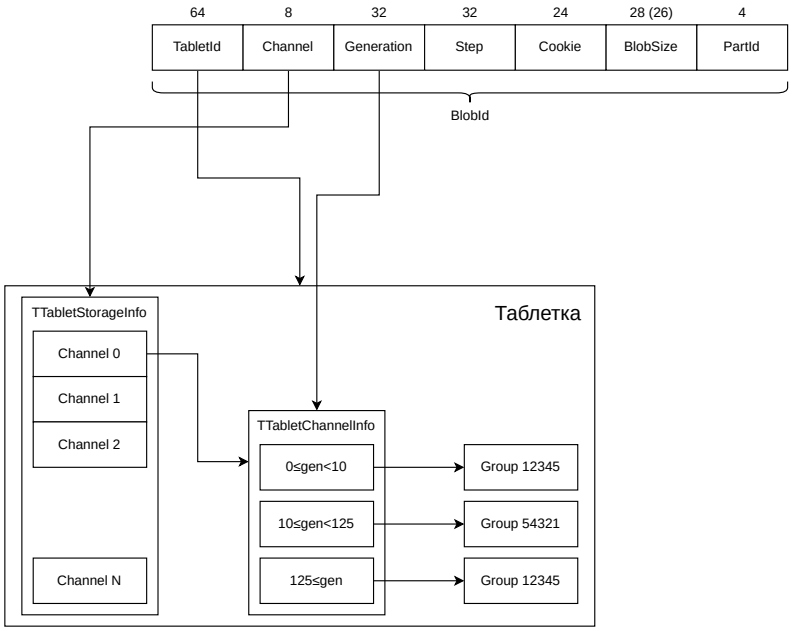
Блобы разных типов таблетка записывает в разные *каналы*. Канал указывает ветвь хранилища, в которой следует хранить блобы, и выполняет несколько функций, а именно:

1. Выбор типа хранилища (разные каналы могут быть привязаны к разным типам устройств — SSD, HDD, NVMe).
2. Балансировка нагрузки, так как каждый канал имеет лимит по IOPS, доступному месту и пропускной способности.
3. Указание типа данных. При восстановлении лога читаются только блобы из нулевого канала, что позволяет отделить их от прочих блобов.

История каналов в таблетке

Как уже говорилось, каждая группа имеет фиксированный объём данных, которые в неё могут помещаться, а также делит полосу пропускания и число операций в секунду между всеми потребителями. Нагрузка на таблетки может меняться, и в результате может сложиться так, что группа станет перегруженной и возникнет необходимость использовать для записи другую группу. Для этого вводится понятие истории, которое позволяет для каждой таблетки, зная Channel и Generation блоба, определить, в какую группу записан данный блоб.

Иллюстрация работы этого механизма приведена ниже:



Для каждого канала в структуре TTabletStorageInfo содержится подструктура TTabletChannelInfo, которая содержит диапазоны поколений и номер группы, соответствующий каждому диапазону. Диапазоны строго примыкают друг к другу, последний диапазон открыт. Номера групп могут пересекаться в разных диапазонах и даже между разными каналами — это не запрещено и достаточно часто встречается.

При выполнении записи блоба таблетка выбирает самый последний диапазон для соответствующего канала, так как запись всегда идёт от имени текущего поколения таблетки. При выполнении чтения номер группы извлекается исходя из BlobId.Generation читаемого блоба.

Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

Руководство по акторной системе

Акторная система NActors выполняет C++-объекты, доставляя сообщения в их почтовые ящики (mailbox). Это руководство описывает контракты, на которых строится реализация акторов во всех компонентах YDB, включая таблетки. Персистентное хранение данных таблеток, транзакции и восстановление добавляют к этим акторным контрактам правила конкретного компонента.

Выберите страницы, необходимые для вашего изменения:

Задача	Справочный материал
Создать и зарегистрировать актор, организовать обработку событий, выбрать пул	Жизненный цикл актора
Отправить, перенаправить, запланировать сообщение или передать его на другой узел	События и обмен сообщениями
Обработать недоставку, отмену и завершение актора	Сбои и завершение работы
Написать обработчик актора на корутине без собственного стека	Акторы на корутинах
Выбрать примитив ожидания, срок завершения, область отмены или группу задач	Примитивы корутин
Интегрировать обратные вызовы, собственные объекты ожидания или фоновые операции с ключами	Интеграция корутин
Настроить, измерить или протестировать выделение памяти для кадров корутин	Кеш кадров корутин
Выбрать тесты и проверить инварианты	Тестирование акторов

Основные понятия

Объект	Значение
Актор	C++-объект, унаследованный от IActor, с собственным состоянием и одной текущей функцией StateFunc.
ActorId	Адрес, назначаемый актору при регистрации.
Событие	Полезная нагрузка IEventBase, которой владеет оболочка IEventHandle.
Почтовый ящик (mailbox)	Очередь событий с несколькими отправителями, обеспечивающая последовательное выполнение прикреплённых к ней акторов.
Пул исполнителей (executor pool)	Ресурсы процессора и механизм планирования для выполнения активаций почтовых ящиков.
Активация	Запрос рабочему потоку обработать часть очереди почтового ящика.
ServiceId	Стабильное локальное имя, сопоставленное с текущим ActorId.
Сессия Interconnect	Один экземпляр соединения между двумя узлами.

Один цикл обработки события устроен так:

1. Отправитель передаёт оболочку события акторной системе.
2. Акторная система активирует почтовый ящик получателя.
3. Один рабочий поток находит актор-получатель и вызывает IActor::Receive.
4. Receive выбирает обработчик в следующем порядке:
   - если установлен FlagSystemMessage, используется внутренняя диспетчеризация среды выполнения (TEvResumeRunnable, TEvCheckActorLiveness); любой другой тип с этим флагом игнорируется и никогда не попадает в пользовательский код;
   - иначе, если ожидающий ActorWaitForEvent корутины без собственного стека соответствует cookie и типу события, он получает это событие;
   - иначе вызывается текущая StateFunc.
5. Затем деструктор TActorRunnableQueue в Receive выполняет задачи (runnables), запланированные в течение этого цикла, без дополнительного события в почтовом ящике. Только после этого почтовый ящик может обработать следующее событие. TEvResumeRunnable используется для возобновления через отдельное событие в почтовом ящике, если возобновление происходит не в рамках текущей активации.

Инварианты, которые нужно учитывать при проектировании

1. **Почтовый ящик выполняется последовательно.** Два обработчика, прикреплённые к одному почтовому ящику, никогда не выполняются одновременно. В разное время этот ящик могут обслуживать разные рабочие потоки. Акторы в разных почтовых ящиках могут выполняться параллельно.
2. **Последовательное выполнение не задаёт глобальный порядок.** Почтовый ящик обрабатывает события в порядке своей очереди, но для конкурирующих отправок от разных источников нет полезной гарантии взаимного порядка. Не выводите причинно-следственные связи из сообщений независимых отправителей. Каналы Interconnect и смена сессий создают дополнительные границы гарантий порядка.
3. **Планирование кооперативное.** Выполняющийся обработчик не вытесняется по истечении кванта исполнителя. Проверка кванта происходит между событиями. Долгий или блокирующий обработчик занимает один рабочий поток и задерживает все акторы в своём почтовом ящике; достаточное число таких обработчиков исчерпывает ресурсы пула. В обработчиках System/User нельзя вызывать блокирующий sleep, ожидать в цикле, ждать future, выполнять медленный синхронный ввод-вывод или удерживать блокировку, за которую конкурируют потоки.
   Блокирующую работу следует передавать диспетчеру ввода-вывода или актору в пуле IO (см. раздел о пулах).
4. **Собственное состояние актора обычно не требует блокировки.** Изменяйте его только во время выполнения этого актора. Общая память безопасна лишь при явно продуманных времени жизни

и синхронизации; типичный случай совместного использования без блокировок — акторы в одном почтовом ящике. Мьютекс не устраняет вред от блокировки рабочего потока исполнителя.

5. **Контекст активации ограничен текущим циклом обработки.** TlsActivationContext, IActor::Send, SelfId().Send, методы TActivationContext и большинство асинхронных примитивов с привязкой к актору допустимы только во время выполнения акторного кода. Никогда не сохраняйте TActorContext после возврата из обработчика или через co_await: он содержит ссылки на рабочий поток и почтовый ящик конкретной активации. Поток вне акторной системы должен использовать указатель TActorSystem.
6. **Отправка передаёт владение.** После Send, Schedule, Register или Forward считайте переданный указатель или оболочку использованными, в том числе при булевом результате false. В обработчике ev владеет полезной нагрузкой; указатель, возвращённый ev->Get, действителен только пока эта оболочка сохраняет владение. Скопируйте данные, которые нужны после текущего цикла, либо явно передайте владение через перемещение, Release или Forward.
7. **Доставка не означает обработку или успех.** Успешный Send означает, что сообщение принято для маршрутизации, а не обработано. Даже TEvUndelivered описывает лишь некоторые виды недоставки и не доказывает, была ли выполнена прикладная операция.
   Протоколы, которым нужен определённый результат, используют запрос и ответ, срок ожидания и, как правило, ключ идемпотентности.
8. **Завершение актора явно и однократно.** Зарегистрированным актором владеет среда выполнения. Он вызывает PassAway ровно один раз; повторный вызов приводит к аварийному завершению, и никто не удаляет актор напрямую. PassAway может отложить отсоединение актора, пока отменяются корутины верхнего уровня без собственного стека. В этот промежуток последующие события ещё могут попадать в StateFunc, поэтому перед вызовом PassAway переключитесь в состояние, которое их допускает. Запланированные события и записи в таблице сервисов вместе с актором не удаляются.
9. **Родитель при регистрации — это метаданные, а не управление дочерними акторами.** Он передаётся в Registered/AfterRegister и указывается отправителем события Bootstrap для актора с начальной загрузкой. Время жизни родителя и потомка не связывается автоматически.
10. **ActorId — временный адрес.** Он не является устойчивым идентификатором после перезапуска процесса. Его отладочное строковое представление намеренно не позволяет восстановить исходное значение без потерь.
11. **Исключения должны останавливаться на явно выбранной границе.** Синхронные исключения выходят из Receive, если их не принимает IActorExceptionHandler. Необработанное исключение корутины верхнего уровня проходит через тот же обработчик, а если не принято — приводит к аварийному завершению. Ожидаемые ошибки перехватывайте внутри протокола.
12. **Сообщения между узлами не являются персистентными и не имеют гарантии «ровно один раз».** При сбое сессии результат выполняющейся операции может остаться неопределённым. Переподключение и продолжение передачи — оптимизации транспорта, а не прикладные гарантии.
13. **ActorWaitForEvent сопоставляет тип и cookie, но не Sender.** Используйте новый cookie для каждой попытки запроса и не используйте его повторно, пока может прийти более ранний ответ. Несопоставленные ответы попадают в StateFunc.
14. **Каждая приостановка корутины — точка чередования выполнения.** До возобновления кадра другие события и задачи верхнего уровня могут изменить состояние актора, включая глобальное для актора состояние Become. Проверяйте состояние заново после каждой приостановки. Корутина верхнего уровня принимает все явные параметры по значению, а Bootstrap-корутина должна вызвать Become до первой приостановки.
15. **Отмена кооперативна и не является исключением.** PassAway, тайм-ауты и области отмены лишь запрашивают её; catch (...) её не перехватывает, а обычное завершение операции всё ещё может выиграть гонку. Ожидание без поддержки отмены может откладывать завершение актора неограниченно долго. Принудительное уничтожение удаляет кадры без возобновления и, возможно, без TLS актора, поэтому деструкторы не должны зависеть от Send или TActorContext.

Навигация по исходному коду среды выполнения

Контракты выше описывают поведение приложений. При изменении среды выполнения изучите реализацию и её тесты:

Тема	Исходный код в `ydb/library/actors/`
Receive, PassAway, задачи актора	core/actor.cpp, core/actor.h
Bootstrap и диспетчеризация	core/actor_bootstrapped.h, core/hfunc.h
Оболочки событий и передаваемые по сети данные	core/event.h, core/event_pb.h
Регистрация, сервисы, маршрутизация	core/actorsystem.cpp, core/executor_pool_base.cpp
Планирование почтовых ящиков	core/executor_thread.cpp, core/mailbox_lockfree.h
Вынесение блокирующего ввода-вывода	core/io_dispatcher.h, core/invoke.h
Межузловой транспорт	interconnect/
Корутины без собственного стека	async/

Если реализация и текст расходятся, приоритет имеют исходный код и тесты поведения. При изменении наблюдаемой гарантии среды выполнения обновляйте соответствующую страницу с контрактом.

Жизненный цикл актора

Актор — C++-объект, которым владеет среда выполнения и чьё состояние изменяется во время последовательных циклов обработки событий почтового ящика. Эта страница описывает создание, регистрацию, диспетчеризацию, сервисы и пулы исполнения. Начните с инвариантов в руководстве по акторной системе; переход к завершению описан в разделе Сбои и завершение работы.

Минимальный актор

```
#include <ydb/library/actors/core/actor_bootstrapped.h>
#include <ydb/library/actors/core/event_local.h>
#include <ydb/library/actors/core/events.h>
#include <ydb/library/actors/core/hfunc.h>

using namespace NActors;

enum EEv {
    EvRequest = EventSpaceBegin(TEvents::ES_PRIVATE),
    EvResponse,
};

struct TEvRequest : TEventLocal<TEvRequest, EvRequest> {
    TString Value;
    explicit TEvRequest(TString value)
        : Value(std::move(value))
    {}
};

struct TEvResponse : TEventLocal<TEvResponse, EvResponse> {
    TString Value;
    explicit TEvResponse(TString value)
        : Value(std::move(value))
    {}
};

class TExampleActor final : public TActorBootstrapped<TExampleActor> {
public:
    void Bootstrap() {
        Become(&TThis::StateWork);
    }

    void Handle(TEvRequest::TPtr& ev) {
        Send(ev->Sender, new TEvResponse(ev->Get()->Value), 0, ev->Cookie);
    }

    void BeginShutdown() {
        Become(&TThis::StateDying);
        PassAway();
    }

    STRICT_STFUNC(StateWork,
        hFunc(TEvRequest, Handle);
        cFunc(TEvents::TEvPoison::EventType, BeginShutdown);
    )

    STFUNC(StateDying) {
        Y_UNUSED(ev); // PassAway may be waiting for coroutine tasks.
    }
};
```

В этом примере TEvRequest и TEvResponse предназначены только для локальной передачи. Для сообщения, которое может пересекать границу узла, используйте сериализуемое событие.

Создание, Bootstrap и диспетчеризация состояний

`TActor<TDerived>` принимает начальную функцию состояния и начинает ожидать события. `TActorBootstrapped<TDerived>` устанавливает состояние начальной загрузки; при регистрации в очередь добавляется событие Bootstrap, которое вызывает одну из поддерживаемых перегрузок Bootstrap.

Конструктор также сохраняет тип активности, используемый в счётчиках мониторинга. Значение по умолчанию выбирается в таком порядке: TDerived::ActorName (статическая строка), затем TDerived::ActorActivityType() (перечисление), затем имя C++-типа. Можно передать строку или перечисление в конструктор либо вызвать SetActivityType, но только до Register: SetActivityType выбрасывает исключение, если у актора уже есть ActorId.

Bootstrap должен установить состояние для последующих событий. В синхронном Bootstrap вызовите Become до возврата. В Bootstrap на корутине вызовите Become до первой приостановки: в этот момент управление возвращается вызывающему коду.

Сигнатура функции состояния:

```
void State(TAutoPtr<IEventHandle>& ev);
```

Become меняет текущее состояние всего актора, а не только текущего запроса или корутины. Поэтому обработчики на корутинах, чьё выполнение пересекается во времени, могут наблюдать результаты вызовов Become друг друга.

Макросы диспетчеризации из hfunc.h упрощают описание веток switch:

Макрос	Сигнатура вызываемого обработчика

hFunc(TEv, Handle)	Handle(TEv::TPtr&)
HFunc(TEv, Handle)	Handle(TEv::TPtr&, const TActorContext&)
sFunc / SFunc	Handle() / Handle(ctx), без использования полезной нагрузки
cFunc / CFunc	Диспетчеризация по числовому типу события, без типизированного указателя
fFunc / FFunc	Handle(ev) / Handle(ev, ctx), с исходной оболочкой события
IgnoreFunc(TEv)	Отбросить событие этого типа

STRICT_STFUNC диагностирует неизвестный тип через Y_DEBUG_ABORT_UNLESS в отладочных сборках. Это средство отладки, а не гарантия протокола в сборке release. Обычный STFUNC позволяет задать собственное поведение по умолчанию. Для диспетчеризации используйте GetTypeRewrite, как это делают макросы: перенаправление и Interconnect могут изменять эффективный тип события и получателя.

У обработчиков на корутинах другая сигнатура

Корутина верхнего уровня без собственного стека — нестатический метод IActor, возвращающий void. Все явные аргументы должны передаваться **по значению**:

```
void HandleAsync(TEvRequest::TPtr ev) {
    // ...
    co_await Something();
}

STRICT_STFUNC(StateWork,
    hFunc(TEvRequest, HandleAsync);
)
```

hFunc передаёт lvalue TAutoPtr, чья операция, похожая на копирование, передаёт владение параметру корутины, принятому по значению. Ссылочный параметр не позволяет выбрать promise акторной корутины и приводит к ошибке компиляции.
Полный контракт описан в разделе Акторы на корутинах.

Регистрация и владение

Из цикла выполнения актора:

```
TActorId child = Register(new TChild);
TActorId other = Register(
    new TChild,
    TMailboxType::HTSwap,
    AppData()->UserPoolId);
TActorId colocated = RegisterWithSameMailbox(new THelper);
```

Из кода, располагающего указателем TActorSystem:

```
TActorId id = actorSystem->Register(
    new TExampleActor,
    TMailboxType::HTSwap,
    poolId);
```

Регистрация немедленно передаёт владение актором. После неё нельзя разыменовывать исходный указатель. Среда выполнения прикрепляет актор к почтовому ящику, назначает ActorId и вызывает Registered. Реализация Registered по умолчанию вызывает AfterRegister и отправляет возвращённую им оболочку события — так актор с начальной загрузкой получает Bootstrap. Переопределение Registered без вызова базовой реализации пропускает оба действия. Эти обратные вызовы регистрации не должны исходить из того, что выполняются в активации нового актора; работу, зависящую от активации, выполняйте в Bootstrap или при обработке следующего события.

RegisterWithSameMailbox допустим в активации актора. После этого акторы выполняются последовательно в одном почтовом ящике и могут намеренно использовать общее состояние, но не могут работать параллельно. IActor::InvokeOtherActor временно меняет идентичность актора в контексте активации для прямого вызова в рамках одного почтового ящика. Такой вызов должен быть полностью синхронным: он не должен пересекать co_await; никогда не вызывайте асинхронный метод другого актора через обычный указатель.

Аргумент типа почтового ящика сохранён в API для совместимости. Текущий путь регистрации встроенного исполнителя использует интрузивный почтовый ящик без блокировок и не выбирает между историческими значениями TMailboxType. Не подбирайте Simple/HTSwap/ReadAsFilled и другие значения, не проверив используемую реализацию исполнителя.

ActorId и сервисы

TActorId — 16-байтовое значение, содержащее локальный идентификатор, подсказку (hint) и закодированные биты узла и пула. Обычный идентификатор назначается при регистрации и действителен только для данного экземпляра актора. NodeId указывает узел-владелец; нулевой узел в идентификаторе сервиса означает локальный узел.

ToString и Out предназначены для отладки. Они опускают биты пула, поэтому разбор строки в общем случае не позволяет восстановить исходный идентификатор. Не сохраняйте такую строку как адрес или устойчивый ключ базы данных.

Идентификатор сервиса хранит имя длиной до 12 байт и сопоставляется с актором локально:

```
const TActorId serviceId = TActorId(nodeId, "myservice");
actorSystem->RegisterLocalService(serviceId, actorId);
```

```
actorSystem->RegisterLocalService(serviceId, TActorId()); // clear
```

Маршрутизация выполняет один поиск соответствия между сервисом и актором. Не рассчитывайте на цепочки псевдонимов сервисов. Имя сервиса стабильно лишь потому, что прикладной код поддерживает это соответствие; при переключении после сбоя нужно установить новое.

Замена одного непустого идентификатора актора на другой без удаления соответствия не делает сервис временно отсутствующим: параллельный поиск может вернуть старый или новый идентификатор актора, но не пустой. Это не продлевает жизнь старого актора и не гарантирует доставку событий. Поиск не захватывает блокировку писателя; если во время просмотра записей произошла замена, поиск без результата может повториться. Поэтому при непрерывных заменах поиск не имеет гарантии wait-free.

Почтовые ящики, пулы и поведение исполнителя

Гарантируется последовательное выполнение почтового ящика, а не конкретный алгоритм очереди. Текущая встроенная среда выполнения использует интрузивный почтовый ящик без блокировок. При регистрации почтовый ящик назначается логическому пулу исполнителей.

Исполнитель обрабатывает события, пока не исчерпан настроенный квант времени или числа событий, не опустел почтовый ящик, Tail не передал выполнение либо не произошло мягкое вытеснение рабочего потока общего пула. Эти ограничения проверяются между событиями, но никогда внутри обработчика. Значения по умолчанию и производственная конфигурация YDB различаются, поэтому корректность не должна зависеть от конкретной длительности или числа событий.

Логические пулы определяют планирование нагрузки и учёт ресурсов. В некоторых конфигурациях общие или объединённые рабочие потоки исполнителей могут временно предоставляться пулам или переключаться между ними. Поэтому утверждение «пулу A принадлежит отдельный набор потоков ОС» не является универсальным инвариантом. Имена User/System также не подразумевают фиксированный бюджет времени для обработчика.

В узле YDB TAppData предоставляет идентификаторы пулов нагрузки процесса (SystemPoolId, UserPoolId, IOPoolId, BatchPoolId). Передавайте их в Register; не задавайте числовые идентификаторы пулов в коде.

Блокирующую работу нужно выносить из обработчиков System/User:

- InvokeIoCallback(callback, AppData()->IOPoolId, activity) отправляет TEvInvokeQuery локальному сервису IoDispatcher (MakeIoDispatcherActorId). Выделенные потоки ОС («kikimr IO») выполняют callback. Этот обратный вызов не является циклом выполнения актора: передавайте уведомление о завершении через TActorSystem::Send. Если диспетчер не зарегистрирован, Send завершается неудачей и вспомогательная функция использует актор ExecuteLater в переданном poolId, который *занимает* рабочий поток исполнителя.
- CreateInvokeActor(callback, complete, activity) вместе с Register в IOPoolId выполняет callback в короткоживущем акторе пула IO, затем отправляет TEvInvokeResult родителю. Обрабатывайте это событие через Process() или шаблонный `GetResult<TCallback>` (). Рабочий поток пула IO всё равно занят на всё время выполнения callback.
- Регистрация долгоживущего актора в IOPoolId предназначена для акторов, чьи обработчики сами выполняют блокирующую работу. Другие почтовые ящики этого пула всё равно могут оставаться без обслуживания, пока он выполняется.

Акторы одного почтового ящика всегда выполняются последовательно, даже если в пуле много рабочих потоков. Разные почтовые ящики могут выполняться параллельно. Заблокированный рабочий поток уменьшает доступные ресурсы пула; достаточное число заблокированных потоков останавливает все почтовые ящики, которым нужны эти ресурсы.

Потоки вне акторной системы

Внешний поток может использовать указатель TActorSystem с гарантированным временем жизни для Send или Ask. Он не должен вызывать IActor::Send, SelfId().Send, методы доступа TActivationContext, объекты ожидания с привязкой к актору или напрямую работать с объектами акторов. Определите владельца указателя TActorSystem и убедитесь, что ни один отправитель не использует его после завершения системы.

Для асинхронных внешних API передавайте уведомление о завершении обратно как событие или используйте описанный потокобезопасный мост ожидания для корутин без собственного стека. Никогда не возобновляйте акторную корутину напрямую во внешнем потоке.

События и обмен сообщениями

Сообщение актора состоит из полезной нагрузки, которой оно владеет, и конверта со сведениями для маршрутизации и сопоставления запросов с ответами. Эта страница объясняет локальную и удалённую доставку, пересылку, таймеры и время жизни полезной нагрузки. Инварианты почтового ящика описаны в руководстве по акторной системе, а освобождение ресурсов — в разделе «Сбои и завершение работы».

События и конверты

IEventHandle содержит:

- исходные и переписанные тип и получателя;
- ActorId отправителя Sender и получателя Recipient;
- 64-битный Cookie;
- флаги и 12-битный номер канала Interconnect;
- внутрипроцессный IEventBase или сериализованные данные;
- трассировку и, для принятых удалённых событий, ActorId сессии InterconnectSession.

Среда выполнения не интерпретирует Cookie, кроме случаев, когда отдельный вспомогательный механизм документирует иное. Распространённое соглашение запрос/ответ — копировать ev->Cookie в ответ, но это часть протокола, а не автоматическое поведение. Аргумент cookie у Send по умолчанию равен 0. Считайте 0 источником коллизий, если несколько незавершённых ожиданий могут использовать один тип: ActorWaitForEvent и уведомления TrackDelivery сопоставляются по cookie, поэтому для каждой попытки запроса используйте новое ненулевое значение и не используйте его повторно, пока может прийти более ранний ответ или уведомление.

Номера типов событий разделены на пространства по 65 536 значений. Для протокола компонента выделяйте уникальный диапазон в соответствующем назначенном пространстве событий. TEvents::ES_PRIVATE намеренно повторно используется приватными и локальными протоколами акторов и тестами; оно не уникально глобально, поэтому значения, способные попасть в один диспетчер, всё равно должны быть однозначными.

Классы полезной нагрузки

- `TEventLocal<T, Type>` — внутрипроцессная полезная нагрузка C++, которая не сериализуется.
- `TEventPB<T, Proto, Type>` — обычное сетевое событие на основе protobuf. Бинарные блобы, которым не место в protobuf, передаются рядом с ним: AddPayload(TRope) возвращает индекс; сохраните его в proto; получатель читает GetPayload(i) / GetPayloadCount(). Не помещайте TRcBuf в поля bytes protobuf, если нужна передача без копирования. Сетевое кодирование — деталь реализации: сейчас это маркер 0x06, размеры полезной нагрузки, блобы, затем proto; 0x07 — устаревшая схема, поддерживаемая только при разборе.
- `TEventSimpleNonLocal<T, Type>` сериализуется, но не передаёт поля производного класса. Добавление полей в производный класс не включает их в сетевое представление. TEvPoison использует этот класс и может отправляться на другой узел.
- API событий Flat/rope предназначены для специализированных форматов без копирования.

Не отправляйте локальное событие на другой узел по ошибке. Максимальный размер сериализованного события Interconnect по умолчанию — 140 МиБ с учётом накладных расходов сериализации; конфигурация может изменить или уменьшить этот предел. Слишком большое сетевое событие завершает всю сессию, а не просто отбрасывается само. Для больших данных предпочитайте протоколы с разбиением на части или блобовые протоколы.

Флаги отправки

Чаще всего нужны следующие младшие биты:

Флаг	Контракт
FlagTrackDelivery	Запрашивает у маршрутизации уведомления о поддерживаемых случаях недоставки. Это не подтверждение доставки.
FlagForwardOnNondelivery	При локальной недоставке направляет событие получателю пересылки из конверта; для специальных сценариев.
FlagSubscribeOnSession	При удалённой отправке подписывает Sender на текущий экземпляр сессии.
FlagGenerateUnsureUndelivered	Вместе с FlagTrackDelivery также сообщает о событиях Interconnect с неопределённым результатом доставки, устанавливая Unsure; сам по себе не действует.
FlagDebugTrackReceive	Средство для отладки.
FlagDisablePayloadChecksums	Специальная настройка компромисса для IC/XDC/RDMA; не используйте без необходимости.

Верхние 12 бит флагов кодируют канал Interconnect через IEventHandle::MakeFlags. FlagExtendedFormat — метаданные сериализации среды выполнения, а не прикладной флаг. FlagSystemMessage также не является прикладным флагом: Receive поглощает любое событие с этим битом до StateFunc и до ActorWaitForEvent. Места отправки в библиотеке передают TEvResumeRunnable::EventFlags или TEvCheckActorLiveness::RequestFlags; конструктор конверта не выводит этот бит из типа события C++. Если установить его на собственном событии, полезная нагрузка будет проигнорирована. В текущем дереве исходного кода FlagFailFastWhenDisconnected и FlagUseSubChannel не предоставляют пригодного прикладного контракта; не стройте на них протокол.

FlagForwardOnNondelivery имеет приоритет над TrackDelivery: он передаёт исходную полезную нагрузку настроенному резервному получателю вместо создания TEvUndelivered. Сформированное TEvUndelivered содержит исходные тип и cookie, а Sender его конверта — получатель, которому не удалось доставить событие. Само это уведомление также может оказаться недоставленным.

Отправка, пересылка и планирование

Внутри активации:

```
Send(recipient, new TEvRequest("x"), flags, cookie);
SelfId().Send(recipient, new TEvRequest("x"), flags, cookie);
TActivationContext::Send(
    new IEventHandle(recipient, SelfId(), new TEvRequest("x"),
                     flags, cookie));
```

Вне выполнения актора:

```
actorSystem->Send(
    new IEventHandle(recipient, sender, new TEvRequest("x"),
                     flags, cookie));
```

Не используйте API Send, зависящие от TLS, из произвольных потоков. Если вызывающей стороне вне актора нужен результат, ActorSystem::Ask создаёт временный ожидающий актор и возвращает future. Используйте конечный тайм-аут: по умолчанию он может быть бесконечным. Ask принимает первое полученное событие, каким бы оно ни было: совпадающий тип или отсутствие ограничения на тип приводит к успешному завершению future, любой другой тип — к исключению. Ask не проверяет подлинность Sender и не продолжает ждать после постороннего события, поэтому там, где это существенно, используйте выделенный адрес или дополнительное сопоставление на уровне протокола.

Сокращённая перегрузка ActorSystem::Send(recipient, event) использует синтетический сервисный идентификатор акторной системы в качестве Sender. Этот адрес не предназначен для ответов.

Forward переносит полезную нагрузку в новый конверт, адресованный новому получателю. Он сохраняет основные сведения об отправителе, флагах и cookie, но не все поля происхождения, сессии, переписывания и недоставки. Если протоколу нужен явный контракт конверта, создание нового события понятнее.

Отправка Common, Lazy и Tail

Common обычным образом активирует неработающий почтовый ящик получателя. Lazy и Tail — локальные оптимизации планирования:

- Lazy может позволить текущему рабочему потоку захватить неработающий почтовый ящик получателя и отложить его активацию до завершения обработки текущего ящика. Это может улучшить локальность, но добавить задержку за долгой текущей активацией;
- Tail завершает обработку текущего почтового ящика после этого события и передаёт оставшийся квант захваченному получателю, если исполнитель может это сделать.

Если оптимизация недоступна, например из стороннего потока или при неподходящем соотношении пулов, используется обычная отправка. Эти режимы не меняют семантику доставки, владения или порядка; применяйте их только после измерений.

Таймеры

```
Schedule(TDuration::Seconds(1), new TEvWakeup);
Schedule(deadlineMonotonic, new TEvWakeup);
```

Удобная перегрузка IActor создаёт конверт, адресованный самому актору, с пустым Sender и нулевым Cookie события. Если нужен логический идентификатор таймера, передавайте его в полезной нагрузке или явно создайте IEventHandle.

ISchedulerCookie координирует отмену в планировщике и не является IEventHandle::Cookie. Сразу при планировании используйте двусторонний держатель cookie. Его отсоединение может предотвратить доставку только в том случае, если выиграет гонку с планировщиком; после передачи события в почтовый ящик оно всё ещё может прийти. Таймеры не удаляются автоматически при смерти актора, а их полезная нагрузка не продлевает его жизнь. Делайте обработчики таймеров идемпотентными и проверяйте поколение или монотонный срок. Для логики прошедшего времени предпочитайте TMonotonic/TDuration: системные часы могут скачкообразно изменяться.

Проверка существования актора

SendActorLivenessCheck — локальная проверка маршрутизации. Живой локальный ActorId даёт TEvActorAlive, а неизвестный локальный ActorId — TEvActorDead. Удалённая цель даёт TEvActorLivenessUnsure; это не распределённая проверка работоспособности, и она ничего не говорит о продвижении прикладной работы.

Локальная доставка

Для обычного локального Send:

- если ActorId зарегистрирован в почтовом ящике, конверт ставится в очередь;

- если он неизвестен или уже отсоединён, событие уничтожается;
- с FlagTrackDelivery поддерживаемая локальная недоставка создаёт TEvUndelivered с ReasonActorUnknown.

После принятия события в Receive задача TrackDelivery выполнена. Он не сообщает о том, что обработчик выбросил исключение, проигнорировал событие, завершил работу до ответа или не выполнил прикладную операцию.

Poison — обычное событие. События, уже стоящие перед ним, выполнятся первыми. События за ним не исчезают автоматически: после немедленного отсоединения актора они отбрасываются как адресованные неизвестному получателю, но если PassAway ждёт задачи корутин, они могут дойти до StateFunc завершающегося актора. Конкурирующий Send после постановки Poison в очередь всё ещё успешен; позднее событие просто оказывается за Poison в порядке FIFO.

Interconnect

Отправка на ActorId с другим NodeId маршрутизируется через прокси Interconnect этого узла. Через эту границу могут проходить только сериализуемые события. Трафик разделён на каналы; общего прикладного порядка между каналами нет.

Контракт сбоев, на который безопасно опираться:

- экземпляр сессии может установить соединение и позднее завершиться;
- часть сообщений могла быть доставлена, тогда как результат других незавершённых передач остаётся неопределённым;
- классические сессии могут продолжаться после некоторых переподключений TCP, тогда как IC v2 намеренно не поддерживает такое продолжение;
- заменяющая сессия задаёт новую границу порядка и подписок.

Поэтому прикладным протоколам всё равно нужны тайм-ауты ответов, устойчивость к дубликатам или идемпотентность и восстановление после смены сессии.

TrackDelivery через Interconnect

При сбое классической сессии:

- отслеживаемое событие, ещё находящееся в неотправленной Queue канала, получает определённое уведомление TEvUndelivered;
- сериализованное, но ещё не подтверждённое событие получает TEvUndelivered только при установленном вместе с FlagTrackDelivery флаге FlagGenerateUnsureUndelivered; уведомление содержит Unsure=true, поскольку получатель мог уже принять событие;
- без флага unsure события с неопределённым результатом могут не породить уведомления;
- IC v2 даёт ещё меньше гарантий: отслеживаемые события, уже переданные его движку, при завершении могут быть отброшены вообще без уведомления.

Эти уведомления о сбое сессии создаются локально на узле отправителя. Только обратное уведомление ReasonActorUnknown от удалённого узла пересекает сеть и само может потеряться; уведомления Unsure локальны и не сериализуются. Это полезная диагностика и быстрое оповещение об ошибках, а не протокол «ровно один раз».

Подписка на сессию

FlagSubscribeOnSession в удалённом событии подписывает его Sender на этот экземпляр сессии. TEvNodeConnected указывает установленную сессию; TEvNodeDisconnected означает, что **эта сессия завершилась**, а не обязательно что узел недоступен. Исправная неиспользуемая сессия может закрыться. Более позднее соединение — другой экземпляр.

Явно отправляйте TEvUnsubscribe, когда связь больше не нужна. Текущая среда выполнения также периодически проверяет существование умерших локальных подписчиков, но ожидание такой медленной проверки излишне удерживает состояние сессии. Сессия хранит одну подписку на ActorId отправителя; более поздние сведения о подписке этого отправителя могут заменить предыдущую запись.

Если на запрос нужно ответить именно через ту сессию, которая его доставила, используйте ev->InterconnectSession:

```
auto reply = MakeHolder<IEventHandle>(
    ev->Sender, SelfId(), new TEvResponse("ok"), 0, ev->Cookie);
if (ev->InterconnectSession) {
    reply->Rewrite(TEvInterconnect::EvForward, ev->InterconnectSession);
}
TActivationContext::Send(reply.Release());
```

Это не позволяет ответу незаметно перейти на заменяющую сессию. Но такой приём не делает доставку устойчивой к потере сессии.

Время жизни данных и большие буферы

TRope и API событий с разделяемыми буферами позволяют дёшево создавать срезы и передавать владение. Используйте их, если тип события это поддерживает; не собирайте данные в непрерывный буфер только ради отправки. При этом сохранение конверта события или rope на время долгого ожидания удерживает все лежащие в его основе блоки памяти. Извлекайте небольшие поля и освобождайте большие полезные нагрузки как можно раньше.

Никогда не сохраняйте сырой указатель, возвращённый ev->Get, после уничтожения конверта, пересылки, передачи владения или ленивого преобразования из сериализованного представления.

Сбои и завершение работы

Завершение актора — явный переход, который может продолжаться после возврата из инициировавшего его обработчика. Эта страница описывает отмену, освобождение ресурсов и связь времени жизни актора с кадрами корутин. О сбоях доставки читайте в разделе События и обмен сообщениями, о владении кадрами — в разделе Акторы на корутинах.

Завершение и освобождение ресурсов

TEvPoison — обычное пользовательское событие (TEventSimpleNonLocal), в том числе при передаче между узлами. Среда выполнения не завершает актор за вас: получатель должен обработать событие и вызвать PassAway. Уже поставленный в очередь Poison не отбрасывает последующие Send; см. раздел о локальной доставке.

PassAway работает кооперативно:

1. При повторном вызове аварийно завершает процесс, иначе помечает актор флагом PassedAway.
2. Запрашивает отмену каждой зарегистрированной корутины верхнего уровня без собственного стека.
3. Если хотя бы одна такая задача остаётся, возвращает управление, не снимая актор с регистрации.
4. Последняя задача при снятии с регистрации вызывает FinishPassAway.
5. FinishPassAway отсоединяет ActorId, а исполнитель удаляет C++-объект актора после завершения текущего Receive.

Вложенные кадры `async<T>` принадлежат задаче верхнего уровня, которая их ожидает; они не регистрируются как отдельные задачи актора.

Важно учитывать промежуток до отсоединения. Обычные обработчики StateFunc всё ещё могут вызываться. Новая корутина верхнего уровня может выполнить свой синхронный начальный участок, но с самого начала считается отменённой и должна завершиться при первой приостановке с поддержкой отмены. Если последующий TEvPoison всё ещё попадает в StateFunc и снова вызывает PassAway, процесс аварийно завершается. Если у актора могут оставаться активные задачи, перед вызовом PassAway переключитесь в состояние, которое отбрасывает или явно обрабатывает поздние события, как в минимальном примере. Корутина, инициирующая завершение, должна выполнить:

```
Become(&TThis::StateDying);
PassAway();
co_return;
```

Задача, приостановленная на объекте ожидания без поддержки отмены, например на TFuture, может сохранять актор зарегистрированным неограниченно долго. Уничтожение акторной системы или почтового ящика — аварийный путь: DestroyActorTasks принудительно уничтожает оставшиеся кадры корутин, а затем актор. Пользовательский код после await не выполняется, но живые параметры и локальные переменные кадра уничтожаются по правилам C++, поэтому деструкторы RAII вызываются.

PassAway лишь запрашивает отмену; он не прерывает вызывающий обработчик, поэтому после него не выполняйте работу, чувствительную к завершению актора. Обычное завершение операции, уже находящееся в обработке, может выиграть гонку, поэтому поздняя работа после await должна проверять флаг завершения или поколение и оставаться безопасной. TActorSystem::Stop выполняет жёсткое завершение: он не рассылает каждому актору Poison для корректного завершения. Дочерние акторы и запланированные события также не уничтожаются автоматически при вызове PassAway у родителя.

PassAwayGuard — RAII-объект, поддерживающий только перемещение, чей деструктор вызывает PassAway. Он полезен лишь тогда, когда единолично отвечает за переход актора к завершению; сочетание с другим путём завершения или обработкой Poison создаёт риск запрещённого повторного вызова.

Псевдонимы почтового ящика, созданные через IActor::RegisterAlias, удаляются при отсоединении актора. Записи в таблице сервисов TActorSystem существуют отдельно и вместе с актором не удаляются; очищайте или заменяйте их явно.

Протокол отмены

Отмена — отдельный протокол завершения корутин, реализованный через методы await_cancel и await_cancelled. Это не механизм throw/catch.

При отмене корневой задачи:

1. Запрос фиксируется в её promise.
2. У текущего объекта ожидания, на котором задача приостановлена, запрашивается отмена, если он поддерживает это расширение.
3. Объект ожидания может подтвердить отмену сразу, выполнить асинхронное освобождение ресурсов и подтвердить позже либо возобновить задачу обычным образом, если завершение операции выиграло гонку.
4. После подтверждения отмены кадр уничтожается и вызываются деструкторы живых RAII-объектов.
5. Promise снимает корневую задачу с регистрации в акторе.

Важные следствия:

- код после отменённого co_await обычно не выполняется;
- catch (...) не является обработчиком отмены;
- операция await_ready может завершиться, не заметив отмену;
- уже отменённая задача может выполнить синхронный начальный участок до первой приостановки с поддержкой отмены;
- объект ожидания без поддержки отмены всё ещё может приостановить задачу и позже возобновить её обычным образом;
- завершение, запланированное до отмены, часто выигрывает гонку.

RAII: что и когда выполняется

Путь выхода	Продолжается ли выполнение тела?	Вызываются ли деструкторы живых объектов кадра?	Можно ли полагаться на акторный TLS?

Обычный co_return	Да, с обычным выходом из областей видимости	Да	Да, во время цикла выполнения актора
Исключение во вложенной корутине	Стек раскручивается до catch или родителя	Да	Да, во время цикла выполнения актора
Подтверждённая отмена	Код после отменённого await не выполняется	Да	Как правило, во время выполнения актора
Уничтожение `async<T>`, который никто не ожидал	Тело не начинает выполняться	Только promise и параметры; локальных переменных тела ещё нет	Акторный TLS не требуется
Принудительное уничтожение почтового ящика	Возобновления и кода очистки в теле не будет	Да, через handle.destroy	Нет

Для синхронного освобождения принадлежащих объектов используйте обычный RAII или Y_DEFER:

- удаляйте состояние запросов, принадлежащее актору;
- отменяйте регистрации;
- освобождайте буферы, блокировки, которые не удерживаются через await, и интрузивные ссылки;
- отмечайте завершение и обновляйте состояние владельца там, где гарантирован акторный контекст.

Деструкторы должны оставаться безопасными, даже если сохранилось лишь владение памятью.
При жёстком завершении не вызывайте безусловно Send, не обращайтесь к TActorContext
или счётчикам, принадлежащим исполнителю. В существующем производственном коде такие
необязательные действия часто защищены проверкой if (TlsActivationContext), тогда как
очистка собственных данных и памяти выполняется всегда.

RAII не может выполнять co_await. Если для отмены требуется асинхронная отписка или откат,
оформите их как ожидаемую операцию либо используйте InterceptCancellation.
Не прячьте асинхронное освобождение ресурсов в деструкторе.

Поведение деструкторов двух примитивов особенно важно:

- уничтожение активного TAsyncContinuation планирует возобновление ожидающей задачи с logic_error;
- уничтожение TAsyncEvent с ожидающими задачами планирует их возобновление со значением false.

Эти операции допустимы только при наличии подходящей акторной очереди выполнения.
Перед уничтожением во внешнем потоке или вне активации убедитесь, что объекты пусты.

Исключения

Вложенный `async<T>` сохраняет исключение и повторно выбрасывает его в await_resume родителя,
поэтому действуют обычные try/catch и RAII:

```
try {
    co_await Child();
} catch (const TExpectedError& e) {
    // protocol error handling
}
```

Исключение, вышедшее из обработчика верхнего уровня с возвращаемым типом void, передаётся
в IActorExceptionHandler. Если оно не принято, среда выполнения аварийно завершается.
Отмена никогда не попадает в этот блок catch.

Акторы на корутинах

Этот раздел описывает библиотеку NActors async для корутин без собственного стека в текущем дереве исходного кода YDB. Он сочетает руководство по использованию с кратким контрактом корректности. Устаревший API TActorCoro для корутин с собственным стеком использует другой механизм и здесь не рассматривается.

Основные заголовки находятся в `ydb/library/actors/async/`:

- async.h;
- wait_for_event.h;
- continuation.h;
- event.h;
- sleep.h и yield.h;
- timeout.h;
- cancellation.h;
- task_group.h;
- low_priority.h.

Подключайте async.h в единице трансляции, где определяется метод-корутина актора. Этот заголовок предоставляет специализацию coroutine_traits, которая превращает метод актора, возвращающий void, в задачу актора верхнего уровня.

Описание API ожидания приведено в разделе «Примитивы корутин», завершения из внешнего контекста и типовых приёмов — в разделе «Интеграция корутин», а отмены и поведения RAII — в разделе «Сбои и завершение работы». Эти правила опираются на руководство по системе акторов.

Краткий контракт

1. **Корутины не добавляют потоки.** Код актора по-прежнему выполняется последовательно через почтовый ящик. Несколько кадров могут быть приостановлены, но одновременно выполняется только один цикл обработки события или одно продолжение.
2. **У двух видов кадров разные владельцы.** Корутина актора, возвращающая void, — задача верхнего уровня, зарегистрированная в акторе. `async<T>` — ленивый дочерний кадр, которым владеет ожидающее его выражение.
3. **Параметры верхнего уровня передаются по значению.** Каждый явный параметр корутины актора, возвращающей void, должен иметь нессылочный тип. В частности, асинхронный обработчик hFunc принимает TEv::TPtr ev, а не TEv::TPtr&.
4. **Вызов задачи верхнего уровня выполняется немедленно и не создаёт структурированной связи.** Он работает синхронно до первой фактической приостановки и не возвращает дескриптор задачи. Вызов другой корутины, возвращающей void, не делает её дочерней; если важны порядок выполнения и владение, используйте `async<T>` и co_await.
5. **Вызов `async<T>` ленивый.** Он создаёт приостановленный кадр; тело начинает выполняться только при непосредственном ожидании. Если объект отбрасывается, ещё не запущенный кадр уничтожается.
6. **Каждая возможная приостановка — точка чередования выполнения.** До возобновления этого кадра другое событие или задача верхнего уровня может изменить состояние актора. Последовательная обработка почтового ящика предотвращает гонки данных, но не устаревшие ссылки и логические ошибки.
7. **Отмена — кооперативная передача управления, а не исключение.** Код после отменённой приостановки обычно пропускается; живые объекты кадра уничтожаются, поэтому RAII выполняется. catch (...) не перехватывает отмену.
8. **PassAway запрашивает отмену и может вернуть управление.** Актор остаётся зарегистрированным, пока все задачи верхнего уровня не завершат отмену. Ожидание, не поддерживающее отмену, может отложить завершение актора на неопределённое время.
9. **Примитивы, учитывающие контекст актора, локальны для актора.** Resume/Throw/уничтожение продолжения, Notify/уничтожение TAsyncEvent, Cancel области отмены и планирование runnable требуют активации нужного актора. Внешний поток отправляет событие или завершает универсальный объект ожидания через мост.
10. **Тайм-аут не задаёт жёсткий крайний срок.** Он запрашивает отмену дочерней операции и ждёт подтверждения. Обычное завершение всё ещё может опередить отмену после истечения срока.
11. **Освобождение ресурсов через RAII должно выдерживать принудительное уничтожение.** Принудительная очистка почтового ящика уничтожает кадры без возобновления их тел и может выполняться без пригодного TLS актора. Деструкторы должны освобождать память и связи, не полагаясь на Send или TActorContext.
12. **Ожидайте временные объекты непосредственно.** Неперемещаемые объекты ожидания библиотеки с атрибутом nodiscard предназначены для использования как непосредственные операнды co_await. Если сохранить обёртку или отделить вызов лямбды-корутины от её ожидания, ленивый кадр может сохранить ссылку на уже уничтоженное замыкание.

Два вида кадров

Свойство	Корутина актора верхнего уровня	Вложенная `async<T>`
Сигнатура	Нестатический метод IActor, возвращающий void	Функция, метод или лямбда, возвращающие `async<T>`
Запуск	Немедленный, initial_suspend — suspend_never	Ленивый, initial_suspend — suspend_always
Возвращаемый объект	Нет	Некопируемый и обычно неперемещаемый `async<T>`
Владелец	Список TActorTask актора	Непосредственное выражение co_await / декоратор
Время жизни актора	PassAway ждёт эту задачу	Живёт внутри ожидающей её задачи верхнего уровня
Результат	Нет	T или void; исключение повторно выбрасывается в co_await
Отмена	Корневой источник отмены	Наследуется от ожидающего родителя

Жизненный цикл кадра верхнего уровня

Вызов корутины актора, возвращающей void, логически выполняет следующие действия:

1. выделяет кадр корутины и копирует или перемещает в него все явные параметры, переданные по значению;
2. создаёт TActorAsyncHandlerPromise, который регистрирует себя в акторе как TActorTask;
3. немедленно выполняет тело, поскольку initial_suspend никогда не приостанавливает выполнение;
4. при фактической приостановке в co_await оставляет кадр в ожидании и возвращает void вызывающему коду;
5. при нормальном завершении уничтожает локальные объекты тела, достигает final suspend, уничтожает кадр и снимает регистрацию задачи в деструкторе promise-объекта;
6. при подтверждённой отмене уничтожает приостановленный кадр, а вместе с ним — все живые локальные объекты, объекты ожидания (awaiter), параметры и promise-объект;
7. при принудительной очистке почтового ящика DestroyActorTasks напрямую вызывает coroutine_handle::destroy: объекты C++ уничтожаются так же, но код тела больше не выполняется.

Если тело фактически ни разу не приостанавливается, регистрация, выполнение, завершение и уничтожение кадра происходят до возврата из вызова.

Вызывающий код не получает дескриптор задачи. Это подходит для точки входа обработки события или намеренно независимого фонового цикла актора. Вызов другой корутины, возвращающей void, из обработчика не ждёт её завершения, а запускает отдельную задачу того же верхнего уровня:

```
void Handle(TEvRequest::TPtr ev) {
    ChildRoot();              // sibling TActorTask; Handle does not wait
    co_await SomethingElse();
}

void ChildRoot() {            // also a top-level void coroutine
    co_await Work();
}
```

Предпочитайте вложенную `async<T>`, которую Handle ожидает через co_await:

```
void Handle(TEvRequest::TPtr ev) {
    co_await Child();
}

async<void> Child() {
    co_await Work();
}
```

Жизненный цикл вложенного кадра

Вызов функции `async<T>`:

1. создаёт её кадр и сохраняет копии параметров или ссылки на них;
2. останавливается на initial_suspend до выполнения тела;
3. возвращает объект `async<T>`, владеющий дескриптором кадра.

Затем происходит одно из двух:

- если объект async отбрасывается, его деструктор уничтожает кадр; локальные объекты тела ещё не создавались, а тело никогда не выполняется;
- при непосредственном ожидании выражение await устанавливает актор, источник отмены и продолжение родителя, затем передаёт выполнение дочерней корутине.

После завершения дочерней корутины final suspend возвращает выполнение родителю. Promise-объект сохраняет значение или исключение, пока await_resume не извлечёт его. В конце полного выражения co_await владеющий объект async уничтожает дочерний кадр. Исключение, сохранённое вложенной корутиной, повторно выбрасывается в родителе в точке co_await.

`async<T>` намеренно не является универсальным future или задачей, которую можно отделить от владельца. Не используйте UnsafeMove в обычном прикладном коде: он предназначен для библиотечных комбинаторов.

Кадры на рабочих потоках выделяются через кеш рабочего потока; это не меняет правил владения и времени жизни, описанных на этой странице. Лимит, статистика и вспомогательные средства для тестов описаны в разделе «Кеш кадров корутин».

Параметры, захваты и данные кадра

Специализация верхнего уровня выбирается только для:

```
template<IsActorSubClassType T, IsNonReferenceType... Args>
struct std::coroutine_traits<void, T&, Args...>;
```

Поэтому все явные аргументы должны передаваться по значению:

```
void Handle(TEvRequest::TPtr ev) { // correct: event ownership enters frame
    co_await Process();
}

void Handle(TEvRequest::TPtr& ev) { // not a stackless actor coroutine promise
    co_await Process();             // compile-time failure
}
```

hFunc передаёт свой TAutoPtr lvalue в обработчик с параметром по значению, передавая владение событием. Неявная ссылка this ожидаема и не входит в Args. Перегрузки Bootstrap для корутин также должны принимать явные параметры по значению. Для корутины предпочитайте `Bootstrap()` или `Bootstrap(TActorId parent)`. Копирование `TActorContext` не продлевает жизнь активации, на которую он ссылается; никогда не используйте этот контекст после приостановки. Установите Become до первой приостановки Bootstrap.

Вложенные функции `async<T>` могут иметь ссылочные параметры, но это обычные заимствования:

```
async<void> Use(const TString& value); // caller must keep value alive
```

Дескриптор события, переданный в корутину верхнего уровня по значению, остаётся безопасным с точки зрения памяти после приостановки, поскольку им владеет кадр. Однако так можно надолго удержать большой объём данных. Извлеките нужные небольшие поля и вызовите ev.Reset() перед длительным ожиданием.

У лямбд-корутин есть дополнительная ловушка времени жизни: их кадры обычно заимствуют замыкание лямбды через её указатель this. Следующие варианты безопасны:

```
auto operation = [value]() -> async<void> {
    co_await Use(value);
};
co_await operation(); // named closure outlives the await

co_await WithTimeout(TDuration::Seconds(1), [value]() -> async<void> {
    co_await Use(value);
}); // closure temporary lives through this co_await full-expression
```

Не создавайте и не сохраняйте возвращённый ленивый async или декоратор в одном полном выражении, чтобы затем ожидать его в другом. TTaskGroup::Add устроен иначе: он хранит копии функции обратного вызова и аргументов с типами после decay до завершения дочерней задачи.

В целом по возможности избегайте лямбд-корутин.

Минимальный актор с запросом и ответом

Определения событий опущены; существенны параметры по значению, уникальные cookie, вложенная вспомогательная функция, обработка тайм-аута, диспетчеризация устаревших ответов и состояние завершения.

```
#include <ydb/library/actors/async/async.h>
#include <ydb/library/actors/async/timeout.h>
#include <ydb/library/actors/async/wait_for_event.h>
#include <ydb/library/actors/core/actor_bootstrapped.h>
#include <ydb/library/actors/core/hfunc.h>

using namespace NActors;

class TClient final : public TActorBootstrapped<TClient> {
public:
    explicit TClient(TActorId backend)
        : Backend(backend)
    {}

    void Bootstrap() {
        Become(&TThis::StateWork);
    }

    void Handle(TEvStart::TPtr ev) {
        const TActorId replyTo = ev->Sender;
        const ui64 replyCookie = ev->Cookie;
        TString payload = ev->Get()->Payload;
        ev.Reset(); // do not retain the incoming envelope while waiting

        auto value = co_await WithTimeout(
            TDuration::Seconds(5),
            RoundTrip(std::move(payload)));

        if (value) {
            Send(replyTo, new TEvDone(*value), 0, replyCookie);
        } else {
            Send(replyTo, new TEvFailed("timeout"), 0, replyCookie);
        }
    }

    async<int> RoundTrip(TString payload) {
        const ui64 cookie = ++NextCookie;
        Send(Backend, new TEvBackendRequest(std::move(payload)), 0, cookie);
        auto reply = co_await ActorWaitForEvent<TEvBackendReply>(cookie);
        co_return reply->Get()->Value;
    }

    void BeginShutdown() {
        Become(&TThis::StateDying);
        PassAway();
    }

    STRICT_STFUNC(StateWork,
        hFunc(TEvStart, Handle);
        IgnoreFunc(TEvBackendReply); // late reply after timeout/cancellation
        cFunc(TEvents::TEvPoison::EventType, BeginShutdown);
    )

    STFUNC(StateDying) {
        Y_UNUSED(ev);
    }
```

```
private:
    const TActorId Backend;
    ui64 NextCookie = 0;
};
```

Каждый TEvStart запускает независимую задачу верхнего уровня. Эти задачи могут одновременно находиться в ожидании, но никогда не исполняются одновременно. ActorWaitForEvent быстро отменяется, поэтому именно этот тайм-аут может завершиться кооперативно. Другой дочерний объект ожидания может не обладать таким свойством.

ActorWaitForEvent сопоставляет только тип и cookie, но не Sender. В примере предполагается, что только Backend использует эти cookie ответов. Если требуется проверять отправителя, направляйте ответы через StateFunc или специальный поток протокола, который выполняет такую проверку.

Выполнение и возобновление

Обработчик верхнего уровня начинает работу внутри вызова StateFunc или Bootstrap и выполняется синхронно до приостановки. Пока он ждёт, Receive продолжает обрабатывать события почтового ящика:

Поступившая работа	Что происходит
Событие, соответствующее ActorWaitForEvent	Объект ожидания поглощает его и возобновляет корутину непосредственно внутри Receive.
Несовпадающее пользовательское событие	Оно попадает в текущий StateFunc и может запустить другую корневую задачу.
Запланированный runnable актора	Он выполняется при обработке текущей очереди runnable.
Системное событие TEvResumeRunnable	Возобновление происходит при обработке этого более позднего события почтового ящика.

TActorRunnableQueue устанавливается для каждого Receive. Такие примитивы, как TAsyncContinuation и TAsyncEvent, планируют runnable, чтобы не возобновлять другой кадр рекурсивно из Resume/Notify. Очередь обрабатывается до завершения Receive. ActorWaitForEvent намеренно устроен иначе: совпавшее событие возобновляет ожидающую корутину непосредственно и никогда не достигает StateFunc.

AsyncSleep, AsyncYield, конечные таймеры и универсальные мосты к внешним потокам используют переход через почтовый ящик с TEvResumeRunnable. Они не исполняют код актора в потоке планировщика или внешнем потоке. Вызовы Send в библиотеке передают TEvResumeRunnable::EventFlags (FlagSystemMessage). Если такое событие уничтожается необработанным и его Item всё ещё активен, ~TEvResumeRunnable вызывает Item->Run(nullptr). Если для деактивации события Item был установлен в nullptr, уничтожение ничего не делает. Библиотека может таким образом деактивировать ожидающие события, локальные для актора; события, принадлежащие планировщику, требуют отдельного моста, поскольку их уничтожение может происходить одновременно с выполнением актора.

Правила чередования выполнения

Каждый co_await, который может приостановить выполнение, рассматривайте как следующую последовательность:

```
сохранить состояние кадра
допустить произвольные последующие циклы выполнения актора
возобновить выполнение через почтовый ящик
перепроверить предположения
```

В частности:

- Become действует на весь актор; другой обработчик может изменить функцию состояния;
- запись в словаре, указатель, итератор, контекст транзакции или поколение могут быть удалены или заменены, пока этот кадр ждёт;
- ссылка на данные входящего события безопасна, только пока кадр владеет этим событием; сброс или пересылка дескриптора делает её недействительной;
- TActorContext никогда не безопасен после приостановки: он ссылается на активацию и рабочий поток предыдущего цикла выполнения;
- никогда не удерживайте мьютекс или guard транзакции/БД через приостановку;
- указатель this актора остаётся действительным, пока его задача верхнего уровня зарегистрирована, но у сырого указателя на другой актор такой защиты нет.

Надёжный приём для рабочего кода — хранить в кадре идентификатор запроса или поколение и после каждой приостановки заново искать принадлежащее актору состояние. Последовательная обработка почтового ящика не делает ранее сохранённый указатель логически действительным.

Примитивы корутин

Библиотека акторных корутин без собственного стека предоставляет ожидания, уведомления, сроки завершения, области отмены и группы задач. Их выполнение возобновляется в почтовом ящике актора-владельца. Этот справочник предполагает знакомство с правилами работы с кадрами и временем жизни из статьи Акторы на корутинах. Завершение операций из другого потока описано в статье Интеграция корутин.

Выбор примитива ожидания

Задача	Примитив
Дождаться одного ответа протокола с заданными типом и cookie	ActorWaitForEvent
Приостановиться, пока код этого же актора не вызовет Resume	WithAsyncContinuation
Разбудить одного или всех ожидающих в этом акторе, не сохраняя сигнал	TAsyncEvent
Дождаться заданного момента времени	AsyncSleepFor / AsyncSleepUntil
Однократно уступить выполнение, отправив собственное событие самому себе	AsyncYield / AsyncSleepFor(0)
Задать кооперативный срок завершения дочерней операции	WithTimeout / WithDeadline
Отменить часть работы	TAsyncCancellationScope
Структурированно запустить несколько задач и собрать их результаты	WithTaskGroup
Уступать выполнение из множества задач без переполнения почтового ящика событиями	TAsyncLowPriorityQueue
Завершить операцию из внешнего потока	Адаптер обычного awaitable-объекта либо ActorSystem::Send вместе с ActorWaitForEvent

ActorWaitForEvent

```
const ui64 cookie = ++NextCookie;
Send(peer, new TEvRequest, 0, cookie);
auto ev = co_await ActorWaitForEvent<TEvReply>(cookie);
```

Контракт:

- сопоставляет GetTypeRewrite и Cookie; Sender не проверяется;
- `ActorWaitForEvent<IEventHandle>` принимает любое несистемное пользовательское событие с этим cookie;
- сопоставление происходит до StateFunc и передаёт ожидающей корутине весь дескриптор события;
- выполнение возобновляется непосредственно внутри Receive, принявшего событие;
- отмена или уничтожение снимает регистрацию ожидания;
- событие с другим типом или cookie передаётся другим ожидающим, а затем в StateFunc.

Используйте новый cookie для каждой попытки запроса и не используйте его повторно, пока может прийти более ранний ответ того же типа: иначе поздний ответ на брошенную попытку завершит новое ожидание. По умолчанию Send использует cookie 0, поэтому не ожидайте ответа с этим значением. AllocateWaitCookie возвращает ненулевые значения из общего для процесса счётчика; ActorRequest выделяет такой cookie, отправляет запрос и возвращает ожидание за один шаг. Если тип и cookie совпадают у нескольких ожидающих, событие разбудит первого зарегистрированного из них; такой код не выражает однозначного намерения.

Отправлять запрос непосредственно перед ожиданием безопасно в пределах одного цикла выполнения актора: почтовый ящик не сможет обработать ответ, пока этот код не приостановится. Ответ, который уже был обработан до запускающего события, восстановить нельзя.

После отмены или тайм-аута поздний ответ больше не перехватывается и попадает в StateFunc. Актор с STRICT_STFUNC должен явно обрабатывать или игнорировать типы устаревших ответов. Проверки поколения по-прежнему нужны, поскольку cookie со временем может использоваться повторно.

AsyncSleep и AsyncYield

```
co_await AsyncYield();
co_await AsyncSleepFor(TDuration::MilliSeconds(10));
co_await AsyncSleepUntil(monotonicDeadline);
```

- Sleep всегда приостанавливает выполнение.
- Нулевая длительность отправляет самому актору TEvResumeRunnable, поэтому выполнение возобновится при обработке более позднего события почтового ящика. AsyncYield — это такое ожидание нулевой длительности.
- Бесконечная длительность или срок не планирует событие и ждёт только отмены либо принудительного уничтожения.
- Конечное ненулевое ожидание использует событие планировщика и объект-адаптер со счётчиком ссылок.
- Отмена деактивирует адаптер, но не может удалить элемент планировщика; отменённое длительное ожидание удерживает небольшой объём выделенной памяти

до истечения срока или завершения работы системы.

Для сроков, отсчитываемых по прошедшему времени, предпочитайте TMonotonic. Sleep учитывает акторный контекст и должен запускаться из акторной корутины.

WithAsyncContinuation

Используйте этот примитив, когда корутина должна дождаться работы, которую уже выполняет **этот же актор**, но её завершение нельзя перехватить как событие почтового ящика: `Execute` / `Complete` транзакции таблетки, список ожидающих, который заполняет последующий `Handle`, или кеш, вызывающий обработчик после заполнения. Вы сохраняете `TAsyncContinuation<T>`, а в одном из последующих циклов выполнения **этого же** актора вызываете `Resume`.

Не используйте его для следующих задач:

- ответ другого актора — для этого есть `ActorWaitForEvent`, поскольку само событие и есть сигнал пробуждения;
- несколько ожидающих изменения флага без возвращаемого значения — для этого есть `TAsyncEvent`;
- завершение в другом потоке — отправьте событие через `Send` или используйте `co_await` с обычным `TFuture`. `Resume` не является потокобезопасным.

Лямбда настройки выполняется синхронно внутри `await_suspend`. Она должна только принять продолжение и подключить его к существующему механизму обратного вызова. Пример по образцу рабочего кода DataShard для ожидания блокировки строк или идентификатора транзакции:

```
void Handle(TEvNeedTx::TPtr ev) {
    const TActorId replyTo = ev->Sender;
    const ui64 replyCookie = ev->Cookie;
    ev.Reset();

    ui64 txId = co_await WaitForTxId();
    Send(replyTo, new TEvTxId(txId), 0, replyCookie);
}

async<ui64> WaitForTxId() {
    if (!TxIdCache.empty()) {
        ui64 id = TxIdCache.back();
        TxIdCache.pop_back();
        co_return id;
    }
    co_return co_await WithAsyncContinuation<ui64>(
        [this](TAsyncContinuation<ui64> c) {
            TxIdWaiters.push_back(std::move(c));
            // This example's allocator returns one ID per request, so each
            // cache miss needs its own allocation request.
            Send(TxAllocator, new TEvAllocateTxId);
        });
}

void Handle(TEvAllocateTxIdResult::TPtr ev) {
    ui64 txId = ev->Get()->TxId;
    // Still this actor. Resume parked waiters; check bool after cancel.
    while (!TxIdWaiters.empty()) {
        auto c = std::move(TxIdWaiters.front());
        TxIdWaiters.pop_front();
        if (c) {
            c.Resume(txId);
            return;
        }
    }
    TxIdCache.push_back(txId);
}
```

`TTxLockRows` использует тот же подход без пользовательского события: запускается транзакция, которая хранит `TAsyncContinuation<void>` и вызывает `Resume()` из `Execute` или `Complete`.

Продолжение можно только перемещать, а завершить — только один раз:

- `Resume(value)` завершает ожидание нормально; для типа `void` используется `Resume()`;
- `Throw(exception_ptr)` заставляет `await_resume` выбросить исключение;
- уничтожение активного продолжения или присваивание поверх него завершает ожидание с `logic_error("continuation object was destroyed")`;
- `Resume` / `Throw` для пустого продолжения выбрасывает `logic_error`.

Вызов `Resume` внутри лямбды настройки завершает ожидание без приостановки. Последующий `Resume` / `Throw` или уничтожение ставит действие в очередь выполнения; ожидающая корутина не запускается рекурсивно до возврата из `Resume`.

Если ожидающую корутину отменяют, сохранённое продолжение отсоединяется и преобразуется в false. Проверяйте его перед последующим завершением. Все операции с активным продолжением, включая уничтожение, требуют очереди выполнения этого актора.

TAsyncEvent

TAsyncEvent — локальная для актора очередь уведомлений, которая не сохраняет сигнал при отсутствии ожидающих:

```
bool notified = co_await Changed.Wait();
Changed.NotifyOne();
Changed.NotifyAll();
```

- NotifyOne будит ожидающего, который первым встал в очередь; NotifyAll будит всех, кто ожидает в данный момент.
- Notify при отсутствии ожидающих теряется. Событие не запоминает сигнал и не хранит счётчик.
- При уведомлении Wait возвращает true.
- Уничтожение события отсоединяет текущих ожидающих, и они возвращают false; это нормальное завершение, а не отмена.
- Отмена ожидающего, пока он ещё в очереди, удаляет его из очереди и запускает раскрутку его кадра вместо возврата false.
- Когда действие уведомления или отсоединения уже поставлено в очередь выполнения, последующая отмена обычно не срабатывает и Wait завершается нормально.

Notify и уничтожение события с ожидающими требуют активации актора-владельца. Не используйте один TAsyncEvent в несвязанных акторах или почтовых ящиках и не обращайтесь к нему из внешнего потока.

Wait(callback) сначала регистрирует объект ожидания, а затем синхронно вызывает callback. Это удобно для счётчиков и логики запуска: между регистрацией и вызовом callback уведомление не потеряется.

WithTimeout и WithDeadline

```
bool completed = co_await WithTimeout(
    TDuration::Seconds(1),
    DoVoidWork());

std::optional<int> value = co_await WithDeadline(
    deadline,
    DoValueWork());
```

Для дочерней операции `async<void>` результат имеет тип bool, для `async<T>` — `optional<T>`. true или значение означает нормальное завершение. false/nullopt означает, что отмена по тайм-ауту была подтверждена.

Последовательность обработки срока:

1. запустить таймер и ленивую дочернюю операцию;
2. если таймер сработал первым, запросить отмену дочерней операции;
3. дождаться подтверждения раскрутки её кадра;
4. только после этого вернуть false/nullopt.

Поэтому срок не задаёт жёсткую границу по реальному времени:

- объект ожидания дочерней операции, не поддерживающий отмену, может отложить возврат навсегда;
- после запроса отмены дочерняя операция может завершиться нормально — тогда возвращается её успешный результат;
- немедленно готовая операция может завершиться раньше нулевого тайм-аута;
- отмена вызывающей корутины, в том числе при PassAway, распространяется дальше как отмена и не приводит к возврату false;
- бесконечный тайм-аут не добавляет обёртку.

Гарантированно удалить запланированный таймер нельзя, поэтому деактивированный длительный тайм-аут удерживает небольшой объект-адаптер до истечения срока или завершения работы системы. Если внешний эффект операции может пережить локальный тайм-аут, задайте в протоколе правила проверки поколения и идемпотентности.

WithTimeout/WithDeadline вместе с их callback или аргументом async должны оставаться непосредственным операндом одного и того же co_await.

Области отмены

TAsyncCancellationScope отменяет выбранные дочерние операции независимо от корневой отмены актора.

Его деструктор не отменяет работу. Вызывайте Cancel явно. Область запоминает отмену; повторный Cancel идемпотентен. Приёмник отмены (sink), подключённый после отмены, отменяется сразу. Cancel и операции с активными приёмниками выполняются локально в акторе.

Wrap

```
TAsyncCancellationScope Scope; // actor member; outlives wrapped work

void Handle(TEvStart::TPtr ev) {
    bool completed = co_await Scope.Wrap([this]() -> async<void> {
        co_await WaitForWork();
    });
    // Inspect completed here if this handler itself was not cancelled.
```

```
}

void Handle(TEvCancel::TPtr& ev) {
    Scope.Cancel(); // another turn can run while the first handler waits
}
```

`Wrap<void>` возвращает bool, а `Wrap<T>` — `optional<T>`. false/nullopt означает, что отмена области подтверждена. Нормальное завершение может опередить запрос отмены — тогда возвращается успех. Отмена вызывающей корутины или корневой задачи обычно раскручивает кадр за пределы Wrap вместо возврата false.

Уже отменённая область всё равно запускает ленивое тело. Его синхронная начальная часть может выполниться, после чего корутина должна начать раскрутку кадра при первой приостановке, поддерживающей отмену.

Подключение всего корневого обработчика

Допустимо только внутри обработчика верхнего уровня с возвращаемым типом void:

```
Request.Scope =
    co_await TAsyncCancellationScope::WithCurrentHandler();
```

Возвращённая область содержит этот корневой обработчик как приёмник отмены. Последующий Request.Scope.Cancel() запрашивает отмену всего обработчика. Используйте вызов непосредственно в корневом обработчике: он опирается на специальный await_transform его объекта promise.

Это удобно для запросов с ключами, которыми владеет актор: храните область рядом с состоянием запроса, отменяйте её при дубликате или явной отмене и используйте RAII-объект для удаления состояния и уведомления о завершении. Поскольку отмена отложенная, перед повторным использованием ключа замещающему запросу может потребоваться дождаться явного уведомления через Finished TAsyncEvent.

WrapShielded и InterceptCancellation

WrapShielded не пропускает отмену вызывающей корутины в обёрнутую дочернюю операцию до её завершения. Он не сбрасывает уже зафиксированную отмену вызывающей корутины, а явный scope.Cancel по-прежнему отменяет дочернюю операцию. Такая защита может отложить PassAway навсегда; используйте её только для очистки, которая действительно обязана завершиться.

InterceptCancellation(body, onCancel) — низкоуровневый механизм очистки, которую нужно выполнить при отмене вызывающей корутины. onCancel может вернуть:

- void или true, чтобы распространить отмену;
- false, чтобы отклонить её распространение;
- `async<void>` или `async<bool>`, чтобы дождаться обработки отмены.

Исключение из onCancel выбрасывается повторно после внутренней очистки. Предпочитайте обычный RAII, если не требуется асинхронная обработка отмены или возможность отклонить её.

Группы задач

WithTaskGroup предоставляет структурированный кооперативный запуск нескольких задач. Дочерние задачи — это кадры `async<T>` в том же акторе, а не потоки.

```
int total = co_await WithTaskGroup<int>(
    [&](TTaskGroup<int>& group) -> async<int> {
        group.Add([this]() -> async<int> {
            co_return co_await Fetch(1);
        });
        group.Add([this]() -> async<int> {
            co_return co_await Fetch(2);
        });

        int sum = 0;
        while (group.Running() || group.Ready()) {
            sum += co_await group.Next();
        }
        co_return sum;
    });
```

Семантика и особые случаи:

- Add возвращает монотонно назначаемый индекс дочерней задачи.
- Add хранит копии callback и аргументов после преобразования decay и планирует запуск через очередь выполнения; callback не вызывается рекурсивно.
- Если тело группы возвращает управление до запуска запланированной дочерней задачи, её callback может вообще не выполниться.
- Результаты попадают в очередь в порядке завершения.
- Next забирает значение и повторно выбрасывает исключение дочерней задачи.
- NextResult возвращает `TTaskGroupResult<T>` с GetIndex и значением либо исключением; используйте HasValue/HasException и ExtractValue/ExtractException.
- WhenReady ожидает результат, не забирая его.
- У группы может быть только один ожидающий. Ожидание при отсутствии готовых результатов и незавершённых задач выбрасывает исключение.
- operator bool и Running учитывают только незавершённые задачи. Готовые, но ещё не забранные результаты не учитываются, поэтому для полного сбора

результатов проверяйте Running() || Ready().

Когда тело группы возвращает управление, выбрасывает исключение или подтверждает отмену, обёртка отменяет все незавершённые дочерние задачи. Она возвращает управление внешней вызывающей корутине только после завершения их кооперативной раскрутки. Дочерние задачи не продолжают работу отдельно от группы.

Особенность RAII: автоматические локальные переменные корутины тела группы уничтожаются при выходе из этого тела, до того как обработка final-suspend начнёт отменять оставшиеся дочерние задачи. Дочерняя задача не должна заимствовать эти переменные, если тело может завершиться раньше неё. Сначала дождитесь таких задач либо захватывайте владеющее состояние, время жизни которого охватывает внешнее ожидание WithTaskGroup.

При отмене актора тело группы должно сначала раскрутить свой кадр или вернуть управление, и только затем группа начнёт отменять дочерние задачи. Ожидание без поддержки отмены в теле группы или дочерней задаче может задержать выход из группы и уничтожение актора.

TAsyncLowPriorityQueue

Используйте этот примитив, когда корутинам нужно **дать почтовому ящику обработать другие события**, чтобы продолжать обслуживать клиентов, но не отправлять отдельный `TEvResumeRunnable` от каждого ожидающего.

`AsyncYield` / `AsyncSleepFor(0)` отправляет по одному событию самому актору на каждого ожидающего. Пятьдесят обработчиков, одновременно уступающих выполнение, поставят в очередь пятьдесят событий; одна активация может обработать их пачкой и задержать входящую работу до конца кванта.

`TAsyncLowPriorityQueue` — общий член актора: одновременно в пути находится не более одного события возобновления. `co_await Idle.Next()` приостанавливает корутину до **следующего цикла почтового ящика**, а затем возобновляет **одного** ожидающего. Обрабатывая это событие возобновления, очередь сразу отправляет следующее. Оно будет обработано в более позднем цикле после пользовательских событий, уже стоящих в очереди. Ожидающие, которые присоединились после отправки текущего события, попадают в следующее поколение и не могут возобновиться в текущем цикле, даже если исходный ожидающий был отменён.

Типичный сценарий — разбить ресурсоёмкую работу на элементы, уступать выполнение после каждого из них и использовать одну очередь для нескольких обработчиков:

```
TAsyncLowPriorityQueue Idle; // actor member, one per actor

void Handle(TEvCompact::TPtr ev) {
    TVector<TKey> keys = std::move(ev->Get()->Keys);
    ev.Reset();
    co_await CompactKeys(std::move(keys));
}

async<void> CompactKeys(TVector<TKey> keys) {
    for (const TKey& key : keys) {
        CompactOne(key);     // keep this bounded
        co_await Idle.Next(); // other Handles run before the next key
    }
}

void Handle(TEvClient::TPtr ev) {
    // still served between CompactOne slices
    Send(ev->Sender, new TEvClientAck, 0, ev->Cookie);
}
```

Одного `co_await AsyncYield()` достаточно, если уступить выполнение нужно только одному кадру и общая политика не важна.

Все ожидающие должны принадлежать одному актору. Очередь должна жить дольше них; её уничтожение при наличии ожидающих аварийно завершает процесс. Отмена отсоединяет ожидающего и может деактивировать либо повторно использовать отправленное самому актору событие.

Интеграция корутин

Внешние обратные вызовы и операции с ключами требуют явного управления владением между циклами выполнения актора. Эта статья описывает взаимодействие с другими потоками, собственные объекты ожидания и фоновую работу на основе контрактов из статей Акторы на корутинах и Примитивы корутин.

Завершение из внешнего потока

Поддерживаются две модели. Выберите один путь завершения.

Отправка акторного события

Это наиболее понятный вариант, если у потока ввода-вывода или обработки завершений уже есть TActorSystem* и целевой ActorId:

```
// Actor:
const ui64 cookie = ++NextCookie;
StartIo(SelfId(), cookie);
auto done = co_await ActorWaitForEvent<TEvIoDone>(cookie);

// Foreign thread:
actorSystem->Send(
    new IEventHandle(actorId, senderId, new TEvIoDone(status), 0, cookie));
```

Во внешнем потоке используйте TActorSystem::Send, а не IActor::Send, SelfId().Send или TActivationContext::Send. Последним API нужен акторный контекст в локальном хранилище потока (TLS). Событие завершения — обычное событие почтового ящика; оно может прийти поздно, уже после отмены, поэтому StateFunc должен допускать такой ответ.

Чтобы уведомить несколько ожидающих, отправьте из внешнего потока одно событие через Send. В обработчике этого события обновите состояние актора и вызовите continuation.Resume или TAsyncEvent::NotifyAll.

Ожидание обычного awaitable-объекта C++

Объект ожидания без IsActorAwareAwaiter оборачивается в потокобезопасный прокси-объект. При приостановке библиотека создаёт адаптер, содержащий актор, ActorId, почтовый ящик и ActorSystem*. Внешняя операция получает дескриптор корутины-адаптера, а не кадра актора.

При возобновлении этого дескриптора:

- в той же активации почтового ящика адаптер ставит действие в очередь выполнения;
- в остальных случаях, в том числе во внешнем потоке, адаптер использует ActorSystem::Send для отправки TEvResumeRunnable;
- код актора возобновляется только тогда, когда целевой почтовый ящик обрабатывает эту работу.

NThreading::TFuture — обычный awaitable-объект. Подключите library/cpp/threading/future/core/coroutine_traits.h: одного future.h недостаточно для operator co_await. Future может завершиться в другом потоке, но акторную отмену он не поддерживает. Поэтому PassAway обычно ждёт завершения future; future, который никогда не завершится, может навсегда задержать уничтожение актора.

Обязанности обычного объекта ожидания:

- завершиться ровно один раз;
- для одного адаптера ровно один раз возобновить или уничтожить его; если протокол с поддержкой отмены предоставляет отдельные адаптеры нормального завершения и отмены, завершить ровно один из них и никогда не обращаться к обоим;
- не отправлять через Send второе пробуждение для того же ожидания;
- обеспечить последующее завершение или отмену, чтобы адаптеры не утекали;
- гарантировать, что захваченный TActorSystem ещё существует, когда позднее завершение из другого потока пытается поставить возобновление в очередь.

После принудительной очистки актора поздний TEvResumeRunnable обычно уничтожается без обработки: ~TEvResumeRunnable вызывает Item->Run(nullptr), который не должен возобновлять уничтоженный кадр актора. В момент отправки отправителю всё ещё нужен живой TActorSystem. Остановите внешние источники завершений до уничтожения акторной системы.

Операции, локальные для актора

Следующие операции нельзя завершать или выполнять из внешнего потока:

- Resume, Throw, присваивание или уничтожение активного TAsyncContinuation;
- Notify или уничтожение TAsyncEvent с ожидающими;
- Cancel для TAsyncCancellationScope с активными приёмниками отмены;
- TActorRunnableQueue::Schedule;
- обращение к дескрипторам продолжения объектов ожидания, учитывающих акторный контекст;
- IActor::Send, SelfId().Send и API TActivationContext.

Помечайте собственный объект ожидания как IsActorAwareAwaiter, только если все действия по возобновлению, отмене и уничтожению сериализованы в правильном акторе и почтовом ящике. Если пометить так объект ожидания, который завершается в другом потоке, он обойдёт адаптер и сможет запустить код актора в неверном потоке.

Собственный объект ожидания

Предпочитайте библиотечные примитивы. Если собственный объект ожидания всё же необходим, сначала определите, какой контракт он реализует.

Объект ожидания с поддержкой акторного контекста

Объявляйте IsActorAwareAwaiter = true только для пробуждения внутри актора. Такой объект может использовать TActorRunnableQueue::Schedule во время Receive, чтобы избежать рекурсивного возобновления. ActorWaitForEvent — исключение: он возобновляет выполнение непосредственно внутри обработки события.

Для кооперативной отмены объект ожидания может реализовать:

- await_cancel(cancellationHandle), вызываемый во время активной приостановки;
- await_cancelled(cancellationHandle), необязательный метод для случая, когда отмена уже зафиксирована до приостановки.

await_cancel может вернуть void/false для отложенной отмены, true для немедленной раскрутки кадра или дескриптор корутины, выполняющей отмену. В итоге должен сработать ровно один путь: нормальное продолжение или подтверждение отмены. Никогда не возобновляйте оба.

Если отмена уже зафиксирована, await_ready всё равно имеет приоритет, когда возвращает true. Для неготового объекта ожидания с поддержкой отмены этот случай обрабатывает await_cancelled. Если await_cancel существует, а await_cancelled нет, библиотека пропускает обычный await_suspend и сразу начинает раскрутку кадра. Объект ожидания без обоих методов отмены всё ещё может приостановиться обычным образом.

Деструктор объекта ожидания должен удалить все внешние указатели на продолжение: принудительное завершение может уничтожить кадр без нормального завершения операции. Не уничтожайте дескриптор подтверждения акторной отмены; возобновляйте его согласно протоколу.

Обычный объект ожидания

Не объявляйте IsActorAwareAwaiter. await_suspend получает дескриптор адаптера, который можно безопасно завершить из другого потока. Внешняя операция обязана ровно один раз возобновить или уничтожить его. По возможности объект ожидания всё же должен предоставлять работающий механизм отмены; иначе PassAway останется в ожидании нормального завершения.

Проверьте как минимум следующие случаи:

- готовность без приостановки;
- нормальное отложенное завершение;
- отмена до приостановки и во время неё;
- гонка нормального завершения с отменой;
- исключение из await_ready/await_suspend/await_resume;
- вызов PassAway, пока актор ожидает;
- принудительное завершение акторной системы и последующее позднее внешнее завершение;
- отсутствие двойного возобновления и утечек адаптера или кадра.

Типовые приёмы проектирования

Множество одновременных запросов

Каждый входящий запрос имеет собственную корутину `Handle` верхнего уровня. Несколько таких корутин могут одновременно ожидать, но выполняются они по-прежнему по одной. Это обычная модель: не переносите незавершённые запросы в написанный вручную автомат состояний без необходимости.

Если другое событие должно отменять конкретный запрос, храните отображение идентификатора запроса в область отмены, подключённую к этому обработчику. Следующий фрагмент отклоняет повторные активные идентификаторы ответом `TEvBusy`, определённым в приложении. Отмена не освобождает идентификатор, пока старый кадр действительно не завершит раскрутку.

```
struct TInFlight {
    TAsyncCancellationScope Scope;
};
THashMap<ui64, TInFlight> InFlight;

void Handle(TEvRequest::TPtr ev) {
    const ui64 id = ev->Get()->RequestId;
    const TActorId replyTo = ev->Sender;
    const ui64 replyCookie = ev->Cookie;
    TString payload = std::move(ev->Get()->Payload);
    ev.Reset();

    if (auto it = InFlight.find(id); it != InFlight.end()) {
        Send(replyTo, new TEvBusy, 0, replyCookie);
        co_return;
    }

    auto& slot = InFlight[id];
    slot.Scope = co_await TAsyncCancellationScope::WithCurrentHandler();
    Y_DEFER { InFlight.erase(id); };

    const ui64 cookie = ++NextCookie; // actor member; fresh for every attempt
    Send(Backend, new TEvWork(std::move(payload)), 0, cookie);
```

```
    auto done = co_await ActorWaitForEvent<TEvWorkDone>(cookie);
    Send(replyTo, new TEvDone(done->Get()->Value), 0, replyCookie);
}

void Handle(TEvCancel::TPtr ev) {
    if (auto it = InFlight.find(ev->Get()->RequestId); it != InFlight.end()) {
        it->second.Scope.Cancel();
    }
}
```

Храните в кадре идентификаторы и владеющие значения, а не указатели на элементы `InFlight`. После каждого ожидания заново находите элемент, если он ещё нужен. Протокол замены может отменить старый запрос и дождаться явного уведомления о завершении перед повторным использованием ключа. Никогда не перезаписывайте его область сразу после `Cancel`: отложенная очистка может удалить состояние замещающего запроса. Для каждой попытки используйте новый cookie протокола, чтобы поздний старый ответ не завершил замещающий запрос. StateFunc должен допускать ответы, ожидание которых уже закончилось.

Один фоновый цикл на ключ

Обработчик запроса запускается, ожидает, отвечает и завершается.
Фоновый цикл — корутина `void`, которая не ожидает отдельный клиентский запрос: она поддерживает подписку, повторные попытки, переподключение или наблюдение за ресурсом до отмены.

«Ключ» — любой объект, для которого нельзя допустить дублирования: идентификатор таблетки, удалённый ActorId или идентификатор потока данных. Если цикл для этого ключа уже работает, не запускайте ещё один.

Здесь `Handle` — обычный обработчик, не корутина: в нём нет `co_await`, поэтому он может принимать `TEv::TPtr&`. `WatchLoop` — акторная корутина **void**: `co_await` в методе с возвращаемым типом `void` создаёт вторую задачу верхнего уровня `TActorTask`. Вызов `WatchLoop(id)` запускает её до первой фактической приостановки, после чего `Handle` возвращает управление. Именно это и требуется: если бы `WatchLoop` возвращал `async<T>`, обработчику `Handle` пришлось бы вызвать его через `co_await` и ждать всё время существования подписки.

Не пишите `co_await WatchLoop(id)`.

```
struct TWatch {
    TAsyncCancellationScope Scope;
    ui32 Refs = 0;
};
THashMap<ui64, TWatch> Watches; // key = resource id

void Handle(TEvWatch::TPtr& ev) {          // sync handler: TPtr&
    const ui64 id = ev->Get()->Id;
    if (auto it = Watches.find(id); it != Watches.end() && !it->second.Refs) {
        // Application-defined reply: retry after the old loop has unwound.
        Send(ev->Sender, new TEvWatchRetry, 0, ev->Cookie);
        return;
    }
    auto& w = Watches[id];
    ++w.Refs;
    if (w.Refs == 1) {
        WatchLoop(id);                     // starts a sibling root task
    }
}

void Handle(TEvUnwatch::TPtr& ev) {         // sync handler
    auto it = Watches.find(ev->Get()->Id);
    if (it != Watches.end() && it->second.Refs && --it->second.Refs == 0) {
        it->second.Scope.Cancel();
    }
}

void WatchLoop(ui64 id) {                   // void + co_await => TActorTask
    Y_DEFER { Watches.erase(id); };
    auto it = Watches.find(id);
    Y_ABORT_UNLESS(it != Watches.end());
    it->second.Scope = co_await TAsyncCancellationScope::WithCurrentHandler();

    const ui64 cookie = ++NextCookie; // fresh for this subscription incarnation
    for (;;) {
        Send(Source, new TEvSubscribe(id), IEventHandle::FlagTrackDelivery, cookie);
        auto ev = co_await ActorWaitForEvent<IEventHandle>(cookie);
        ApplyUpdate(id, ev);               // re-find Watches[id] if needed
    }
}
```

`Y_DEFER` удаляет ключ при раскрутке кадра цикла. `Scope.Cancel()` из `TEvUnwatch` служит сигналом остановки; пока отмена не завершилась, новые запросы наблюдения должны повторяться позже. `ApplyUpdate` зависит от протокола: он должен отличать обновления от `TEvUndelivered`, проверять отправителя и реализовывать политику повторных попыток или выхода. Само по себе получение уведомления о недоставке не отменяет корутину. Так устроен механизм разрешения таблеток: один `TabletStateSubscriptionLoop` на подписавшийся актор, запускаемый из `SubscribeTabletState`, только если записи в отображении ещё нет.

Не пишите `co_await WatchLoop(id)` : цикл `void` не является `async<T>`. Не удерживайте указатель или итератор на элемент отображения через ожидание.

Несколько задач с ограниченным временем жизни

Используйте WithTaskGroup, когда родитель не должен завершаться, пока все дочерние задачи не завершатся нормально либо с раскруткой кадров. Не имитируйте независимые дочерние задачи вызовом нескольких методов void: так создаются несвязанные корневые задачи.

API внешних обратных вызовов

- Если завершение callback гарантированно происходит в этом акторе, адаптируйте его с помощью WithAsyncContinuation.
- Если завершение возможно в другом потоке, отправляйте событие через Send или предоставьте обычный объект ожидания.
- Если одно завершение будит много ожидающих в акторе, отправьте одно событие через Send и уведомите их через продолжения или TAsyncEvent внутри StateFunc.

Кеш кадров корутин

Кеш кадров корутин — локальный для рабочего потока набор списков свободных блоков памяти, которые повторно используются для кадров корутин без собственного стека. Он сокращает накладные расходы на выделение памяти из кучи и не меняет правила владения кадрами, времени жизни актора, отмены и принудительного уничтожения; эти правила описаны в разделах «Акторы на корутинах» и «Сбои и завершение работы». Читайте эту страницу, если настраиваете лимит кеша, читаете его статистику или тестируете выделение памяти.

Какие кадры используют кеш

Корутины актора верхнего уровня и все кадры `async<T>`, включая свободные функции, константные методы и лямбды-корутины, выделяют память через `TAsyncFrameCache::Allocate` и освобождают её через `TAsyncFrameCache::Free` с передачей размера. Компилятор может полностью устранить выделение памяти для корутины.

У каждого исполнительного потока пулов BASIC, IO и общих пулов есть один `TAllocationCache<TAsyncFrameCacheTag>`, общий для всех акторов, выполняющихся в этом потоке. Рабочий поток общего пула при переключении между пулами сохраняет тот же кеш. Вне исполнительного потока текущего кеша нет, и кадры выделяются из кучи.

Классы размеров и лимит

Кеш хранит семь списков свободных блоков по принципу «последним пришёл — первым вышел» (LIFO) ёмкостью 1, 2, 4, 8, 16, 32 и 64 KiB:

- запрос до 1 KiB включительно использует список 1 KiB, а больший запрос до 64 KiB включительно — наименьший подходящий список;
- запрос свыше 64 KiB обслуживается напрямую из кучи и никогда не сохраняется в кеше;
- запрос до 64 KiB включительно округляется до ёмкости списка даже вне рабочего потока или при нулевом лимите, поэтому любой рабочий поток позже может сохранить такой блок.

Лимит ограничивает суммарную округлённую ёмкость свободных блоков, которые хранит один рабочий поток. Возвращаемый блок, который не помещается в оставшийся лимит, освобождается в кучу; отдельного ограничения на список нет. Лимит не учитывает активные кадры, метаданные кеша и служебные данные аллокатора, поэтому он не ограничивает общий объём памяти корутин. Для выделенной памяти сохраняется выравнивание `__STDCPP_DEFAULT_NEW_ALIGNMENT__`; повышенное выравнивание кадров корутин не поддерживается.

Кадр не хранит указатель на кеш, из которого он выделен. Освобождение с передачей размера возвращает блок в кеш освобождающего потока с учётом его лимита, а вне исполнительного потока — в кучу. Поэтому кадр можно освободить в другом рабочем потоке или при завершении работы без TLS актора. Когда исполнительный поток останавливается, его кеш отвязывается, и последующие освобождения в этом потоке возвращают память в кучу. Свободные блоки, сохранённые в кешах рабочих потоков, освобождаются только при уничтожении системы акторов; живые кадры кеш никогда не освобождает.

Настройка лимита

Лимит по умолчанию — 4 MiB на рабочий поток (`TAsyncFrameCache::DefaultSizeBytes`). Значение `0` отключает сохранение блоков: кеш существует, но каждый возвращаемый блок освобождается в кучу. Чтобы задать другой лимит, зарегистрируйте подсистему `TAsyncFrameCache` в `TActorSystemSetup` до запуска системы акторов:

```
setup->RegisterSubSystem(std::make_unique<TAsyncFrameCache>(8 * 1024 * 1024));
```

Если настройки не регистрируют эту подсистему, система акторов регистрирует её с лимитом по умолчанию. Для серверов YDB значение `NKikimrConfig::TActorSystemConfig::AsyncFrameCacheSizeBytes` (по умолчанию `4194304`) передаётся в эту подсистему. Кеши рабочих потоков создаются при запуске системы акторов, поэтому новый лимит применяется только после перезапуска.

Статистика

В исполнительном потоке вызов `TAllocationCache<TAsyncFrameCacheTag>::GetCurrent()->GetStats()` возвращает статистику кеша этого рабочего потока: `HeapAllocations` (промахи кеша, включая запросы свыше 64 KiB), `CachedBytes` (округлённая ёмкость свободных блоков), `CachedFrames` и `SizeClasses` (число непустых списков). Вызывайте его только в потоке-владельце. Измеряйте показатели в потоке, выполняющем нагрузку, и прогревайте кеш в том же потоке.

Для всей системы акторов вызов `actorSystem->GetSubSystem<TAllocationCacheSubSystem>()->GetCachedStats(TAsyncFrameCache::FamilyId())` возвращает суммарные `CachedFrames` и `CachedBytes`. После запуска системы акторов его можно вызывать из любого потока без блокировки, но он должен завершиться до уничтожения системы акторов.

Сумма учитывает каждый исполнительный поток этой системы акторов один раз, включая остановленные потоки до уничтожения системы акторов. В неё не входят активные кадры, другие системы акторов и кеши, привязанные в тестах. Рабочие потоки обновляют счётчики relaxed-загрузками и записями, поэтому сумма приблизительна, пока потоки выделяют или освобождают память, и точна, когда кеши не изменяются.

Сборщик статистики системы акторов публикует эту сумму один раз за цикл сбора, обычно каждую секунду, как показатели текущего значения без разбивки по потокам, пулам или классам размеров:

Группа счётчиков	Показатели
`counters=utils/subsystem=async_frame_cache`	`CachedFrames` , `CachedBytes`
`counters=utils/subsystem=allocation_cache/family=AsyncFrames`	`CachedBlocks` , `CachedBytes`

Санитайзеры

Повторное использование блоков остаётся включённым в сборках с санитайзерами. Под AddressSanitizer (ASAN) свободные блоки полностью помечаются как недоступные, включая ссылку списка; кеш временно разрешает доступ к ссылке для собственных операций. В активном блоке доступны только запрошенные байты, а оставшаяся округлённая ёмкость помечена как недоступная, в том числе для выделений вне рабочих потоков или при нулевом лимите. Под MemorySanitizer (MSAN) содержимое повторно используемого кадра снова помечается как неинициализированное. Эти аннотации обнаруживают обращения по устаревшим указателям, пока блок свободен, но не могут обнаружить их после допустимого повторного использования того же адреса.

Тестирование выделения памяти

Детерминированный тест может привязать собственный кеш к текущему потоку, не полагаясь на планирование исполнительных пулов:

```
TAllocationCache<TAsyncFrameCacheTag> cache(64 * 1024);
TScopedAllocationCache<TAsyncFrameCacheTag> guard(&cache);
```

`TScopedAllocationCache` объявлен в `ydb/library/actors/testlib/scoped_allocation_cache.h`. Вложенная привязка при уничтожении восстанавливает предыдущую. Выделение, освобождение и `GetStats()` должны выполняться в потоке привязки; `cache.GetCachedStats()` считывает число свободных блоков из любого потока. Существующие тесты находятся в `ydb/library/actors/async/ut/frame_cache_ut.cpp` и `ydb/library/actors/core/ut/allocation_cache_ut.cpp`.

Реализация

Акторы не владеют кешами кадров: метод `IActor::GetAsyncFrameCache()` и учёт `LiveFrames` в исходном кеше удалены, а аргумент с актором для выбора кеша не нужен. Кеш кадров — одно из семейств общей подсистемы кешей выделения памяти; другой компонент может определить собственный тег и зарегистрировать подсистему `TAllocationCacheFamily` с отдельным лимитом.

Файл в `ydb/library/actors/core/`	Содержимое
`allocation_cache.h`	Классы размеров, выделение и освобождение, аннотации санитайзеров, статистика
`subsystems/allocation_cache.h`	`TAllocationCacheSubSystem`, кеши каждого исполнительного потока, `TAllocationCacheFamily`
`subsystems/allocation_cache_tls.h`	`TAllocationCache<TTag>::GetCurrent()`
`subsystems/async_frame_cache.h`	`TAsyncFrameCacheTag` и `TAsyncFrameCache`

Тестирование акторов

Тесты акторов управляют доставкой сообщений и переходами жизненного цикла, чтобы проверять поведение протокола. Выбирайте наиболее узкое существующее тестовое окружение, которое моделирует изменяемый контракт. Общие инварианты описаны в руководстве по системе акторов; для акторов хранения также нужны тесты сохранности данных и восстановления соответствующего компонента.

Отправные точки для тестирования

Изменение	Тестовая цель или окружение
Ядро акторов, диспетчеризация, регистрация, завершение работы	`ydb/library/actors/core/ut`
Кадры корутин без собственного стека, объекты ожидания, отмена	`ydb/library/actors/async/ut`
Поведение Interconnect	`ydb/library/actors/interconnect/ut`
Общее тестовое окружение акторов	testlib/test_runtime.h
Интеграция таблеток и компонентов YDB	Существующее окружение компонента, часто на основе core/testlib/test_client.h

Используйте граничные акторы (edge actors) для наблюдения за ответами, а механизмы диспетчеризации и наблюдения тестового окружения — для задержки или перехвата событий. Явно задавайте проверяемый момент в причинной последовательности событий вместо ожидания по реальным часам. Проверяйте режим планирования окружения: тесты с моделируемым временем не воспроизводят все гонки реальных исполнителей или внешних потоков.

При изменении протокола запросов проверяйте успешный ответ и значимые переходы при сбоях: запоздалые или повторные ответы, завершение после тайм-аута, смену сессии и остановку при незавершённых операциях. Для проверки сохранности данных перезапускайте компонент с его сохранённым состоянием и проверяйте наблюдаемое снаружи восстановление. Отдельно тестируйте принудительное уничтожение, если освобождение ресурсов может выполняться без TLS актора.

Сборка и запуск тестов описаны в Ya Make; применяйте к выбранным целям действующие инструкции рабочего пространства или личные настройки сборки.

Быстрое модельное время

В `TTestActorRuntimeBase` быстрое модельное время включено по умолчанию. Когда непосредственные события обработаны, но остаётся таймер или замороженный почтовый ящик, окружение пропускает реальное ожидание 10 мс перед проверкой этой работы и продвижением модельного времени. Если такой работы нет, ожидание сохраняется. Окружения с реальными потоками и диспетчеризация с `Quiet` сохраняют прежнее поведение.

Если тесту нужно дождаться завершения внешней операции до обработки модельных таймеров или продвижения модельного времени, оптимизацию можно отключить для его окружения. Для существующего окружения с именем `runtime` выполните следующий вызов до диспетчеризации событий и укажите внешнюю зависимость в комментарии к тесту:

```
runtime.SetFastSimulatedTime(false);
```

Это возвращает реальные ожидания перед проверкой, но не гарантирует, что внешняя операция завершится за время ожидания. Если тесту нужен определённый порядок завершения операций, предпочтительна явная синхронизация.

Пропущенные ожидания по-прежнему расходуют по 10 мс из бюджета ожидания диспетчеризации, задаваемого через `SetDispatchTimeout`. Успешная доставка выбранных отложенных событий сбрасывает этот бюджет, как и раньше. Бюджет не равен времени по реальным часам: быстрая диспетчеризация может исчерпать его без ожидания соответствующей длительности.

Проверки при ревью

- Изменяются ли все поля, принадлежащие актору, только во время его последовательного выполнения?
- Может ли какой-либо цикл выполнения в System/User блокироваться, активно ожидать, выполнять медленный ввод-вывод или удерживать блокировку, за которую конкурируют потоки?
- Выполняется ли блокирующий ввод-вывод в диспетчере IO или изолирован в IOPoolId?
- Устанавливает ли актор ActorName / тип активности до Register?
- Переходит ли каждый зарегистрированный сырой указатель во владение среды исполнения немедленно?
- Устанавливается ли принимающее состояние Bootstrap до возврата или первой приостановки?
- Принимают ли обработчики-корутины верхнего уровня все явные параметры по значению?
- Соблюдаются ли времена жизни данных события и TActorContext при приостановке?
- Вызывает ли остановка PassAway ровно один раз и допускает ли получение событий, пока задачи завершаются?
- Можно ли безопасно отменить каждое длительное ожидание, ограничить его тайм-аутом или принудительно уничтожить?
- Отличает ли сопоставление ответов устаревшую попытку от текущей и корректно ли обрабатываются устаревшие ответы?
- Обрабатывает ли удалённый протокол тайм-аут, повторный или неоднозначный результат и новую сессию Interconnect?
- Можно ли сериализовать большие события, ограничен ли их размер и своевременно ли они освобождаются?
- Поддерживаются ли явно отображения ServiceId и псевдонимы?

Проверки корутин при ревью

- Должна ли это быть корневая задача, или следует вернуть `async<T>` и дождаться её завершения?
- Имеют ли все явные параметры корневой задачи нессылочные типы?
- Вызывает ли Bootstrap-корутина Become до первой приостановки?
- Используется ли каждый `async<T>` непосредственно в co_await или в библиотечном комбинаторе, который сам служит непосредственным операндом co_await?
- Живут ли замыкания лямбд и все заимствованные ссылки до конца ожидания?
- Освобождаются ли большие данные входящего события перед длительной приостановкой?

- Новый ли cookie ActorWaitForEvent для каждой попытки запроса (не 0 и не используется повторно, пока может прийти более ранний ответ), обрабатывает ли StateFunc запоздалые ответы?
- Перепроверяются ли после каждой приостановки состояние актора, указатели, поколения и состояние остановки?
- Может ли отмена дойти до каждого потенциально бесконечного ожидания?
- Рассматривается ли тайм-аут как кооперативная операция с неоднозначным внешним результатом?
- Переключает ли PassAway актор в состояние завершения и следует ли сразу после него return/co_return?
- Остаётся ли каждый RAII-деструктор безопасным при принудительном уничтожении без TLS?
- Вызывается ли TAsyncCancellationScope.Cancel явно при окончании владения?
- Избегают ли дочерние задачи группы заимствования локальных объектов, которые могут исчезнуть раньше этих задач?
- Выполняются ли обращения к примитивам, учитывающим контекст актора, только при активации владеющего ими актора?
- Для завершения во внешнем потоке предусмотрены ли ровно одно пробуждение через мост или событие и порядок сохранения объектов при остановке системы?

Исходный код и тесты корутин

Пути заголовков ниже указаны относительно `ydb/library/actors/async/`; модульные тесты находятся в его каталоге `ut/`. Путь `core/actor.cpp` указан относительно `ydb/library/actors/`.

Тема	Исходный код	Тесты поведения
Promise-объекты корневых и вложенных корутин, передача отмены	async.h, async.cpp	async_ut.cpp
Ожидание события по cookie	wait_for_event.h	wait_for_event_ut.cpp
Продолжение, локальное для актора	continuation.h	continuation_ut.cpp
Уведомление, локальное для актора	event.h	event_ut.cpp
Ожидание времени и передача управления	sleep.h, yield.h	sleep_ut.cpp
Тайм-аут и крайний срок	timeout.h	timeout_ut.cpp
Области отмены и перехват	cancellation.h	cancellation_ut.cpp
Структурированное параллельное выполнение	task_group.h	task_group_ut.cpp
Низкоприоритетное планирование	low_priority.h	low_priority_ut.cpp
Связь корневых задач с завершением актора	core/actor.cpp	Модульные тесты async
Общий мост к внешнему потоку	async.cpp, async.h	Тесты универсального объекта ожидания в async_ut.cpp

Характерные примеры из рабочего кода, которые полезно изучать критически:

- ydb/core/tx/datashard/datashard__lock_rows.cpp: состояние запросов по ключу, WithCurrentHandler, освобождение ресурсов через Y_DEFER, отмена дубликатов и повторный поиск состояния после приостановки;
- ydb/core/tablet/tablet_resolver.cpp: локальные события актора, области отмены, фоновые циклы по ключу и защита при уничтожении;
- ydb/core/kqp/node_service/kqp_query_control_plane.cpp: асинхронный обработчик событий с параметрами по значению и ActorWaitForEvent.

Контракт определяется тестами и исходным кодом библиотеки; места вызова в рабочем коде служат примерами, а не нормативным описанием.

Распределённое хранилище YDB

YDB distributed storage — это подсистема YDB, которая отвечает за надёжное хранение данных.

Блобовый интерфейс позволяет хранить *блобы* (бинарные фрагменты размером от 1 байта до 10 мегабайт) с уникальным идентификатором. Подсистема также предоставляет примитивы DDisk и PersistentBuffer для прямого блочного доступа. Эти пути используют общую инфраструктуру управления устройствами и размещения, но имеют разные контракты данных и репликации.

Компоненты и границы ответственности

Компонент	Ответственность	С чего начать
DS proxy	Выполняет запросы к группам блобов, включая erasure-кодирование и обеспечение избыточности на уровне группы.	Исходный код DS proxy
VDisk	Хранит части блобов группы и поддерживает их индексы, синхронизацию, сборку мусора и состояние восстановления.	Исходный код VDisk
PDisk	Владеет физическим устройством, выделяет чанки владельцам и предоставляет журналирование и ввод-вывод устройства.	Исходный код PDisk
DDisk и PersistentBuffer	Предоставляют адресуемое блочное хранилище и сохраняемые промежуточные записи для клиентов прямого блочного доступа.	DDisk, PersistentBuffer
NodeWarden	Применяет конфигурацию хранения на узле и управляет локальными акторами PDisk, VDisk/DDisk и прокси.	NodeWarden
Контроллер BlobStorage (BSC)	Поддерживает пулы хранения, размещение, конфигурацию групп и закрепление ресурсов за группами прямого блочного доступа.	Исходный код контроллера

Обычный путь блобовых запросов — `tablet → DS proxy → VDisk → PDisk`. При прямом блочном доступе клиент, например партиция NBS, отправляет запросы непосредственно сервисам DDisk и PersistentBuffer. Этот клиент выбирает политику репликации, кворума, повторных попыток и переноса данных из PB. DDisk не реализует блобовый протокол VDisk или его алгоритм восстановления группы.

Группы прямого блочного доступа описывает размещение и выделение ресурсов для прямого блочного доступа. Остальная часть этой страницы посвящена обычному блобовому интерфейсу и группам VDisk.

Общие контракты

- Сервисные идентификаторы акторов обозначают расположение или сервис. Актор, обслуживающий такой идентификатор, может быть заменён при восстановлении; сам по себе идентификатор не доказывает, что запрос относится к текущему экземпляру.
- Поколение таблетки, поколение группы блобов, раунд владельца PDisk и токен соединения DDisk имеют разных владельцев и назначение. На каждом интерфейсе сохраняйте соответствующее значение защиты от устаревших запросов, не подменяя его другим.
- Закрепление ёмкости пула, локальное выделение чанков и завершённая запись данных — разные состояния. Успешное выделение ресурсов контроллером не подтверждает запись данных.
- Успех запроса ограничен областью ответственности получателя. Запись на локальное устройство или одна успешная реплика не означают автоматического подтверждения надёжного сохранения на уровне группы.
- Правила жизненного цикла акторов и обмена сообщениями применяются к таблеткам, сервисам хранения, акторам запросов и завершениям их асинхронных операций ввода-вывода.

Расположение исходного кода

Начните с `ydb/core/blobstorage`: там находятся `dsproxy/`, `vdisk/`, `pdisk/`, `ddisk/` и `nodewarden/`. Общие для нескольких компонентов определения событий и форматов также находятся в `ydb/core/base` и `ydb/core/protos`; транзакции размещения — в `ydb/core/mind/bscontroller`. Низкоуровневая реализация `TUringRouter` находится в `ydb/library/pdisk_io`, а политика партиции NBS — в `ydb/core/nbs/cloud/blockstore/libs/storage/partition_direct`.

Описание интерфейса distributed storage

Формат идентификатора блоба

Каждый блоб имеет 192-битный идентификатор, состоящий из следующих полей (в порядке, используемом для сортировки):

1. **TabletId** (64 бита) — идентификатор таблетки-владельца блоба.
2. **Channel** (8 бит) — порядковый номер канала.
3. **Generation** (32 бита) — номер поколения, в котором была запущена таблетка, записавшая данный блоб.
4. **Step** (32 бита) — внутренний номер группы блобов в рамках Generation.
5. **Cookie** (24 бита) — идентификатор, который можно использовать, если Step не хватает.
6. **CrcMode** (2 бита) — выбирает режим для избыточного контроля целостности блоба на уровне distributed storage.
7. **BlobSize** (26 бит) — размер данных блоба.
8. **PartId** (4 бита) — номер фрагмента при erasure-кодировании блоба; на уровне взаимодействия «BlobStorage ↔ таблетка» этот параметр всегда равен 0, что означает целый блоб.

Два блоба считаются различными, если у их идентификаторов отличается хотя бы один из первых пяти параметров (TabletId, Channel, Generation, Step, Cookie). Таким образом, нельзя записать два блоба, которые различаются только BlobSize и/или CrcMode.

Для целей отладки существует строковое представление идентификатора блоба, которое имеет формат `[TabletId:Generation:Step:Channel:Cookie:BlobSize:PartId]`, например, `[12345:1:1:0:0:1000:0]`.

При выполнении записи блоба таблетка выбирает параметры Channel, Step и Cookie. TabletId фиксирован и должен указывать на ту таблетку, которая выполняет запись, а Generation — на поколение, в котором запущена таблетка, выполняющая операцию.

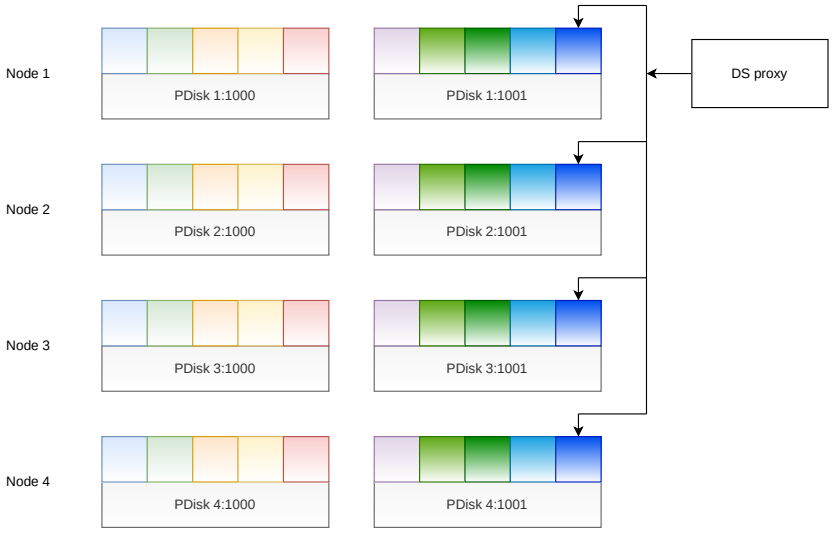
При чтении указывается идентификатор блоба, который может быть произвольным, но желательно ранее записанным.

Группы

Запись блобов производится в логическую сущность, называемую *группой*. На каждом узле для каждой группы, в которую осуществляется запись, создаётся специальный актор, который называется DS proxy. Этот актор отвечает за выполнение всех операций, связанных с группой. Создание этого актора производится автоматически через сервис NodeWarden.

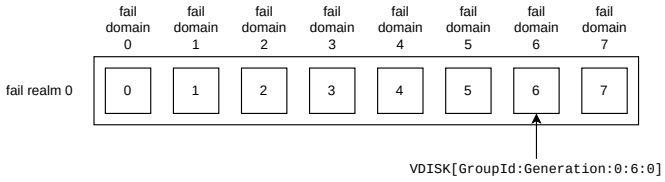
Физически группа представляет собой набор из нескольких физических устройств (блочные устройства в ОС), которые расположены на разных узлах таким образом, чтобы выход из строя одного устройства как можно меньше коррелировал с выходом из строя другого устройства. Как правило, эти устройства располагаются в разных стойках или в разных дата-центрах. На каждом из этих устройств для группы выделено место, которое управляется специальным сервисом, называемым *VDisk*. Каждый VDisk работает поверх блочного устройства, от которого он отделён другим сервисом — *PDisk*. Блобы разбиваются на фрагменты в соответствии со стирающим кодированием, и на VDisk'и записываются строго фрагменты. Перед разбиванием на фрагменты может выполняться опциональное шифрование данных в группе.

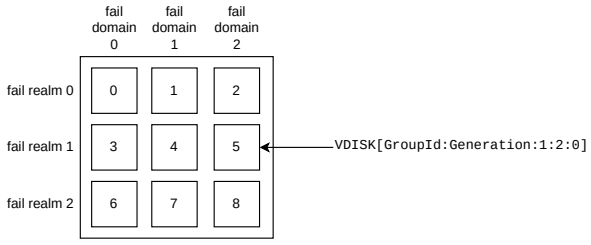
Схематично это показано на рисунке ниже.



Разноцветными квадратиками выделены VDisk'и разных групп; один цвет означает одну группу.

Группу можно рассматривать как совокупность VDisk'ов:





Каждый VDisk внутри группы имеет порядковый номер. Диски нумеруются от 0 до N-1, где N — число дисков в группе.

Кроме того, диски в группе объединены в fail domain'ы, а fail domain'ы объединяются в fail realm'ы. Как правило, каждый fail domain имеет ровно один диск внутри (хотя теоретически возможно и больше, но на практике это не используется), а несколько fail realm'ов используется только для групп, которые размещают свои данные сразу в трёх дата-центрах. Так, каждый VDisk получает, помимо порядкового номера в группе, идентификатор, который состоит из индекса fail realm'а, индекса fail domain'а внутри fail realm'а и индекса VDisk'а внутри fail domain'а. Этот идентификатор в строковом виде записывается как `VDISK[GroupId:GroupGeneration:FailRealm:FailDomain:VDisk]`.

Все fail realm'ы имеют одинаковое число fail domain'ов, а все fail domain'ы — одинаковое число дисков внутри. Количество fail realm'ов, количество fail domain'ов внутри fail realm'а и количество дисков внутри fail domain'а образуют геометрию группы. Геометрия зависит от способа кодирования данных в группе. Например, для block-4-2 numFailRealms = 1, число numFailDomainsInFailRealm >= 8 (на практике используется только 8), numVDisksInFailDomain >= 1 (на практике строго 1); для mirror-3-dc numFailRealms >= 3, numFailDomainsInFailRealm >= 3, numVDisksInFailDomain >= 1 (используется 3x3x1).

Каждый PDisk имеет идентификатор, который складывается из номера узла, на котором он запущен, а также внутреннего номера PDisk'а внутри этого узла. Как правило, этот идентификатор записывается в виде NodeId:PDiskId, например, 1:1000. Зная идентификатор PDisk'а, можно вычислить сервисный ActorId этого диска и отправить ему сообщение.

Каждый VDisk запущен поверх определённого PDisk'а и имеет *идентификатор слота*, который состоит из трёх полей — NodeId:PDiskId:VSlotId, а также идентификатор VDisk'а, о котором было сказано выше. Строго говоря, существуют различные понятия: «слот» — это место, зарезервированное на PDisk'е, которое занимает VDisk, и сам VDisk — это элемент группы, который занимает определённый слот и выполняет над ним операции. По аналогии с PDisk'ами, зная идентификатор слота, можно вычислить сервисный ActorId запущенного VDisk'а и отправить ему сообщение. Для отправки сообщений из DS proxy в VDisk используется промежуточный актор, который называется *BS_QUEUE*.

Для диагностики распределения чанков, выделенных VDisk, используйте внутренний API, описанный в Отчёт об использовании дискового пространства VDisk.

Состав каждой группы не является фиксированным — он может меняться в процессе работы системы. Для этого вводится понятие «поколения группы». Каждой паре "GroupId:GroupGeneration" соответствует фиксированный набор слотов (вектор, состоящий из N идентификаторов слотов, где N — размер группы), в которых расположены данные всей группы. *Не следует путать поколение группы и поколение таблетки — они никак не связаны*.

Как правило, группы двух соседних поколений различаются не более чем на один слот.

Подгруппы

Для каждого блоба вводится специальное понятие *подгруппы* — это упорядоченное подмножество дисков группы, которое имеет строго фиксированное число элементов, зависящее от типа кодирования (число элементов в группе должно быть не меньше), на которых будут храниться данные этого блоба. Для однодатацентровых групп с обычным кодированием подмножество выбирается как первые N элементов циклической перестановки дисков в группе; перестановка зависит от хэша BlobId.

Каждый диск в подгруппе соответствует диску в группе, но ограничен по допустимым хранимым блобам. Например, для кодирования block-4-2 с четырьмя фрагментами данных и двумя фрагментами чётности (data part, parity part) функциональное назначение дисков в подгруппе следующее:

Номер в подгруппе	Допустимые PartId
0	1
1	2
2	3
3	4
4	5
5	6
6	1,2,3,4,5,6
7	1,2,3,4,5,6

В данном случае PartId=1..4 соответствует фрагментам данных (которые получаются разрезанием исходного блоба на 4 равные части), а PartId=5..6 — фрагментам чётности. Диски с номерами 6 и 7 в подгруппе называются *handoff-дисками*. На них могут быть записаны любые фрагменты, включая несколько. На диски 0..5 можно записывать только соответствующие им фрагменты блоба.

На практике при выполнении записи система пытается разместить 6 фрагментов на первых 6 дисках подгруппы, и в подавляющем большинстве случаев это проходит успешно. Однако если один из этих дисков недоступен, то операция записи не может завершиться успешно, и тогда в работу вступают handoff-диски — на них отправляются фрагменты тех дисков, которые не ответили вовремя. Может случиться так, что в результате сложных тормозов и гонок на один handoff-диск уйдёт несколько фрагментов одного блоба. Это допустимо, хотя с точки зрения хранения неэффективно — каждый фрагмент должен иметь свой уникальный диск.

Восстановление избыточности

В случае отказа физического диска YDB может автоматически реконфигурировать связанные с ним группы хранения. При этом не важно, связана ли недоступность диска с выходом из строя сервера целиком или нет. Автоматическая реконфигурация групп хранения снижает вероятность потери данных при возникновении последовательности множественных отказов, при условии, что между отказами проходит достаточно времени для завершения восстановления избыточности. По умолчанию реконфигурация начинается через 1 час после того, как YDB выявляет отказ.

Реконфигурация дисковой группы заменяет VDisk, размещённый на отказавшем оборудовании, новым VDisk, который система старается разместить на работоспособном оборудовании. Действуют те же правила, что и при создании новой группы хранения:

- новый VDisk создаётся в домене отказа, отличающемся от доменов отказа всех остальных VDisk в группе;
- в режиме `mirror-3-dc` новый VDisk размещается в той же области отказа, что и отказавший VDisk.

Для того чтобы реконфигурация была возможной, в кластере должны быть свободные слоты для создания VDisk в разных доменах отказа. При расчёте количества оставляемых свободными слотов следует учитывать вероятность отказа оборудования, длительность репликации, а также время, необходимое на замену отказавшего оборудования.

Процесс реконфигурации дисковой группы создаёт повышенную нагрузку на остальные VDisk группы и на сеть. Чтобы уменьшить влияние восстановления избыточности на производительность системы, суммарная скорость репликации данных ограничивается как на стороне VDisk-источника, так и на стороне VDisk-получателя.

Время восстановления избыточности зависит от объёма данных и производительности оборудования. Например, репликация на быстрых NVMe SSD может завершиться за час, а на больших HDD репликация может длиться более суток.

Реконфигурация дисковых групп может быть ограничена или оказаться полностью невозможной, если оборудование кластера состоит из минимально необходимого количества доменов отказа:

- при отказе всего домена реконфигурация теряет смысл, так как новый VDisk может быть размещён только в уже отказавшем домене отказа;
- при отказе части домена реконфигурация возможна, но нагрузка, ранее обрабатываемая отказавшим оборудованием, будет перераспределена только по оставшемуся оборудованию в том же домене отказа.

Если в кластере доступен хотя бы на один домен отказа больше, чем минимально необходимо для создания групп хранения (9 доменов для `block-4-2` и по 4 домена в каждой области отказа для `mirror-3-dc`), то при отказе части оборудования нагрузка может быть перераспределена по всему оставшемуся в строю оборудованию.

Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

NodeWarden

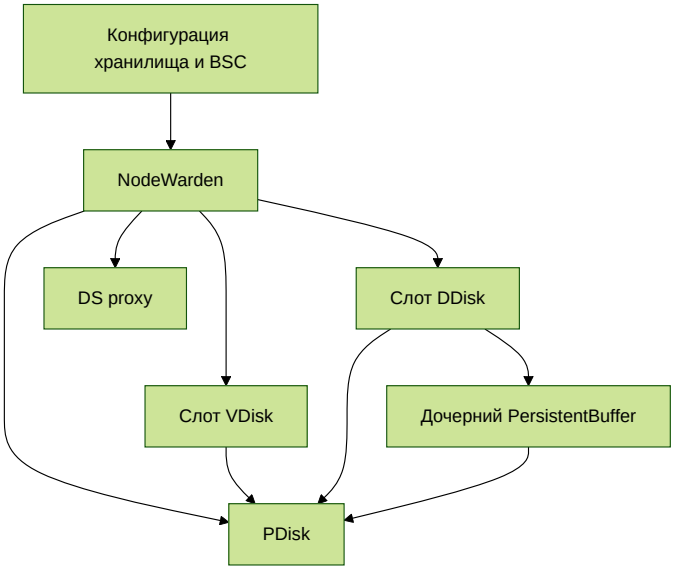
NodeWarden управляет дисковой подсистемой на отдельном узле. Он преобразует конфигурацию хранилища в работающие сервисы и обеспечивает их взаимодействие с контроллером и механизмами конфигурации групп. Эта страница объясняет, с чего начать при изменении логики создания и замены сервисов или распространения конфигурации.

Владение ресурсами

NodeWarden управляет локальными PDisk, слотами VDisk и DDisk, а также DS proxy. Он получает сведения о наборе сервисов и группах, запрашивает недостающую конфигурацию групп и запускает или обновляет соответствующие акторы. Код поддержки распределённой конфигурации в `distconf*` находится в том же каталоге; его следует отличать от автомата состояний отдельного дискового актора.

Если для группы слота установлен флаг DDisk, `StartLocalVDisk` создаёт `NDDisk::TDDiskActor` и регистрирует сервисный идентификатор DDisk. Исторические имена с VDisk в коде управления слотами не означают, что этот актор реализует протокол VDisk.

DDisk создаёт собственный дочерний PersistentBuffer после восстановления своего владельца PDisk и состояния карты чанков. Дочерний актор получает параметры PDisk родителя, чанки PB, формат диска и дубликат дескриптора устройства. Родитель регистрирует отдельный сервисный идентификатор PersistentBuffer. Поэтому к PB можно обращаться независимо, но его время жизни и ресурсы хранения принадлежат родительскому DDisk.



Стрелки показывают зависимости управления или владения ресурсами. DDisk и PB могут выполнять ввод-вывод данных через `TUringRouter`; схема не подразумевает, что каждая операция с данными проходит через актор PDisk.

Идентификаторы и замена акторов

Идентификатор	Значение
`NodeId:PDiskId`	Расположение сервиса физического диска в пределах узла.
`NodeId:PDiskId:VSlotId`	Расположение слота, используемое для адресации VDisk или DDisk.
`VDiskId`	Идентификатор, поколение и позиция в группе блобов; это не идентификатор занимаемого слота.
Владелец PDisk и раунд владельца	Текущий доступ локального владельца хранилища к ресурсам PDisk.
GUID экземпляра DDisk и токен соединения	Сведения об экземпляре актора и сессии, возвращаемые клиенту прямого блочного доступа.

Сервисные идентификаторы формируются функциями `MakeBlobStorage*Id` из blobstorage_service_id.h. Для DDisk и его дочернего PB в одном слоте используются разные функции. Парные записи данных и PB в логической группе прямого блочного доступа могут ссылаться на разные слоты; см. Группы прямого блочного доступа.

При проверке логики перезапуска или замены прослеживайте и незавершённые запросы старого актора, и регистрацию нового. Сохраняйте проверки раунда владельца и соединения, чтобы запоздалые завершения не воспринимались как работа для нового экземпляра. При завершении DDisk также нужно остановить дочерний PB и завершить либо оградить от дальнейших обращений доступ к общим ресурсам PDisk.

Границы конфигурации

`node_warden_vdisk.cpp` преобразует `DDiskConfig` и `PBufferConfig` в настройки C++, передаваемые DDisk. Сюда относятся выбор резервного пути прямого ввода-вывода, режимы контрольных сумм, ограничения выделения ресурсов и кеша PB, пакетирование записей, резерв для удаления и политика повторных попыток получения списка. Чтобы новый параметр можно было задавать через набор сервисов, он должен дойти и до реализации, и до этого слоя преобразования.

Локальный список DDisk, предоставляемый NodeWarden, содержит сервисные идентификаторы DDisk и PB для каждого локального слота DDisk. Это перечень локальных сервисов; он не описывает, какой таблетке принадлежит логическая группа прямого блочного доступа и какие слоты за ней закреплены.

Исходный код и тесты

Задача	Исходный код
Жизненный цикл сервисов и общее состояние	node_warden_impl.h, node_warden_impl.cpp

Запуск PDisk и владение ресурсами	node_warden_pdisk.cpp
Запуск и настройки VDisk/DDisk	node_warden_vdisk.cpp
Обновления групп и прокси	node_warden_group.cpp, node_warden_proxy.cpp
Перечень локальных DDisk	node_warden_resource.cpp
Создание дочернего PB и восстановление	ddisk_actor_boot.cpp

Тесты NodeWarden собираются целью `ydb/core/blobstorage/nodewarden/ut`; тесты последовательностей получения конфигурации находятся в `nodewarden/ut_sequence`. Замена дисков, запуск PB и завершение работы также проверяются отдельными тестами в `ydb/core/blobstorage/ddisk/ut` и `ddisk/ut_large`.

См. также

- Распределённое хранилище YDB
- DDisk
- PersistentBuffer

Группы прямого блочного доступа

Группа прямого блочного доступа (direct block group, DBG) — это принадлежащий таблетке набор выделенных DDisk для данных и PersistentBuffer (PB). Контроллер BlobStorage (BSC) сохраняет сведения об этом наборе и выбирает подходящие слоты. Клиент использует полученную схему размещения для реализации своей политики репликации и восстановления блоков.

Эта страница описывает общие контракты размещения и выделения ресурсов. Политики партиции NBS — основные и резервные роли, кворум записи, чередование регионов и планирование переноса данных из PB — описаны в руководстве по реализации партиции.

Три разных вида идентификаторов

Сущность	Идентификатор и значение
Группа пула DDisk в BSC	`TGroupInfo` с `IsDDisk = true`, содержащая слоты, подобранные для пула хранения DDisk. Она предоставляет ресурсы, из которых выделяются логические DBG.
Логическая DBG	Запись о закреплённых ресурсах с ключом `(TabletId, DirectBlockGroupId)` в `Schema::DirectBlockGroupClaims`, содержащая упорядоченные списки слотов данных и PB.
Сервис DDisk/PB	Сервис по адресу `(NodeId, PDiskId, DDiskSlotId)`. В слоте DDisk работают DDisk и его дочерний PB с разными идентификаторами акторов.

Между группами пула в BSC и логическими DBG таблетки нет соответствия один к одному. Аллокатор собирает подходящие VSlot из всех групп выбранного пула. Один слот может участвовать в нескольких логических DBG, а его роли для данных и PB учитываются отдельно.

`DirectBlockGroupIndex` в протоколе имеет ещё одно назначение: он выбирает сессию клиента и пространство имён записей PB. PB хранит этот индекс в одном байте, поэтому допустимый диапазон — `0..255`. Это не идентификатор группы BSC, а `DirectBlockGroupId` контроллера не сериализуется автоматически в этот индекс.

Определение пула и выделение ресурсов

`DefineDDiskPool` создаёт или обновляет пул хранения с флагом DDisk, типом erasure-кодирования `none`, геометрией размещения, фильтрами PDisk и параметром `NumDDiskGroups`. Группы ресурсов и слоты создаёт обычный механизм подбора групп пула. Само определение пула не закрепляет ресурсы DBG за таблеткой.

Затем клиент отправляет `TEvControllerAllocateDDiskBlockGroup` с идентификатором своей таблетки, именем пула данных, именем пула PB и запрошенными операциями. Контроллер сохраняет полученное распределение ресурсов и обновляет два счётчика каждого затронутого VSlot:

- `DDiskNumVChunksClaimed` — суммарное число виртуальных чанков, закреплённых для данных;
- `PersistentBufferRefs` — число ссылок на PB из логических DBG.

В неиспользуемом слоте пула оба счётчика могут быть равны нулю; в одном и том же слоте оба могут быть положительными. Ссылка на PB не резервирует определённый объём данных в PB. Фактическая ёмкость PB определяется локальными настройками выделения ресурсов и допуска запросов.

Закрепление чанков данных также относится к учёту в контроллере. Текущий аллокатор использует для кандидатов верхнюю границу закреплений `Max<ui32>()`; успешное закрепление не гарантирует, что на физическом устройстве уже есть столько свободных чанков. DDisk выделяет ресурсы для данных и контроля целостности по мере необходимости при записи.

Размещение и сопоставление пар

Размещение данных соблюдает ограничения пула на общую область отказа и разные домены отказа. Порядок закреплений и счётчики размещений используются для распределения выбранных DDisk по узлам, PDisk и слотам.

PB размещаются по геометрии собственного пула. При формировании начального списка PB аллокатор использует DDisk данных с соответствующим индексом как подсказку для совместного размещения. Сначала он пробует узел этого DDisk, затем предпочтительные узлы, если они указаны, затем другие подходящие узлы. Внутри узла он учитывает число ссылок на PB и может предпочесть другой слот, а не указанный слот данных.

Таким образом, `data[k]` и `pb[k]` образуют пару по позиции в списках, но совпадение идентификаторов слотов или узлов не гарантируется. Для переноса данных из PB нужно использовать возвращённый идентификатор PB, а не вычислять его из идентификатора DDisk данных.

Для двух ролей можно выбрать один пул или разные пулы. Один пул не обязывает размещать пару на одном PDisk. Два пула позволяют настраивать наборы кандидатов для каждой роли, но сами по себе не гарантируют разные физические устройства, если фильтры пулов пересекаются.

Число участников также входит в контракт клиента. Совместимый формат запроса выделения ресурсов определяет число участников для каждой роли по геометрии её пула. API явных операций содержит отдельные поля `NumDDisks` и `NumPersistentBuffers`. Фиксированную топологию NBS и поддерживаемое нагрузочным актором NBS-like число хостов нельзя считать универсальными константами протокола DDisk.

Формы запросов и ответов

Протокол определён в blobstorage.proto.

Форма	Назначение
`Queries`	Совместимый формат выделения ресурсов с `DirectBlockGroupId` и `TargetNumVChunks`; контроллер преобразует каждый запрос в операцию определения группы.
`DirectBlockGroupOperations`	Явное определение группы, изменение закреплений для отдельных DDisk, переназначение и удаление записей данных/PB.
`Responses`	Ответы совместимого формата с `ActualNumVChunks` и индексированными `Nodes`; каждая запись содержит идентификатор данных и, при наличии, идентификатор PB.
`DirectBlockGroups`	Ответы на явные операции с отдельными упорядоченными списками данных и PB.

В одном запросе нельзя смешивать `Queries` и `DirectBlockGroupOperations`. Перед принятием возвращённой топологии клиент должен проверить общий статус ответа. Явные операции могут возвращать списки разного размера, поэтому нельзя предполагать фиксированную парную структуру без проверки требований конкретного клиента.

Повторное определение группы с неизменными закреплениями использует сохранённое распределение ресурсов. Изменение топологии обновляет `DirectBlockGroupClaims` и счётчики затронутых VSlot в транзакции контроллера. Для потребителей сведений о топологии контроллер также поддерживает ревизию топологии каждой таблетки в `DirectBlockGroupTabletState`.

Таблетка должна сохранить успешно полученную схему размещения до создания тракта обработки данных, а после перезапуска заново подключиться к ней. Операции контроллера меняют закрепление ресурсов; они не копируют пользовательские данные и не доказывают, что реплику можно безопасно удалить. Клиент должен согласовать миграцию, завершение сессий, очистку PB и `TEvDeleteTabletChunks` с соответствующим изменением топологии.

Запуск

Выделенные идентификаторы указывают на сервисы, которыми управляет NodeWarden. NodeWarden запускает DDisk для каждого настроенного слота DDisk. Этот DDisk восстанавливает локальное состояние и создаёт дочерний PB независимо от того, какая логическая DBG впоследствии будет ссылаться на каждую из ролей.

Клиенты отдельно подключаются к каждому выбранному сервису DDisk и PB. Сессии DDisk возвращают непрозрачные токены для обычных запросов данных. Доступность сервиса, успешное установление сессии и завершение восстановления PB на стороне клиента — разные условия готовности.

Исходный код и тесты

- cmds_ddisk.cpp: определение пула и фильтры.
- config_fit_groups.cpp: группы ресурсов и слоты.
- ddisk.cpp: транзакция выделения ресурсов, размещение, закрепления и преобразование ответов.
- blobstorage_ddisk.proto: сохранённое распределение ресурсов и форматы запросов DDisk.
- ut_blobstorage/ddisk.cpp: интеграционные сценарии контроллера и тестового окружения хранилища.
- nbs_dbg_like_load_tablet_ut.cpp: выделение ресурсов клиентом, перезапуск, сценарии с несколькими DBG и несколькими таблетками.

См. также

- Распределённое хранилище YDB
- DDisk
- PersistentBuffer
- NbsDbgLikeLoad

DDisk

DDisk предоставляет хранилище с адресацией по блокам для клиентов прямого блочного доступа. Запрос указывает таблетку, виртуальный чанк и диапазон байтов. DDisk управляет отображением на локальные чанки PDisk, метаданными целостности и завершением ввода-вывода. Репликацию группы и кворум, определяющий успех пользовательского запроса, выбирает клиент.

DDisk использует общую с VDisk инфраструктуру PDisk и управления слотами, но реализует другой интерфейс. VDisk хранит части блобов для DS proxy; DDisk обрабатывает события из ddisk.h и blobstorage_ddisk.proto.

Владение ресурсами и запуск

NodeWarden создаёт актор DDisk для слота, настроенного как DDisk. Актор инициализирует своего владельца PDisk, восстанавливает снимки карты чанков и приращения из журнала, восстанавливает отображения метаданных целостности, освобождает резервы без восстановленных ссылок и создаёт дочерний PersistentBuffer. У дочернего актора собственные сервисный идентификатор и обработчики событий, но он разделяет с родителем владельца PDisk и жизненный цикл ресурсов PB.

Если `ForcePDiskFallback` не задан, DDisk запрашивает в `TEvYardInit` клиент io_uring, позволяющий только отправлять операции, и передаёт настроенное значение `IdleSpinUs`. При первом таком запросе PDisk дублирует дескриптор устройства, создаёт и запускает один `TUringRouter`, общий для слотов DDisk и их дочерних PB. Поэтому первый запрос задаёт `IdleSpinUs` общего роутера на время жизни этой инкарнации PDisk, а последующие запросы используют уже выбранное значение. DDisk и PB хранят разделяемые ссылки на `IUringRouterClient` и не могут управлять жизненным циклом роутера.

`ForcePDiskFallback` отключает общий роутер и выбирает путь через raw-события PDisk. Этот же резервный путь используется, если дескриптор устройства недоступен, платформа не поддерживается или проверка io_uring завершается неудачно. В нём `TEvChunkReadRaw` и `TEvChunkWriteRaw` передают владельца PDisk и раунд владельца. Журналирование и управление чанками продолжают использовать сервисы PDisk при любом бэкенде ввода-вывода данных.

`TDDiskConfig::DevNullMode` (по умолчанию `false`) меняет обработку ввода-вывода данных в общем роутере для одноразовых тестов: принятые записи завершаются без изменения устройства, а чтения возвращают нулевые буферы. Режим требует работающий io_uring, несовместим с `ForcePDiskFallback` и при недоступности роутера приводит к ошибке инициализации. Все слоты DDisk, подключающиеся к общему роутеру одного PDisk, должны запрашивать одинаковый режим; конфликтующий слот отклоняется, а для смены режима нужен перезапуск PDisk. Плоское поле NBS `DevNullMode` при явной установке, в том числе в `false`, переопределяет `GlobalDDiskConfig.DevNullMode`. В stress tool режим включается флагом `--ddisk-devnull`.

Форматирование PDisk, управление чанками и журнал отображений продолжают выполнять реальный ввод-вывод. Данные и метаданные целостности в режиме DevNull не сохраняются: после вытеснения метаданных или перезапуска чтение нулей не восстанавливает допустимый образ контрольных сумм. В тесте производительности с проверкой контрольных сумм нужно сначала записать нулевые значения в используемые блоки и удерживать их метаданные в кеше. Значение `IntegrityChecksumCacheBytes` должно быть достаточным для хранения всех используемых пар метаданных рабочего набора; нулевое значение отключает кеширование. Записанные ненулевые данные нельзя успешно прочитать с проверкой контрольных сумм, поскольку чтение данных возвращает нули. Для проверки холодного кеша, восстановления и сохранности данных используйте обычный ввод-вывод устройства.

Сессии

Клиент устанавливает сессию с помощью `TEvConnect` для каждого получателя DDisk или PB. Метаданные соединения содержат идентификатор таблетки, её поколение, индекс группы прямого блочного доступа и тип получателя. Сессии DDisk дополнительно используют `DDiskSessionSeqNo`; сессии PB не используют этот номер для различения сессий.

`TEvConnectResult` возвращает GUID экземпляра и непрозрачный `TConnectionToken`. Обычные запросы передают токен, по которому получатель находит серверные метаданные соединения. Клиентам следует использовать конструкторы токена в `TQueryCredentials`, а не формировать поля токена самостоятельно или сериализовать начальные метаданные в каждом запросе.

Соединения имеют ключ `(TabletId, DirectBlockGroupIndex)`. Более старое поколение или меньший номер сессии DDisk в том же поколении не может заменить активную более новую сессию: такой запрос соединения возвращает `BLOCKED`. Обычный запрос с устаревшими или недопустимыми реквизитами сессии возвращает `SESSION_MISMATCH`. GUID экземпляра генерируется для каждого нового экземпляра актора, а не только после обнаруженной потери данных. Подключение, отключение, замена сессии и перезапуск сервиса влияют на действительность токена. Перед повторными попытками операций по своей политике клиент должен восстановить действующее соединение.

Проверка чтения, записи и Sync возвращает реквизиты в нативном `TQueryCredentials`, не изменяя protobuf запроса. Запись и Sync сохраняют исходные реквизиты во время ожидания выделения чанка и владения метаданными. При включённых контрольных суммах их общая корутина `ExecuteDataWrite` получает все затронутые пары метаданных, затем повторно проверяет сессию до загрузки отсутствующих в кеше метаданных или отправки данных. Эта заключительная проверка — точка допуска: запрос, чей токен заменён во время ожидания, не загружает метаданные и не отправляет свои записи данных или метаданных. При отключённых контрольных суммах реквизиты повторно проверяются после выделения, непосредственно перед отправкой данных. Чтение проверяет реквизиты один раз, до поиска чанка, и не ожидает незавершённого выделения. Отключение и повторное подключение делают старый токен ожидающего запроса недействительным даже при неизменных поколении и номере сессии. Запись, отклонённая при допуске, возвращает `SESSION_MISMATCH`; Sync возвращает этот статус для затронутого сегмента и ошибку всего агрегата. После допуска последующая смена клиентского токена, в том числе во время загрузки отсутствующих в кеше метаданных, не отменяет запись и обновление метаданных целостности. Для ошибок устройства и остановки актора сохраняются отдельные правила обработки ошибок и завершения принятых операций. При проверке чтения известный старый токен описывается как «stale», а неизвестный — как «invalid». Повторное подключение без изменения токена остаётся допустимым.

Для внутренней пересылки DDisk/PB используется `TQueryCredentials::ForInternal`. Проверка таких запросов отличается от проверки обычного клиентского запроса и, в частности, допускает узел-получатель без существующего клиентского соединения. Это механизм взаимодействия между серверами, который не заменяет установление клиентской сессии.

Адресация и запись

`TBlockSelector` содержит `VChunkIndex`, `OffsetInBytes` и `Size`. Отображения чанков данных имеют ключ `(TabletId, VChunkIndex)`. Индекс группы прямого блочного доступа разделяет сессии и пространства имён PB, но не добавляет ещё одно измерение в карту чанков данных DDisk. Поэтому клиенты, совместно использующие DDisk от имени одной таблетки, должны согласованно назначать индексы виртуальных чанков во всех своих DBG.

Чтение и запись работают с непустыми диапазонами, выровненными по единице контроля целостности 4 КиБ и находящимися внутри чанка PDisk. Полезная нагрузка записи может быть фрагментированной или иметь невыровненные адреса буферов: при подготовке прямого ввода-вывода данные при необходимости копируются в выровненный буфер, а фрагменты горе с подходящим выравниванием могут использовать векторный ввод-вывод (scatter/gather). Используйте вспомогательные функции работы с полезной нагрузкой событий; идентификатор полезной нагрузки — это ссылка внутри события, а не постоянный идентификатор данных.

Размер публичного запроса `TEvWrite` ограничен 1 МиБ независимо от включения контрольных сумм. Более крупный запрос получает `INCORRECT_REQUEST` до выделения ресурсов или ввода-вывода. Запись ровно 1 МиБ допустима, в том числе текущий запрос копировщика данных. Клиенты с настраиваемым размером записи должны делить более крупные запросы; ограничение публичной записи не распространяется на большие чтения и сегменты Sync.

Вызывающая сторона гарантирует не более одного активного Sync для каждого набора блоков, непересекающиеся активные диапазоны данных и отсутствие обычных записей одновременно с чтениями и Sync. Операции Sync для непересекающихся наборов блоков могут выполняться одновременно. DDisk полагается на эти гарантии, но не реализует протокол их обеспечения. Непересекающиеся записи могут использовать общие пары метаданных целостности; DDisk сериализует изменения этих метаданных.

Счётчик `interface/UnalignedWritePayloads` учитывает входящие запросы записи с фрагментированной полезной нагрузкой или адресами буферов, не выровненными по размеру сектора устройства. Каждый запрос учитывается один раз, в том числе если его выполнение откладывается из-за выделения чанка.

Первая запись в виртуальный чанк может ожидать выделения ресурсов для данных и контроля целостности. Одновременные запросы к этому виртуальному чанку используют одну общую операцию выделения. При включённых контрольных суммах подтверждение записи ожидает завершения записи данных, обновления метаданных целостности и устойчивого сохранения отображения выделенного чанка. Обработчик перемещает маршрут ответа, исходные учётные данные, полезную нагрузку и контрольные суммы в запись, принадлежащую кадру корутины, и освобождает входное событие.

`ExecuteDataWrite` — единственная плоская корутина для целевых данных и метаданных как записи, так и Sync; запись — это одна порция, полезная нагрузка которой пришла вместе с запросом. При включённых контрольных суммах после выделения чанка она получает пары метаданных в возрастающем порядке, ожидает занятые пары и повторно проверяет сессию в точке допуска. Отсутствующий в кеше образ метаданных читается отдельным пакетом и преобразуется на потоке актора; для образа в кеше чтение не требуется. Только после этого записи данных и метаданных подготавливаются и отправляются одним пакетом. Обе операции завершаются до публикации метаданных на потоке актора и освобождения владения писателя. При ошибке всё равно ожидается завершение каждой принятой соседней операции, а ошибка загрузки или преобразования метаданных не допускает отправки данных. Успешный ответ требует устойчивого сохранения журнала отображения. Смена клиентского токена после допуска не прерывает запрос, включая загрузку отсутствующих в кеше метаданных. Закрепление чанка защищает отображение и физический чанк до завершения принятых операций и фиксации отображения.

Кадры корутин DDisk используют общий кеш кадров акторной среды, без собственного кеша актора или переопределения аллокатора. Показатели заполнения кеша измеряют свободные кадры, удерживаемые потоками исполнителей, а не живые кадры DDisk; завершение работы актора определяется реестрами запросов, закреплениями чанков и обработкой всех завершений.

Чтение проверяется в обычном обработчике, поэтому невыровненный диапазон, таблетка с удаляемыми чанками и виртуальный чанк без опубликованного физического чанка обрабатываются без выделения кадра корутины. Для выделенного и отформатированного чанка обработчик переносит маршрут ответа, разрешённые реквизиты, селектор и span в принадлежащую кадру запись и запускает `ExecuteDataRead`. При отключённых контрольных суммах корутина отправляет и ожидает данные. При включённых контрольных суммах используются четыре пути:

Путь	Обработка
Метаданные в кеше	Получить принадлежащие ответу контрольные суммы и сведения о незаписанных блоках; отправить и ожидать только данные, если диапазон не полностью нулевой.
Малое чтение, инициирующее загрузки метаданных	Отправить принадлежащее запросу чтение метаданных и данные одним пакетом. Опубликовать метаданные и завершить инициирующий запрос до уведомления ожидающих их запросов, когда все зависимости завершены.
Чтение от 32 КиБ (`MetadataFirstReadThreshold`) с отсутствующими метаданными	Дождаться метаданных, сохранить результат и освободить ожидающие метаданные запросы до последующего ожидания данных. Для полностью нулевого диапазона не читать данные.
Запрос, присоединившийся к существующей операции с метаданными	Сразу начать чтение данных, дождаться его, затем проверить сохранённый результат метаданных. Ожидать событие метаданных только при незавершённом результате.

Запрос может владеть частью загрузок и присоединиться к остальным. Он публикует завершённые загрузки и уведомляет ожидающие их запросы перед приостановкой на другой зависимости. Присоединившиеся запросы сохраняют результат завершения независимо от кеша: одной ссылки на `TAsyncEvent`, не сохраняющий сигнал, недостаточно, если метаданные завершены и вытеснены до окончания чтения данных. Если спекулятивное чтение данных оказалось полностью в никогда не записанных блоках, его завершение всё равно обрабатывается до возврата нулей.

`TBatchedIOAwaiter` объединяет ввод-вывод кадра в два этапа. Подготовка резервирует неподвижные слоты результатов и добавляет операции без отправки. Ожидание пакета публикует универсальный мост callback акторной среды и отправляет все подготовленные операции, удерживая дополнительную единицу в счётчике незавершённых операций. Она не позволяет немедленным callback завершить ещё не полностью отправленный пакет. Каждая операция завершается один раз, в том числе при повторных попытках. Кадр и отправленные операции удерживают пакет через `std::shared_ptr`; callback записывают отдельные слоты результатов и уменьшают атомарный счётчик. Последнее завершение возобновляет мост, который планирует продолжение в почтовом ящике актора. Если все операции завершились во время отправки, ожидающий объект возвращает дескриптор моста, и среда выполнения продолжает кадр без приостановки. Callback устройства никогда не возобновляют локальные продолжения актора напрямую.

Пакет поддерживает немедленное завершение и повторное использование после обработки всех результатов. Callback получают совместное владение пакетом только при отправке, поэтому подготовленные операции не образуют цикл владения. Кооперативная остановка не уничтожает кадр, удерживающий принятые буферы устройства. При принудительном уничтожении кадра ожидающий объект атомарно изымает и уничтожает мост, если его ещё не забрало завершение. Уже забранный callback мост использует обработку отсутствующего актора в среде выполнения и не может повторно войти в уничтоженный кадр. Принятые операции сохраняют общее хранилище callback до завершения.

До отправки ввода-вывода чтение сохраняет отправителя, сессию interconnect, клиентский cookie, идентификаторы таблетки и чанка и время начала во встроенном в корутину контексте ответа, затем освобождает входящее событие. Кадр удерживает span трассировки, а его общий `TBatchedIOAwaiter` владеет `TDDiskReadResult` и буферами завершённых операций, без реестра завершений и без выделяемого в куче события с результатом ввода-вывода. Каждая операция `TDDiskIoOp` удерживает объект callback до возврата в пул или уничтожения. Callback записывают непересекающиеся слоты, и именно это делает безопасным параллельное завершение операций io_uring в один пакет. Загрузки метаданных целостности дополнительно помечают свою операцию как критическую, что сохраняет правило обработки потери сессии для чтения метаданных. Span сохраняется до обработки заключительного результата, в том числе при отмене.

Логическое чтение использует обычную скалярную операцию роутера для диапазона данных, когда это нужно, и не более одного нового чтения метаданных. Метаданные нескольких полученных пар читаются одним непрерывным диапазоном, который разделяет менеджер целостности. Малое инициирующее чтение объединяет данные и метаданные в один пакет `TBatchedIOAwaiter`; большое сначала ожидает метаданные, затем решает, требуется ли чтение данных. Стабильные буферы и слоты результатов выделяются до отправки, а callback могут завершаться в любом порядке, в том числе до возврата вызова отправки. Роутер обрабатывает заполнение очереди и неполные чтения отдельно для каждой принятой скалярной операции.

Если роутер отклоняет отправку, DDisk локально завершает эту операцию через тот же путь callback и планирует переход в Stopping. Каждая оставшаяся подготовленная операция также получает заключительный результат. Пакет по-прежнему ожидает все принятые соседние операции, включая callback ошибок и сброса. При использовании PDisk fallback каждой части соответствует одно сообщение raw-read, которое объединяется тем же пакетом. Чтение присоединяется и к загрузкам метаданных, уже принадлежащим другой операции: его собственный ввод-вывод может завершиться раньше общей загрузки. Закрепления чанков и сам кадр сохраняются до завершения как ввода-вывода, так и общих метаданных. Буферы принятых операций сохраняются до окончания callback и обработки заключительных результатов на потоке актора.

`TReadPayload` владеет одним из трёх вариантов: отсутствием данных, нативным `TRcBuf` или резервным `TRope`. Callback ввода-вывода перемещают это владение в результат общего пакета, сохраняя нативные буферы до подготовки ответа. `FinishDDiskRead` обнуляет никогда не записанные блоки смешанного диапазона через вспомогательный интерфейс полезной нагрузки, затем один раз преобразует нативные клиентские данные в горе перед необязательной проверкой контрольных сумм и прикреплением к ответу. Успешно прочитанные данные передаются без копирования полезной нагрузки. Обработка состояния Broken, преобразование статусов, учёт байтов, счётчики неполного ввода-вывода и трассировка сохраняются; ответы с ошибкой не содержат ни полезной нагрузки, ни контрольных сумм. `TEvReadResult` сразу переносит `TConstArrayRef<ui64>` в protobuf, не сохраняя представление и не создавая промежуточную копию вектора контрольных сумм.

Чтения сохраняют только необходимые контрольные суммы и сведения о незаписанных блоках в принадлежащих им неизменяемых результатах. После получения эти результаты не закрепляют образы метаданных и остаются действительными при изменении соседних метаданных и вытеснении кеша. Пара в кеше остаётся доступной для чтения во время записи метаданных через прежний неизменяемый образ. Чтение, встретившее операцию чтения, изменения и записи отсутствующих в кеше метаданных, ожидает завершения заключительной записи. Эти свойства кеша не обеспечивают гарантии вызывающей стороны о диапазонах данных и взаимоисключении операций.

Чтение виртуального чанка, для которого ещё не выделены ресурсы, возвращает нули. Пока выделение, начатое другим запросом, не опубликовало физический чанк, виртуальный чанк считается невыделенным, поэтому параллельное чтение возвращает нули и не ожидает выделения. Выделение чанков и восстановленное состояние целостности влияют на то, как реализация распознаёт никогда не записанные блоки внутри уже выделенного чанка; успешное чтение не доказывает, что в этот диапазон ранее записывали данные.

Контроль целостности

Контрольные суммы в сетевом протоколе — это значения XXH3-64 без примеси идентификационных данных, по одному на блок полезной нагрузки размером 4 КиБ. Когда включён `TDDiskConfig::EnableChecksums`, обработчики записи отклоняют отсутствующие контрольные суммы, неверное их количество и несовпадающие значения до выделения ресурсов или запуска ввода-вывода.

DDisk хранит метаданные целостности отдельно от данных. Сохраняемые контрольные суммы связываются с логическими и физическими идентификаторами, а сетевой протокол и кеш контрольных сумм используют чистую контрольную сумму полезной нагрузки. Каждый блок метаданных целостности использует пару слотов, собственные контрольные суммы, проверки идентификаторов и поколений, а также порядковый номер для выбора допустимой сохранённой версии после восстановления. Реализация поддерживает схемы для разных свойств атомарной записи устройства; их точный формат определён в ddisk_checksums.h.

`TIntegrityManager` не выполняет физический ввод-вывод. Он хранит стабильные записи кеша в хеш-таблице с отдельными узлами и состояниями `Missing`, `WaitingRead`, `Data` и `WaitingWrite`. При подготовке каждая нужная пара ищется один раз; стабильные ссылки сохраняются, пока операция нуждается во владении. Компактные битовые карты использованных блоков, ожидаемые дайджесты и сведения о текущих слотах каждой пары хранятся независимо от вытесняемых образов контрольных сумм. Выбор слота, проверки идентичности и дайджеста и обнаружение потерянных записей поэтому сохраняются после вытеснения.

`PrepareRead` создаёт немедленный результат в пустом optional вызывающей стороны либо возвращает дескриптор ожидания и не более одного описания новых загрузок метаданных. К существующим загрузкам запрос присоединяется без дублирования. `TReadChecksums` хранит единственную контрольную сумму внутри объекта и владеет хранилищем более крупных снимков. Одноблочный результат выбирает нули или сквозное чтение без маски; многоблочный результат хранит маску только для смешанного диапазона. Результаты владеют контрольными суммами и сведениями о незаписанных блоках, не удерживая кешированные метаданные. Дескрипторы ожидания сохраняют заключительный результат после уведомления и вытеснения.

Подготовка чтения предоставляет `MetadataReads` типа `TMetadataReads` (`absl::InlinedVector<TMetadataRead, 1>`). Необязательный дескриптор задаёт непрерывное чтение метаданных с непрозрачным идентификатором; DDisk передаёт `TMetadataReadResult` в `CompleteMetadataReads`. Чтение охватывает диапазон от первой заново полученной пары до последней, а менеджер извлекает из этого образа два слота целостности каждой полученной пары. В базовом формате пара покрывает 492 блока данных, поэтому чтение 8 КиБ со смещением `491 * 4096` загружает один непрерывный образ метаданных размером 16 КиБ, если обеих пар нет в кеше. К паре, уже принадлежащей другой операции, запрос присоединяется, не получая её повторно. Хранилище результатов callback создаётся до отправки и не перемещается до их завершения.

`CompleteMetadataReads` проверяет и публикует образы, сохраняет результаты затронутых чтений и откладывает уведомления зависимостей до `NotifyCompleted()`. Малое инициирующее чтение может завершиться до уведомления ожидающих его метаданные запросов. Если остаются другие зависимости, а также для больших чтений сначала метаданных, DDisk уведомляет ожидающих завершённые загрузки до следующей приостановки. Активные загрузки и записи удерживают записи кеша. `WaitingWrite` сохраняет предыдущий неизменяемый образ при его наличии; RMW отсутствующих в кеше метаданных не имеет доступного для чтения образа до успеха заключительной записи.

`TWriteOperation` владеет исключительными правами на пары. Записи захватывают несколько пар по возрастанию. Записи в очереди и следующая запись, выбранная для возобновления, удерживают стабильные узлы. Успешное освобождение пробуждает одну следующую запись; отмена освобождает захваченные пары, а заключительная ошибка или остановка завершает все затронутые ожидания. Явное владение заменяет версии изменений и устойчивого сохранения, билеты записи, объединение изменений и корутины сохранения контрольных сумм.

`TMetadataWrite` — владеющий данными контекст операции, подготовленный менеджером. Если все образы есть в кеше, замены строятся без чтения метаданных. Иначе RMW читает одну полную пару или непрерывную область из двух пар, если хотя бы одна отсутствует в кеше: не более четырёх блоков по 4 КиБ. Преобразование проверяет образы и ожидаемые дайджесты, выбирает текущие слоты, применяет новые контрольные суммы и обновляет последовательности и дайджесты. Одна пара записывает заменяющий слот. Две пары записывают непрерывные средние слоты, если замены находятся в них; иначе записываются все четыре блока с сохранением каждого неизменённого текущего образа. Ограничение публичной записи 1 МиБ ограничивает её работу с метаданными двумя парами. Формат диска и правила восстановления не меняются.

Корутина подготавливает загрузку отсутствующих в кеше метаданных с помощью `PrepareMetadataRead` как обычное критическое чтение и ожидает его отдельно. После успешного завершения она преобразует принадлежащий ей контекст `TMetadataWrite` на потоке актора, затем подготавливает и объединяет в пакет записи данных и образа метаданных. Ошибка загрузки, ошибка преобразования, Broken или Stopping предотвращают дальнейшую отправку данных. Успешные метаданные публикуются в кеше на потоке актора после завершения пакета записей.

Резервный путь PDisk использует ту же последовательность корутины через raw I/O на стороне актора. Ошибки данных и критических метаданных сохраняют отдельные результаты. Перегрузки метаданных сохраняют прежние пределы и задержки повторных попыток независимо для загрузки и записи метаданных. Завершения обрабатываются и в Stopping; каждая принятая соседняя операция удерживает буферы и чанки до окончания. `TDataRequestGuard` учитывает клиентские корутины, выделение и форматирование во время отправки, обработки на потоке актора, уведомления и очистки. Освобождение резервов и Gone ожидают эти операции и завершение callback. При отключённых контрольных суммах одна корутина форматирует чанк нулями последовательными порциями.

Размещение нового экстента и его готовность — отдельные этапы. Одна корутина выделения запускает экстент, ожидает размещения, публикует физический чанк данных, ожидает готовности форматирования, отправляет запись отображения в журнал, ожидает её устойчивого сохранения и публикует фиксацию. Готовность требует завершения форматирования экстента и всех трёх заголовков чанка метаданных целостности. Реестр владения выделением хранит токен, физический чанк и признак отправки записи отображения; нулевой физический чанк означает незавершённое резервирование. После приостановок проверяется идентичность токена, а отправленная запись фиксации сохраняет консервативное владение физическим чанком при ошибке и остановке. Удалённые экстенты сохраняют свои слоты до устойчивого сохранения удаления и завершения незаконченного форматирования.

Записи журнала PDisk используют необязательные билеты завершения. Выделение, выделение и освобождение PB, удаление и освобождение неиспользуемых чанков применяют зависящие от завершения результаты после ожидания успеха; построение снимка и карты и помещение в карантин происходят до отправки. Фоновое освобождение может отправить снимок без ожидающего запроса, если результат не зависит от завершения. Все LSN и cookie доставки продолжают отслеживаться, включая записи без ожидающих запросов; фоновые снимки не требуют выделения билета. Обработчик завершения удаляет весь соответствующий пакет из реестра до пробуждения любого ожидающего запроса.

Изменение режима контрольных сумм затрагивает формат и восстановление, а не только производительность. DDisk проверяет совместимость режима контрольных сумм с восстановленным состоянием. У PB есть отдельная настройка контрольных сумм на диске, описанная в Форматы контроля целостности.

Синхронизация

`TEvSync` — единая операция получения и записи данных, используемая как для переноса PB → DDisk, так и для восстановления DDisk → DDisk. Она содержит идентификаторы источников и сегменты; каждый сегмент выбирает запись PB или диапазон исходного DDisk. Получатель отправляет запросы чтения этим сервисам, проверяет возвращённые данные и записывает целевой диапазон.

Все целевые сегменты запроса должны принадлежать одному виртуальному чанку. Обработчик проверяет запрос целиком, включая непустые выровненные диапазоны, ровно один тип каждого сегмента и границы чанка, до начала работы. Действуют гарантии вызывающей стороны об одном активном Sync на набор блоков, непересекающихся диапазонах и исключении обычных записей. Статус `OUTDATED` остаётся в протоколе для ответов сервиса-источника; DDisk считает такой ответ ошибкой входного сегмента.

Обработчик освобождает входное событие и запускает `ExecuteDataWrite`, которая выполняет один цикл корутины. Цикл обрабатывает исходные сегменты в порядке запроса и выровненные порции не более 512 КиБ по возрастанию смещения. Для каждой порции он ожидает один ответ источника, проверяет статус, точный размер полезной нагрузки, количество контрольных сумм и настроенную проверку их хешей, при необходимости выделяет ресурсы и выполняет целевую запись. При включённых контрольных суммах допуск целевой операции следует за получением пар и повторной проверкой сессии; загрузка и преобразование отсутствующих в кеше метаданных предшествуют пакету записей данных и метаданных. Отсутствующий целевой чанк выделяется только после допустимого ответа источника. Следующее чтение источника начинается только после завершения текущих целевых операций. DDisk и PB поддерживают чтения поддиапазонов. Планировщика слотов, очереди допуска, предварительной загрузки метаданных и подписок слотов нет; кооперативная уступка выполнения ограничивает непрерывную синхронную работу.

Каждый запрос источнику получает свежий cookie, определяющий соответствующий Sync и ожидаемый тип события. Обработчики источников сохраняют результат и пробуждают корутину верхнего уровня, которая принимает решения. Устаревшие ответы игнорируются, а ответ другого типа сохраняет действующий маршрут. Принятые локальные callback удерживают общее хранилище пакета при принудительном уничтожении кадра.

Ответ содержит по одному результату на исходный сегмент в порядке запроса. Ошибка порции пропускает остаток этого сегмента, сохраняет завершённые записи, дожидается принятых локальных ветвей и переходит к следующим сегментам, пока актор исправен. Результаты данных, метаданных и сессии сохраняют приоритет ошибок, в том числе приоритет повреждения метаданных над ошибкой сессии для той же порции. Broken или Stopping прекращает ожидание ответов источника, запрещает новую целевую работу и ожидает принятую локальную работу. Уже принятая нативная порция может успешно завершиться в Stopping после окончания обеих ветвей и фиксации отображения. Перед ответом Sync также ожидает окончательной фиксации отображения, в том числе при ошибке на фоне уже начавшегося независимого выделения.

Операция возвращает `TEvSyncResult` после обработки работы на целевом DDisk. Она не удаляет исходные записи PB. Клиент решает, когда данные перенесены в достаточное число реплик, требуется ли восстановление и когда безопасно удалять данные PB. Источник PB может быть удалённым, даже если он занимает тот же логический индекс в DBG, что и целевой DDisk.

Сбои и восстановление

Снимки карты чанков и приращения в журнале PDisk восстанавливают владение ресурсами и отображения экстентов метаданных целостности. После успешного полного воспроизведения журнала DDisk вычисляет кандидатов на освобождение

как `OwnedChunksOnBoot` за вычетом объединения восстановленных чанков данных, перечисленных чанков метаданных целостности, всех чанков метаданных целостности, на которые ссылаются восстановленные экстенты, и чанков PB. Ссылки экстентов защищают чанки, даже если их нет в списке чанков метаданных целостности. DDisk вычисляет это объединение до того, как освобождение ресурсов при запуске изменит отображения, поэтому защищены ссылки как из снимков, так и из более поздних приращений журнала. Такая консервативная защита не делает некорректное отображение допустимым: существующие проверки восстановления сохраняются. При неудачном или неполном восстановлении сверка не выполняется.

DDisk отправляет по одному чанку без восстановленных ссылок в `TEvChunkForget`, отслеживает доставку и ожидает ответа, прежде чем продолжить. Создание PB, новые выделения чанков и готовность обслуживать клиентов заблокированы до завершения сверки. DDisk явно устанавливает `IsDDisk = true` в запросах резервирования и освобождения; для остальных вызывающих сторон значение поля по умолчанию — `false`. Для помеченных запросов освобождения PDisk сохраняет cookie запроса во всех ответах и проверяет владельца и его раунд при выполнении запроса, поэтому запоздалый запрос старого экземпляра не может освободить повторно используемые чанки. Существующее поведение VDisk, включая ответы об ошибках, обработку cookie и уровни диагностических сообщений, не меняется.

Как успешное освобождение, так и отказ при проверке чанка позволяют продолжить сверку. DDisk сохраняет отклонённые чанки и записывает их идентификаторы и причины отказа на уровне WARN. Для помеченных запросов PDisk также записывает BPD91 на уровне WARN при ожидаемых переходных состояниях: `DATA_ON_QUARANTINE`, `DATA_RESERVED_DELETE_IN_PROGRESS`, `DATA_COMMITTED_DELETE_IN_PROGRESS`, `DATA_RESERVED_DELETE_ON_QUARANTINE` и `DATA_COMMITTED_DELETE_ON_QUARANTINE`. Эти чанки освобождаются существующими механизмами завершения ввода-вывода и записи журнала PDisk; сверка не добавляет повторных попыток или принудительного удаления. Отказ для зафиксированного чанка без восстановленных ссылок (`DATA_COMMITTED`) остаётся ошибкой уровня ERROR в PDisk, а остальные неожиданные ошибки проверки сохраняют прежние уровни сообщений. Ошибка сессии или устройства PDisk, а также недоставка запроса переводят актор в Stopping. Poison отменяет оставшуюся сверку, и запоздалый ответ не может возобновить запуск. PDisk возвращает заключительные ответы `CORRUPTED`, если остановка отменяет помеченные запросы резервирования или освобождения в очереди, в том числе в отдельной очереди освобождения. Непомеченные запросы в очередях по-прежнему удаляются без ответа.

Затем PB самостоятельно сканирует чанки и восстанавливает записи. После замены сервиса клиенты должны заново установить соединения.

Остановка и перезапуск

DDisk и PB всегда обязаны дождаться завершения принятого роутером асинхронного ввода-вывода и обратных вызовов перед подтверждением остановки. Ни один из акторов не может отправить Gone, пока эти вызовы ещё владеют его состоянием. На резервном пути PDisk актор отменяет принадлежащие ему запросы и обрабатывает их заключительные результаты до Gone; описанный ниже барьер остановки PDisk дожидается завершения ввода-вывода на самом устройстве.

Потеря сессии PDisk переводит актор в то же идемпотентное состояние Stopping, что и poison, но только poison разрешает уничтожение актора и подтверждение остановки. В состоянии Stopping актор отклоняет новые запросы с `SESSION_MISMATCH`, отменяет принадлежащие ему операции резервного пути PDisk и отложенные повторные попытки, но продолжает обрабатывать результаты уже отправленного ввода-вывода. Отмена балансирует счётчики и публикует результаты до заключительного прохода почтового ящика. Завершение записи по-прежнему требует надёжного сохранения метаданных целостности и записей журнала выделения чанков; новые операции ввода-вывода при остановке не запускаются.

Состояния Broken и Stopping пробуждают логические ожидания, которые не могут продолжиться, но не отменяют ожидание принятого роутером ввода-вывода. Запрос с ошибкой перед ответом ожидает завершения остальных уже отправленных операций; их буферы и физические закрепления чанков сохраняются до обработки заключительных результатов. Как логические операции, так и незавершённые физические операции защищают чанки от удаления. Оставшиеся логические ожидания и ответы на запросы завершаются до заключительного прохода почтового ящика и Gone. Завершения пакетов публикуются как в рабочем состоянии, так и в Stopping. `TDataRequestGuard` учитывает клиентские корутины, корутины выделения и форматирования в одном счётчике `DataRequestsInFlight` во время отправки, обработки результатов, уведомлений и очистки. Освобождение резервов и завершение остановки ожидают нулевого значения этого счётчика и завершения обратных вызовов, в том числе когда callback уже завершены, а результат пакета ещё находится в очереди. При принудительном уничтожении обратные вызовы завершаются до освобождения оставшихся закреплений реестра.

Доставка запросов `TEvLog` для карты чанков отслеживается; уведомление о недоставке, сопоставленное по cookie с незавершённым запросом, переводит актор в Stopping.

После завершения собственного ввода-вывода и обработки заключительных результатов DDisk запрашивает освобождение известных ему резервов через `TEvChunkForget`, если инициализация PDisk и воспроизведение журнала завершены. В запрос попадают неиспользованные зарезервированные чанки, чанки прерванного форматирования и выделения данных, а также чанки метаданных целостности, но только если запись журнала для их фиксации ещё не отправлялась. Отправленная запись фиксации исключает чанк из освобождения, даже если подтверждение ещё не получено; чанки, выделенные PB, также исключены. Запрос содержит текущие идентификатор владельца PDisk и раунд владельца. Принятый роутером ввод-вывод должен завершиться до освобождения; чанки с незавершённым вводом-выводом резервного пути PDisk помещает в карантин до его завершения.

В состоянии Broken чанки прерванного форматирования и выделения данных без отправленной записи журнала хранятся отдельно от неиспользованных резервов. Это не позволяет PB повторно использовать чанк, пока старая запись DDisk ещё может обращаться к нему. Свежие резервы, для которых DDisk не запускал ввод-вывод, остаются доступны PB. Успешные ответы на резервирование, полученные в состоянии Stopping, сохраняются без запуска выделения или форматирования. После завершения собственного ввода-вывода запоздалые ответы могут вызвать дополнительные запросы освобождения, пока актор ещё жив. Каждый чанк передаётся на освобождение не более одного раза за время жизни экземпляра актора, поэтому последующие запросы содержат только новые идентификаторы независимо от ответов на предыдущие запросы.

Незавершённый запрос резервирования (`ChunkManager.IsReservationInFlight()`) не позволяет DDisk отправить Gone, даже если собственный ввод-вывод и остановка PB уже завершены. Любой заключительный ответ на резервирование снимает этот барьер; чанки из успешного ответа добавляются в множество освобождения и освобождаются только после существующего барьера ввода-вывода. Доставка запроса резервирования отслеживается: недоставка снимает ожидание запроса и переводит актор в Stopping. PDisk возвращает заключительный ответ `CORRUPTED`, если остановка отменяет запрос резервирования в очереди с установленным флагом `IsDDisk`. Тайм-аут, позволяющий забыть о незавершённом резервировании, отсутствует.

Остановка не ожидает ответов на освобождение и не повторяет эти запросы. Подтверждения остаются негарантированными: потерянные или отклонённые запросы освобождения могут оставить зарезервированные чанки в работающем PDisk после перезапуска только DDisk. Сверка при запуске освобождает резервы прежних экземпляров, на которые нет восстановленных ссылок, с учётом описанных выше проверок и обработки ошибок. Перезапуск PDisk снимает резервы чанков, которые никогда не были зафиксированы. Очистка не требует новых типов событий, формата хранения или схемы журнала и не меняет протокол удаления зафиксированных чанков.

DDisk и PB независимо ожидают завершения собственного ввода-вывода. PB освобождает ссылку на роутер до отправки `TEvGone` конкретному родительскому актору. Родитель ожидает завершения собственного ввода-вывода, `TEvGone`

конкретного дочернего актора и завершения запроса резервирования, освобождает свою ссылку на роутер и затем уведомляет Warden. Отслеживаемая доставка poison дочернему актору обрабатывает случай уже отсутствующего PB; повторные poison и уведомления безопасны. Через 60 секунд каждый актор с незавершёнными обратными вызовами увеличивает `ddisks/io_stalled` на единицу и уменьшает его после завершения собственных обратных вызовов. Диагностика остановки также отражает ожидание PB и незавершённый запрос резервирования. При штатной остановке время ожидания зависшего ввода-вывода или незавершённого резервирования не ограничено.

Число незавершённых обратных вызовов и признак остановки хранятся в одном атомарном состоянии. Очистка ресурсов и публикация результата предшествуют освобождению владения обратного вызова; события завершения и отмены повторных попыток обрабатываются до уничтожения актора. Принудительная очистка почтового ящика уничтожает кадры корутин до вызова деструктора актора. При этом очищаются активные продолжения ожиданий пакетов и освобождаются принадлежащие кадрам разделяемые указатели; незавершённые операции сохраняют каждый `TBatchedIOAwaiter`, поэтому запоздалые callback могут безопасно записывать результаты, пока деструктор ожидает их завершения. Деструктор сохраняет все поля актора, ожидая до 10 секунд по монотонным часам, после чего аварийно завершает процесс, если обратные вызовы всё ещё владеют его состоянием. Этот срок не связан с 60-секундной диагностикой зависшего ввода-вывода и не ограничивает ожидание при штатной остановке.

Ошибки перегрузки критических операций контроля целостности и форматирования допускают 20 повторных отправок с задержкой `min(1 мс × 2^(retry−1), 100 мс)`. Немедленное событие обратного вызова передаёт операцию в принадлежащую актору карту; таймеры содержат только идентификаторы. Broken/Stopping отменяет отложенные операции, а устаревшие таймеры ничего не делают. Исчерпание попыток возвращает `ERROR` с числом попыток и последним errno; преобразование ошибок обычного ввода-вывода не меняется.

Восстановление PB обрабатывает данные только после успешного чтения. Первая ошибка чтения переводит актор в Broken и завершает ожидающие запросы с `ERROR`; запоздалые результаты лишь завершают учёт операций и не могут разбирать данные, возобновлять восстановление или объявлять готовность. Множество восстановления по-прежнему содержит уже запланированные чанки.

При перезапуске PDisk по запросу NodeWarden сначала запрашивает остановку всех связанных экземпляров акторов DDisk. Каждый DDisk запрашивает остановку своего PB и ожидает завершения собственного ввода-вывода, `TEvGone` от конкретного дочернего актора и завершения незаконченного запроса резервирования. Только после Gone последнего DDisk NodeWarden отправляет разрешение на перезапуск PDisk. Запуск новых DDisk/PB заблокирован как во время ожидания этих акторов, так и до завершения перезапуска PDisk. Завершение ввода-вывода PB и его Gone, завершение ввода-вывода DDisk и завершение резервирования должны предшествовать Gone DDisk, после которого выдаётся разрешение на перезапуск PDisk. Ввод-вывод двух акторов и резервирование могут завершиться в любом порядке.

Новый PDisk не может начать ввод-вывод на устройстве, пока не завершён ввод-вывод предыдущего PDisk. `TPDisk::Stop()` всегда вызывает `StopSync()` общего роутера, даже если DDisk/PB или сохранённые результаты инициализации ещё содержат ссылки на клиенты роутера. Этот вызов закрывает приём операций, ожидает завершения публикации операций и заключительных обратных вызовов, присоединяет поток отправки ввода-вывода, освобождает кольцо и закрывает дублированный дескриптор устройства роутера. Затем PDisk останавливает исходное блочное устройство до начальной загрузки нового экземпляра. Старый клиент может пережить этот барьер, но отклоняет новые операции и больше не владеет активным кольцом или дескриптором устройства. При исправном кольце время ожидания не ограничено; если при сломанном кольце невозможно дождаться освобождения ресурсов ядром, процесс аварийно завершается, не допуская запуска нового PDisk при незавершённом старом вводе-выводе.

Тот же синхронный барьер ввода-вывода действует при остановке или перезапуске PDisk независимо от последовательности, запрошенной NodeWarden. DDisk и PB узнают об остановленном роутере при отклонении отправляемой операции и переходят в Stopping; потеря сессии PDisk также переводит их в Stopping. Роутер не отправляет отдельного уведомления бездействующему актору. PDisk ожидает завершения принятых роутером операций и их обратных вызовов, что сохраняет состояние акторов на время выполнения этих вызовов. Этот барьер не ожидает заключительной обработки почтовых ящиков акторов или их уведомлений Gone. Акторы по-прежнему обрабатывают оставшиеся результаты, а для их уничтожения и уведомления Warden требуется poison. Этот независимый барьер ввода-вывода не следует отождествлять с дополнительным ожиданием Gone акторов при запрошенном перезапуске.

`TEvDeleteTabletChunks` удаляет отображения чанков данных таблетки. Пока удаление выполняется, записи и запросы синхронизации этой таблетки могут возвращать `BUSY`. Снятие закреплений в контроллере и удаление локальных чанков — отдельные операции; вызывающая сторона должна согласовать их порядок и завершить незаконченные клиентские операции.

Исходный код и тесты

Область	Исходный код	Целевые тесты
Состояние актора и сессии	ddisk_actor.cpp, ddisk_actor_connect.cpp	`ut/ddisk_actor_ut.cpp`
Запуск, журнал и чанки	ddisk_actor_boot.cpp, ddisk_actor_chunks.cpp	`ut/ddisk_actor_pdisk_ut.cpp`
Учёт резервов и порядок выделения чанков	chunk_manager.h	`ut/chunk_manager_ut.cpp`
Чтение/запись и адаптеры ввода-вывода	ddisk_actor_read_write.cpp, direct_io_op.cpp	`ut/ddisk_actor_checksum_ut.cpp`, `ut/ddisk_actor_ut.cpp`
Состояние контроля целостности	integrity_manager.cpp	`ut/integrity_manager_ut.cpp`
Синхронизация	ddisk_actor_sync.cpp	`ut/ddisk_sync_ut.cpp`, `ut/ddisk_actor_ut.cpp`

Пути в столбце тестов указаны относительно `ydb/core/blobstorage/ddisk`. `ut_large` содержит более длительные сценарии ввода-вывода и синхронизации с PDisk. Для локального изменения выбирайте соответствующую цель и тесты, а не всё дерево тестов распределённого хранилища.

См. также

- Распределённое хранилище YDB
- Группы прямого блочного доступа
- PersistentBuffer
- Навигация по исходному коду DDisk

PersistentBuffer

PersistentBuffer (PB) надёжно хранит промежуточные записи для клиентов прямого блочного доступа. Клиент может прочитать запись, реплицировать её в другие PB, попросить DDisk перенести её в конечное блочное хранилище и удалить, когда это разрешает его политика надёжности. PB реализует эти примитивы; политику группы задаёт партиция или нагрузочный актор.

PB — дочерний актор DDisk, реализованный классом `TDDiskActor` с отдельной функцией состояния. К его сервисному идентификатору можно обращаться независимо. Чанки хранения и владелец PDisk управляются через родителя; см. NodeWarden.

Идентификатор записи

Запись PB определяется кортежем `(TabletId, Generation, Lsn, DirectBlockGroupIndex)`. Её метаданные содержат целевой виртуальный чанк, смещение в байтах, размер, расположение секторов и необязательные контрольные суммы полезной нагрузки. `DirectBlockGroupIndex` хранится как `ui8` и должен находиться в диапазоне `0..255`; если индекс не указан, используется пространство имён ноль.

Порядковый номер записи журнала (log sequence number, LSN) назначает клиент. Это идентификатор записи в пространстве имён, а не физическое смещение в PB или LSN журнала PDisk. Повторная идентичная запись с тем же ключом идемпотентна и возвращает `OK`. Несовпадающий селектор приводит к `INCORRECT_REQUEST`; несовпадающие данные также приводят к `INCORRECT_REQUEST`, если сохранённые контрольные суммы позволяют PB обнаружить различие. Записи на границе сохранённого барьера удаления или до неё возвращают `OUTDATED`. Клиент не должен использовать один LSN для разных записей в расчёте на перезапись.

> **Примечание**
>
> Обнаружение конфликтов полезной нагрузки в повторных записях находится в разработке. Для записи, созданной в дисковом формате без контрольных сумм и не содержащей контрольных сумм полезной нагрузки, PB сейчас сравнивает только метаданные селектора (`VChunkIndex`, `OffsetInBytes` и `Size`). Поэтому запись с тем же ключом и селектором, но с другими байтами полезной нагрузки, может быть принята как повторная и вернуть `OK`. В задачах, которые используют или изменяют это поведение, сначала проверьте актуальную реализацию `PreprocessPersistentBufferWrite` и соответствующие целевые тесты.

Пространство имён PB включает поколение, а барьер удаления охватывает поколения одной пары `(TabletId, DirectBlockGroupIndex)`. Клиент, восстанавливающий старое поколение, может читать или удалять его записи, используя поле поколения записи в операции, если оно предусмотрено, и предъявляя реквизиты своей текущей сессии.

Регистрация и прекращение обслуживания

Перед операциями с данными клиент подключается к PB и запрашивает одноразовый токен через `TEvGetPersistentBufferRegistrationToken`. Токен привязан к `(TabletId, Generation, DirectBlockGroupIndex)` и передаётся в поле `Token` сообщения `TEvRegisterPersistentBuffer`. Это отдельный токен, отличный от токена соединения. `TPBufferConfig.RegistrationTimeoutMilliseconds` по умолчанию равен 5000; NodeWarden передаёт его в `TPersistentBufferFormat::RegistrationTimeoutMilliseconds`. Срок действия токена измеряется локальными монотонными часами PB и не зависит от синхронизации часов между узлами. Токены хранятся в памяти и становятся недействительными при перезапуске актора PB. Число выданных и ещё действующих токенов ограничено `TPersistentBufferFormat::MaxRegistrationTokens` (по умолчанию 1024); при достижении лимита запрос токена возвращает `OVERLOADED`.

Регистрация возвращает `OUTDATED` для неизвестного, просроченного или уже использованного токена и `INCORRECT_REQUEST`, если токен принадлежит другой таблетке, поколению или DBG. Пока запрос регистрации находится в очереди, токен не расходуется; при готовности PB его срок действия проверяется повторно. Переполнение очереди также не расходует токен. Когда готовый PB обрабатывает регистрацию, он расходует токен, даже если существующий барьер или нехватка места препятствуют регистрации. Повторные запросы должны использовать тот же токен без продления срока его действия. Успешный ответ отправляется после сохранения на диске барьера с нулевыми поколением и LSN. Существующий барьер запрещает регистрацию, даже если живых записей нет. При повторном подключении клиент может проверить через `TEvListPersistentBuffer`, что существующая регистрация по-прежнему обслуживается. Запись, чтение, получение списка и удаление записей требуют зарегистрированной пары `(TabletId, DirectBlockGroupIndex)`; управление соединениями и диагностика доступны и до регистрации.

`TEvUnregisterPersistentBuffer` окончательно прекращает обслуживание регистрации. PB перестаёт принимать его операции, дожидается завершения текущих операций и записывает барьер с максимальными поколением и LSN. Затем он ждёт удвоенный тайм-аут регистрации (по умолчанию 10 секунд), сохраняет удаление барьерной записи на диске и отвечает `OK`. На протяжении прекращения обслуживания регистрация отклоняется. Благодаря задержке токены, выданные до начала прекращения обслуживания, истекают до удаления барьера, поэтому сообщения регистрации с этими токенами не могут воссоздать регистрацию. Если при восстановлении обнаружен максимальный барьер, прекращение обслуживания возобновляется с полным интервалом ожидания. После ошибки записи барьера PB может возобновить обычные операции только после восстановления.

Транспорт прямой партиции NBS завершает подключение к PB только после `TEvConnect`, получения токена, регистрации и успешной проверки через получение списка. Эта проверка также подтверждает обслуживание существующей регистрации после отказа в повторной регистрации при переподключении. При удалении партиции обслуживание каждой регистрации таблетки/DBG прекращается до удаления чанков DDisk и освобождения групп в BSC, в том числе для отдельных регистраций на общем PB. Уже отсутствующая регистрация считается очищенной.

Запись, чтение и репликация

`TEvWritePersistentBuffer` записывает одну запись в один PB. Вспомогательные функции полезной нагрузки из ddisk.h формируют ссылки на полезную нагрузку события и поля контрольных сумм. Если проверка контрольных сумм включена, PB отклоняет не соответствующую им полезную нагрузку до выделения ресурсов или ввода-вывода.

Текущий формат записи допускает не более 128 секторов данных, то есть 512 КиБ при секторе 4 КиБ. Небольшие записи размером до восьми секторов могут объединяться под одним сектором заголовка, если хватает места для метаданных. Подтверждение отправляется после соответствующей дисковой операции, а не сразу после добавления в карту в памяти. Данные записи могут оставаться в ограниченном кеше памяти; при промахе кеша чтение выполняет ввод-вывод по сохранённым адресам секторов и восстанавливает исходную полезную нагрузку. `TEvReadPersistentBuffer` может вернуть запись целиком или запрошенную часть с соответствующими контрольными суммами.

`TEvWritePersistentBuffers` делегирует рассылку координатору PB. Его `TWritePersistentBuffersRequestActor` отправляет запрос на запись каждому явно указанному получателю и собирает их статусы. За кворум и повторные попытки по-прежнему

отвечает вызывающая сторона.

Тайм-аут ответа задаёт срок сбора результатов, а не отмену. В этот момент координатор сообщает о получателях, которые уже ответили; их может не быть вовсе. Незавершённые операции продолжают отслеживаться; после получения их результатов следующий ответ может содержать дополнительные результаты получателей. Клиенты должны объединять результаты по получателю и идентификатору запроса и не предполагать, что на один запрос записи в несколько PB придёт ровно один ответ.

`TEvReadThenWritePersistentBuffers` сначала читает запись из PB координатора, затем отправляет её полезную нагрузку и сохранённые контрольные суммы списку получателей. Это позволяет клиенту управлять повторной репликацией, не пропуская данные через себя.

Для переноса PB → DDisk отправьте целевому DDisk `TEvSync` с сегментами источника PB. Чтение PB входит в операцию синхронизации этого DDisk; ни подтверждение записи PB, ни результат синхронизации автоматически не продвигают барьер удаления.

Удаление и барьеры

Механизм	Контракт и реализация
`TEvBatchErasePersistentBuffer`	Удаляет явно указанные записи `(Generation, Lsn)` в пространстве имён DBG вызывающей стороны. Если ограничения поколения и ёмкости позволяют, может использовать компактную запись быстрого удаления; иначе выполняется инвалидирование отдельных записей.
Быстрое удаление	Сохраняет компактное множество LSN, затем освобождает выбранные записи. Это позволяет обойтись без записи сектора данных для каждой удаляемой записи, но требует места для собственных метаданных.
Инвалидирование отдельной записи	Делает недействительным первый сектор данных каждой записи. Общий заголовок пакета может использоваться живыми записями, поэтому удаление одной записи не должно делать недействительным этот общий заголовок.
`TEvErasePersistentBuffer`	Продвигает сохраняемый барьер отсечения для `(TabletId, DirectBlockGroupIndex)`. Барьер задаёт границу в порядке `(Generation, Lsn)`: старые поколения и записи до границы включительно в поколении барьера становятся устаревшими.

Обновления барьера и быстрого удаления записывают новые метаданные до освобождения старых метаданных и секторов данных. `MinFreeSectorsReserve` сохраняет место для этого продвижения; допуск записей до заполнения всех секторов может заблокировать само освобождение ресурсов.

Удаление — часть сохраняемого состояния, в том числе для записей, которых уже нет в карте в памяти. Восстановление должно применить барьеры и быстрые удаления до публикации множества живых записей, чтобы старое содержимое диска не появилось снова. Точечные удаления могут быть идемпотентными, если указанные записи уже отсутствуют.

Клиент решает, какие записи безопасно удалять. Например, партиция может ожидать все необходимые реплики данных и замещающие копии PB; эти условия находятся за пределами общего обработчика PB. Барьер отсечения особенно значим, поскольку удаляет префикс и старые поколения: его нельзя определять по произвольной завершённой записи, если более ранние записи ещё нужны.

Форматы контроля целостности

Есть две связанные настройки с разными обязанностями:

- `TDDiskConfig::EnableChecksums` управляет проверкой и передачей контрольных сумм полезной нагрузки через интерфейс DDisk/PB. В сетевом протоколе используются чистые значения XXH3-64 для блоков размером 4 КиБ.
- `TPersistentBufferFormat::EnableChecksums` управляет форматом контроля целостности PB на диске. Записи с контрольными суммами хранят сведения о целостности секторов и заголовка. Формат без контрольных сумм сохраняет уникальный идентификатор заголовка записи в секторах данных, а исходные байты их начала — в метаданных.

Флаги заголовка на диске определяют формат каждой записи, поэтому восстановление и чтение интерпретируют существующую запись по её сохранённому формату, а не только по текущей настройке. Контрольные суммы полезной нагрузки от отправителя — это дополнительный необязательный массив метаданных, отмеченный `HAS_PAYLOAD_CHECKSUMS`; их не следует путать с контрольными суммами секторов PB или отдельными чанками метаданных целостности DDisk.

Структура заголовка, коррекция сигнатуры, длина содержимого, покрываемого контрольной суммой, и вместимость упакованных записей определены в persistent_buffer_header.h. Изменение формата должно сохранять возможность при восстановлении различать допустимые заголовки, секторы данных, барьеры и записи удаления.

Восстановление, получение списка и ограничения

Родительский DDisk восстанавливает владение чанками PB и уникальный идентификатор PB до создания дочернего актора. PB сканирует принадлежащие ему чанки, проверяет заголовки и ссылки на данные, применяет барьеры и удаления, восстанавливает метаданные выделения ресурсов и публикует восстановленные записи. Запросы, которым нужно восстановленное состояние, могут ждать в ограниченной очереди; её переполнение возвращает результат перегрузки.

`TEvListPersistentBuffer` предоставляет клиенту представление записей PB для восстановления. Получение списка откладывается, пока для таблетки выполняются соответствующие дисковые операции, чтобы не показывать частично применённую запись или удаление. `ListPersistentBufferMaxRetries` и `ListPersistentBufferRetryPeriodMilliseconds` ограничивают ожидание; при исчерпании попыток возвращается результат перегрузки, а не достоверный пустой список записей.

Ограничения PB относятся к выделенным чанкам, объёму хранения для таблетки, памяти кеша, ожидающим запросам и резерву для удаления. Пороги упреждающего выделения и освобождения управляют увеличением числа чанков и их возвратом. Эти ограничения отличаются от счётчика ссылок на PB в BSC, который описывает совместное использование и размещение, а не объём живых записей в байтах.

`TEvGetPersistentBufferInfo` и страница мониторинга показывают локальные записи, барьеры, ёмкость, кеш и статистику операций. Для восстановления состояния логической группы клиенту всё равно нужно объединить сведения от всех соответствующих участников.

Исходный код и тесты

- persistent_buffer.h: ключи пространств имён, метаданные записей и кешированные данные.

- ddisk_actor_persistent_buffer.cpp: запись/чтение, пакетирование, восстановление, получение списка и выполнение удаления.
- persistent_buffer_barriers_manager.cpp: барьеры отсечения и состояние компактного удаления.
- persistent_buffer_space_allocator.cpp: владение секторами и их выделение.
- write_persistent_buffers_request_actor.cpp: рассылка, частичные ответы и повторная репликация.
- persistent_buffer_mon.cpp: мониторинг.

Целевые тесты находятся в `ydb/core/blobstorage/ddisk/ut`: `ddisk_actor_ut.cpp` проверяет сессии, разделение пространств имён, получение списка, запоздалые ответы координатора и жизненный цикл PB; `ddisk_actor_batch_write_ut.cpp` — упакованные записи; `ddisk_actor_checksum_ut.cpp` — целостность полезной нагрузки. У аллокатора и менеджера барьеров есть отдельные файлы модульных тестов. Сценарии восстановления с PDisk находятся в `ddisk_actor_pdisk_ut.cpp` и `../ut_large`.

См. также

- DDisk
- Группы прямого блочного доступа
- NbsDbgLikeLoad

Отчёт об использовании дискового пространства VDisk

Отчёт об использовании дискового пространства VDisk оценивает, как место, выделенное одному VDisk, распределено между данными, метаданными, мусором, фрагментацией и вспомогательными подсистемами. Чанк представляет собой фиксированную единицу выделения места на PDisk.

Отчёт доступен на странице мониторинга VDisk, через внутренний акторный API и в виде динамических счётчиков.

> **Важно**
>
> Отчёт не является консистентным снимком. Источники статистики и индексы читаются в разные моменты и могут описывать разные состояния VDisk. Рассматривайте каждое значение как мониторинговую оценку: не ожидайте совпадения сумм из разных разделов или повторных запросов. Ненулевой `ReconciliationDeltaBytes` и неклассифицированные байты являются допустимым результатом. Не используйте отчёт для принятия решений, влияющих на корректность, размещение данных или выполнение операций.

Рекомендации по эксплуатации

Возврат отчёта из кеша не требует значительных ресурсов, но пересчёт обходит индексы VDisk. Чтобы снизить влияние на пользовательскую нагрузку, не запускайте принудительный пересчёт одновременно более чем для одного VDisk в группе или десяти VDisk на одном узле. Если отчёты нужны регулярно, предпочитайте кешированные отчёты или настройте периодический сбор.

На одном VDisk одновременно выполняется только один пересчёт. Параллельные принудительные запросы присоединяются к активному пересчёту и получают одинаковый результат.

Страница мониторинга

Добавьте к URL страницы мониторинга нужного VDisk одну из следующих строк запроса:

- `?type=spacereportvisual` показывает кешированный отчёт в виде таблиц и диаграмм распределения места;
- `?type=spacereport` возвращает текстовое представление protobuf;
- добавьте `&force=1` к любому варианту, чтобы дождаться нового отчёта, например `?type=spacereportvisual&force=1`.

На странице визуализации также есть ссылки **Raw Proto** и **Recalculate**. Тайм-аут запроса мониторинга равен одной минуте. Если HTTP-клиент отключился или запрос мониторинга завершился по тайм-ауту, пересчёт на VDisk продолжается.

Без `force=1` страница читает кеш и не запускает пересчёт. До первого успешного сбора ответ имеет статус `NOTREADY` и не содержит отчёта.

Сбор и кеширование

Параметр immediate control `VDiskControls.SpaceReportPeriodSeconds` задаёт интервал периодического сбора в секундах. Допустимый диапазон — от 0 до 86400, значение по умолчанию — 0. Нулевое значение отключает периодический сбор, но отчёт по-прежнему можно получить принудительным запросом. Момент первого сбора случайным образом распределён по заданному интервалу. Последующие интервалы также варьируются, чтобы все VDisk на узле не запускали сканирование одновременно.

Успешный сбор заменяет отчёт в кеше и публикует его счётчики. При ошибке предыдущий успешный отчёт и его счётчики данных не изменяются. Операционные счётчики при этом фиксируют неудачную попытку.

Акторный API

Запрос и ответ объявлены в vdisk_events.h:

- `TEvGetVDiskSpaceReportRequest` содержит protobuf-сообщение `TGetVDiskSpaceReportRequest`;
- `TEvGetVDiskSpaceReportResponse` содержит protobuf-сообщение `TGetVDiskSpaceReportResponse`;
- схема сообщений находится в space_report.proto.

Запрос отправляется непосредственно на сервисный ActorId нужного VDisk.

`TGetVDiskSpaceReportRequest` имеет одно поле:

Поле	Значение
`ForceRecalculation`	Если `false`, немедленно вернуть последний кешированный отчёт или `NOTREADY`, если кеш пуст. Сбор при этом не запускается. Если `true`, запустить сбор или присоединиться к уже выполняющемуся и ответить после завершения попытки.

`TGetVDiskSpaceReportResponse` содержит:

Поле	Значение
`Status`	Строковое имя значения `NKikimrProto::EReplyStatus`.
`Report`	Необязательный структурированный отчёт. Проверяйте его наличие независимо от `Status`.
`ErrorReason`	Диагностический текст. Его формулировка не является частью контракта API.

Успешный ответ имеет статус `OK` и содержит отчёт. Если вспомогательный источник статистики завершился с ошибкой или по тайм-ауту, принудительный запрос может получить `ERROR` с частичным отчётом. Если недоступна обязательная статистика PDisk, ответ не содержит отчёта. Неудачные и частичные результаты не кешируются.

Вызывающая сторона должна сама задать срок ожидания, так как акторный запрос не имеет протокола отмены. Сбор прерывается, если длится более 30 минут. После перезапуска VDisk состояние выполнявшегося запроса не восстанавливается. Повторите запрос, когда VDisk снова будет готов.

Структура отчёта

`TVDiskSpaceReport` содержит общий баланс и детализацию по подсистемам. Все размеры измеряются в байтах.

Общий баланс

Поле	Значение
`ChunkSizeBytes`	Размер одного чанка текущего PDisk.
`PDiskAllocatedChunks`	Число чанков, выделенных владельцу VDisk по данным PDisk.
`PDiskAllocatedBytes`	Произведение `PDiskAllocatedChunks` и `ChunkSizeBytes`.
`AccountedBytes`	Сумма всех полей `Total`.
`ReconciliationDeltaBytes`	Знаковая разность `PDiskAllocatedBytes` и `AccountedBytes`. Положительное значение означает неучтённое место, а отрицательное — избыточно учтённое. Исходные данные снимаются раздельно, поэтому ненулевое значение ожидаемо, а нулевое не делает отчёт консистентным снимком.
`Total`	Суммарная классификация байтов всех компонентов отчёта.
`CollectionStartedAtUnixMs`	Время начала сбора в миллисекундах Unix time.
`CollectionCompletedAtUnixMs`	Время завершения сбора в миллисекундах Unix time. Используйте его для определения возраста кеша.

Для интерпретации результата используйте следующие соотношения:

```
PDiskAllocatedBytes = PDiskAllocatedChunks * ChunkSizeBytes
AccountedBytes = сумма полей Total
ReconciliationDeltaBytes = PDiskAllocatedBytes - AccountedBytes
AllocatedBytes компонента = ChunkCount компонента * ChunkSizeBytes + StripedBytes компонента
```

Компоненты

Каждый обычный компонент имеет поля `ChunkCount`, `StripedBytes`, `AllocatedBytes` и `Breakdown`. В `ChunkCount` входят только чанки, выделенные данному компоненту. `StripedBytes` содержит экстенты компонента в общих stripe-чанках.

В таблице ниже LogoBlob означает запись блоба в Hull, SST — неизменяемый отсортированный сегмент индекса, `Huge` — аллокатор крупных блобов, `SyncLog` — журнал синхронизации, а `ChunkKeeper` владеет чанками вспомогательных подсистем.

Компонент	Что учитывается
`LogoBlobs`	SST и индексы LogoBlob, а также данные блобов, размещённые в Hull. Экстенты Huge учитываются отдельно в `Huge`.
`Blocks`	SST и индексы записей блокировки поколений таблеток.
`Barriers`	SST и индексы барьеров сборки мусора.
`Huge`	Выделенные чанки Huge-аллокатора, его экстенты в общих stripe-чанках, свободный резерв и статистика слотов по классам размера.
`SyncLog`	Активные чанки журнала синхронизации.
`ChunkKeeper`	Повторяющиеся записи по идентификаторам подсистем ChunkKeeper. Известно только число зафиксированных чанков, поэтому все их байты относятся к `UnclassifiedBytes`.
`Unattributed`	Оставшиеся чанки PDisk, не сопоставленные ни с одним именованным компонентом. Все эти байты относятся к `UnclassifiedBytes`.
`StripeHeap`	Сводная статистика аллокатора общих stripe-чанков: `ChunkCount`, `AllocatedBytes`, `UsedBytes`, `FreeBytes` и `LockedFreeBytes`. Это не дополнительный компонент: та же ёмкость распределена по `StripedBytes` и `Breakdown` обычных компонентов.

Каждая запись `ChunkKeeper` содержит идентификатор владельца в `SubsystemId` и соответствующий обычный компонент в `Total`.

Категории байтов

`TVDiskSpaceBreakdown` разделяет физический объём по семантическому назначению.

Поле	Значение	Предлагаемая верхнеуровневая категория
`UsefulBlobDataBytes`	Полезная нагрузка выбранного актуального физического представления блоба.	Полезные данные
`LiveMetadataBytes`	Метаданные актуальных записей и структурные метаданные SST.	Метаданные
`LiveAuxiliaryDataBytes`	Актуальные служебные данные вне блобов. Сюда попадают занятые байты SyncLog.	Системные данные
`GcDeadBlobDataBytes`	Данные блобов, которые барьеры сборки мусора разрешают удалить.	Мусор
`GcDeadMetadataBytes`	Метаданные, которые барьеры сборки мусора разрешают удалить.	Мусор

`MergeRedundantBlobDataBytes`	Старые или дублирующие физические данные, не нужные для сохранения логического значения после слияния.	Мусор
`MergeRedundantMetadataBytes`	Старые или дублирующие метаданные, не нужные после слияния.	Мусор
`WritePaddingBytes`	Дополнение, записанное из-за выравнивания данных внутри Hull или Huge-записи по границе блока записи PDisk.	Фрагментация
`SlotInternalFragmentationBytes`	Свободный хвост внутри занятого Huge-слота после выровненной записи.	Фрагментация
`FreeSlotBytes`	Свободные и незаблокированные Huge-слоты.	Фрагментация
`FreeStripeBytes`	Свободная ёмкость в общих stripe-чанках.	Фрагментация
`ChunkTailBytes`	Остаток ёмкости в выделенных чанках, не отнесённый к другой категории. Для SyncLog сюда входят сообщённые им свободные байты.	Прочее
`FreeChunkReserveBytes`	Свободные чанки в резерве Huge-аллокатора.	Прочее
`LockedOrQuarantinedBytes`	Заблокированные свободные Huge-слоты и заблокированная свободная ёмкость stripe-чанков.	Прочее
`UnclassifiedBytes`	Байты, для которых нет безопасной семантической классификации.	Прочее

Общие stripe-чанки

Hull может размещать экстенты SST и Huge-данные в одном физическом stripe-чанке. Такой чанк один раз входит в `StripeHeap.ChunkCount` и не входит в `ChunkCount` компонентов `LogoBlobs`, `Blocks`, `Barriers` и `Huge`.

Отчёт относит занятые экстенты к владеющему ими компоненту через `StripedBytes`. Свободное место stripe попадает в `Huge.Breakdown.FreeStripeBytes`, заблокированное свободное место — в `Huge.Breakdown.LockedOrQuarantinedBytes`, а остаток без безопасной классификации — в `Huge.Breakdown.UnclassifiedBytes`. Когда статистика stripe доступна, построитель отчёта распределяет всю ёмкость stripe, поэтому внутри этого отчёта сумма `StripedBytes` для `LogoBlobs`, `Blocks`, `Barriers` и `Huge.Total` равна `StripeHeap.AllocatedBytes`.

`StripeHeap.UsedBytes`, `FreeBytes` и `LockedFreeBytes` — сводная статистика аллокатора. Не прибавляйте их к `AccountedBytes`: ёмкость stripe-чанков уже представлена в детализации компонентов.

Классы размера Huge

`Huge.SizeClasses` описывает выделенные чанки Huge-аллокатора. Huge-данные в stripe-чанках не входят в эти классы размера. Каждая запись содержит размер слота, число слотов на чанк, число чанков, счётчики состояний слотов и `Breakdown` полной ёмкости класса.

Поле	Значение
`SlotSizeBytes`	Физический размер одного слота класса.
`SlotsPerChunk`	Число слотов класса в одном чанке.
`ChunkCount`	Число выделенных чанков, назначенных классу.
`LiveSlotCount`	Число слотов с выбранным актуальным представлением блоба.
`GcDeadSlotCount`	Число слотов с данными, удаляемыми сборкой мусора.
`MergeRedundantSlotCount`	Число слотов с избыточными физическими представлениями.
`UnclassifiedSlotCount`	Число слотов без безопасной семантической классификации.
`Breakdown`	Распределение полной ёмкости чанков класса по категориям байтов.

`Huge.FreeReserveChunks` показывает текущее число свободных чанков Huge-аллокатора. Эти чанки также входят в `Huge.Total` через `FreeChunkReserveBytes`.

Если бы счётчики аллокатора и сканирование индекса описывали одно состояние VDisk, сумма `LiveSlotCount`, `GcDeadSlotCount`, `MergeRedundantSlotCount` и `UnclassifiedSlotCount` была бы равна числу несвободных слотов. Источники снимаются раздельно, поэтому не ожидайте этого равенства в каждом отчёте. Свободные и заблокированные свободные слоты представлены только байтами в `Breakdown`. Неописанные слоты добавляются в `UnclassifiedSlotCount`. Если счётчики аллокатора противоречат семантической классификации, весь класс размера относится к `UnclassifiedBytes`. Такое поведение сохраняет физическую ёмкость и не публикует недостоверное распределение.

Динамические счётчики

Последний успешный отчёт экспортируется в подгруппе `subsystem=vdisk_space_report` счётчиков каждого VDisk.

Группа счётчиков	Содержимое
Корень	`ChunkSizeBytes`, `PDiskAllocatedChunks`, `PDiskAllocatedBytes`, `AccountedBytes` и знаковый `ReconciliationDeltaBytes`. `UnaccountedBytes` и `OveraccountedBytes` представляют положительную и отрицательную стороны разности неотрицательными значениями.
`scope=total`	Все поля `TVDiskSpaceBreakdown` для всего отчёта.

`component=inplace_blobs`, `blocks`, `barriers`, `huge_blobs`, `sync_log`, `unattributed`	`ChunkCount`, `AllocatedBytes`, `StripedBytes`, `AccountedBytes` и все поля детализации.
`scope=stripe_heap`	Счётчики чанков, выделенных, занятых, свободных и заблокированных свободных байтов stripe heap.
`slot_size_bytes=<size>`	Счётчики и детализация выделенного Huge-класса размера.
`chunk_keeper_subsystem=<id>`	Счётчики и детализация подсистемы ChunkKeeper.

Операционные счётчики в корне включают `RefreshInProgress`, `LastAttemptSuccessful`, `RefreshSuccesses`, `RefreshFailures`, `PeriodicTicksSkipped`, `ColdCacheRequests`, `LastRefreshDurationMs`, `LastRefreshCpuTimeUs`, `LastRefreshQuanta`, `LastRefreshVisitedKeys` и `LastRefreshPhysicalRecords`.

Как строится отчёт

Воркер сбора работает в batch pool VDisk и освобождает снапшоты Hull между короткими квантами сканирования:

1. Запрашивает число выделенных чанков у PDisk, статистику аллокатора и список stripe-чанков у HugeKeeper, а также компактную статистику у SyncLog и ChunkKeeper.
2. Ждёт ответы источников не более 10 секунд. Статистика PDisk обязательна. При отсутствии вспомогательной статистики возвращает `ERROR` с частичным отчётом.
3. Последовательно обходит индексы Hull для `LogoBlobs`, `Blocks` и `Barriers`. Перед каждым квантом запрашивает новый снапшот Hull.
4. Для каждого ключа просматривает все физические записи, сливает их логическое значение и применяет барьеры сборки мусора. Выбранное физическое представление классифицируется как полезное, удаляемые записи — как `GcDead*`, остальные версии — как `MergeRedundant*`.
5. После кванта сохраняет позицию обхода, уничтожает снапшот и при необходимости продолжает после паузы.
6. Объединяет оценки Hull с данными о выделенных Huge-слотах, общих stripe-чанках, SyncLog, ChunkKeeper и выделении PDisk.
7. При успехе менеджер кеширует отчёт и публикует динамические счётчики.

Целевой квант сканирования составляет 5 миллисекунд, после него планируется пауза 10 миллисекунд. Время проверяется после обработки полного ключа, поэтому один большой ключ может увеличить длительность кванта. Менеджер прерывает попытку, если она длится более 30 минут.

Сканер обходит индексы Hull в порядке убывания ключей и между квантами продолжает с ключа ниже последнего обработанного. Ключи, добавленные выше сохранённой границы, не увеличивают длительность текущего сканирования, но не попадают в отчёт. Ключи, добавленные ниже границы, ещё могут быть учтены. Поэтому отчёт не консистентен: значения и суммы в разных разделах могут описывать разные состояния VDisk и не обязаны совпадать.

Подсистемы распространения метаданных: StateStorage, Board, SchemeBoard

В кластере YDB работают три взаимосвязанные подсистемы, которые обеспечивают распространение метаданных между узлами: **StateStorage**, **Board** и **SchemeBoard**. Каждая из них решает свою задачу, но все три построены на одном архитектурном принципе — распределённом кворумном сервисе с детерминированной адресацией реплик.

Эта статья объясняет, зачем нужны эти подсистемы, как они устроены и как работают. Обзор без деталей ядра для пользователей документации — в разделе Сервисы распространения метаданных. Инструкции по конфигурированию и изменению конфигурации описаны в разделе Конфигурирование подсистем распространения метаданных.

Зачем нужны подсистемы распространения метаданных

Кластер YDB — это распределённая система, в которой одновременно могут работать миллионы таблеток на тысячах узлов. Чтобы компоненты кластера могли взаимодействовать друг с другом, им необходимо знать:

- где сейчас работает лидер конкретной таблетки и как к нему обратиться (**StateStorage**);
- какие узлы предоставляют те или иные сервисы, например, точки подключения для клиентов (**Board**);
- какова актуальная схема базы данных (**SchemeBoard**).

Распространять эти данные с одного узла кластера плохо — это создает высокую нагрузку и проблемы при выходе из строя этого узла. Поэтому метаданные распределяются по множеству узлов кластера с помощью трёх специализированных подсистем.

Подсистемы

StateStorage — хранилище состояния таблеток

StateStorage — это распределённый сервис, который хранит актуальное состояние таблеток: где работает лидер, каков его идентификатор актора, поколение и шаг.

Что хранит StateStorage

Для каждой таблетки StateStorage хранит:

- **Текущий лидер** — идентификатор актора (ActorId) лидера таблетки.
- **Поколение и шаг** (`generation:step`) — монотонно возрастающие счётчики, используемые для разрешения конфликтов при выборе лидера.
- **Список фолловеров** — идентификаторы акторов реплик таблетки (если они есть).
- **Блокировка** — признак того, что таблетка заблокирована в процессе выбора нового лидера.

Как используется StateStorage

StateStorage выполняет роль **службы разрешения имён таблеток**: любой узел кластера может по `TabletId` узнать `ActorId` текущего лидера и напрямую обратиться к нему.

StateStorage также участвует в **процессе выбора лидера таблетки**. Когда таблетка запускается или перезапускается, она регистрирует себя в StateStorage и получает подтверждение от кворума реплик.

> **Примечание**
>
> Данные в StateStorage являются **волатильными**: они хранятся только в памяти реплик и теряются при перезапуске процессов. StateStorage не является постоянным хранилищем — он содержит только ту информацию, которую легко восстановить как при запуске таблеток, так и при запуске самих реплик.

Адресация в StateStorage

Выборка реплик для конкретной таблетки вычисляется по её `TabletId`. Это означает, что реплики разных таблеток могут находиться на разных узлах, что обеспечивает равномерное распределение нагрузки.

Board — доска объявлений сервисов

Board — это распределённый сервис для публикации и поиска метаданных в формате «путь → набор записей». Он работает по модели «публикация — подписка»: один или несколько акторов публикуют информацию по некоторому пути, а другие акторы подписываются на этот путь и получают актуальный список публикаций.

Что хранит Board

Board хранит пары «путь → полезная нагрузка» (payload). Несколько акторов могут одновременно публиковать данные по одному пути — Board хранит все публикации и предоставляет их подписчикам в виде списка.

Как используется Board

Основное применение Board — **хранение информации об эндпоинтах** (точках подключения) баз данных. Когда узел базы данных запускается, он публикует свой адрес в Board по пути, соответствующему имени базы данных. Клиенты и другие компоненты кластера подписываются на этот путь и получают актуальный список доступных эндпоинтов.

Механизм работы:

1. Актор-публикатор регистрирует запись в Board по заданному пути.
2. Актор-подписчик запрашивает список записей по пути и получает уведомления об изменениях.

В отличие от StateStorage, Board не привязан к конкретным таблеткам — он предназначен для произвольных сервисов, которым нужно сделать некоторые данные легко доступными в кластере.

Адресация в Board

Выборка реплик для конкретного пути вычисляется по хешу от этого пути. Это обеспечивает детерминированную маршрутизацию: все публикации и подписки по одному пути попадают на одни и те же реплики.

SchemeBoard — распространение схемных метаданных

SchemeBoard — это распределённый сервис для хранения и распространения метаданных о схеме баз данных: таблицах, индексах, правах доступа и других схемных объектах.

Что хранит SchemeBoard

SchemeBoard хранит описания схемных объектов: таблиц, директорий, индексов, топиков и других объектов схемы. Для каждого объекта хранится его полное описание, включая структуру, настройки и права доступа.

Как используется SchemeBoard

SchemeBoard является **кешем схемных метаданных** для всех компонентов кластера. Когда узел базы данных получает запрос, ему необходимо знать структуру таблиц, с которыми работает запрос. Вместо того чтобы каждый раз обращаться к SchemeShard (таблетке, которая является источником истины для схемы), узел читает схему из локального кеша, который синхронизируется через SchemeBoard.

Механизм работы:

1. При изменении схемы SchemeShard публикует обновление в SchemeBoard.
2. Узлы баз данных подписываются на изменения нужных им схемных объектов и получают уведомления об обновлениях.

Это позволяет узлам баз данных работать с актуальной схемой без постоянных обращений к SchemeShard, что существенно снижает нагрузку на него и уменьшает задержки при выполнении запросов.

Адресация в SchemeBoard

Выборка реплик для конкретного схемного объекта вычисляется по хешу от пути к этому объекту. Это обеспечивает детерминированную маршрутизацию запросов к схемным метаданным.

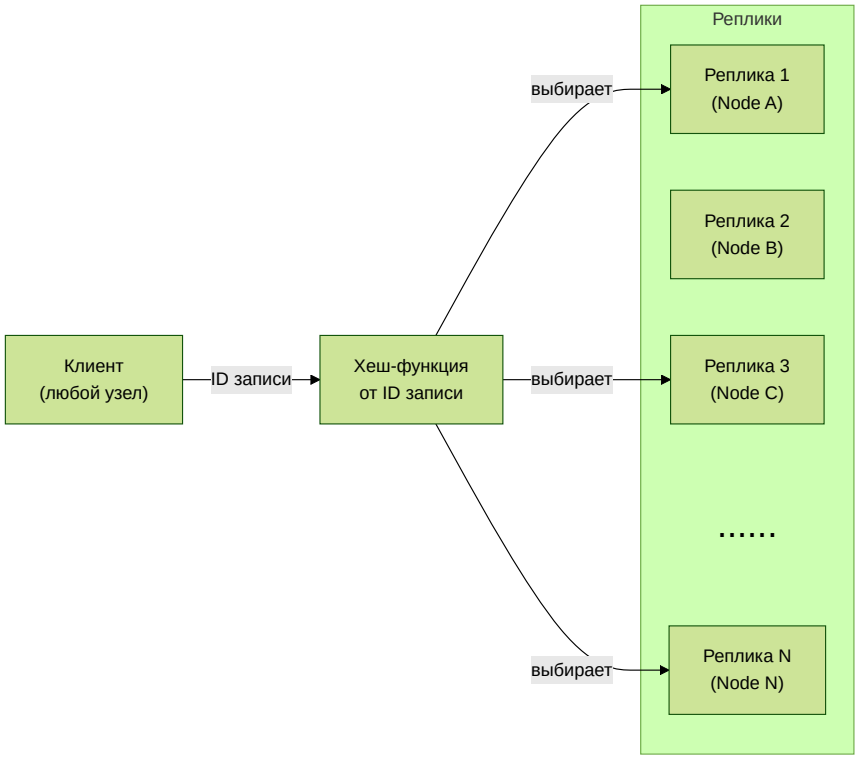
Сравнение подсистем

Характеристика	StateStorage	Board	SchemeBoard
Назначение	Хранение состояния таблеток	Публикация метаданных сервисов	Распространение схемных метаданных
Тип данных	Состояние лидера таблетки (ActorId, generation:step)	Произвольные пары путь → payload	Описания схемных объектов
Ключ адресации	TabletId	Путь (строка)	Путь к схемному объекту
Основные потребители	Tablet Pipe	gRPC-прокси, клиенты (поиск эндпоинтов)	Узлы баз данных (кеш схемы)
Источник данных	Сами таблетки (при запуске/выборе лидера)	Акторы-публикаторы	SchemeShard

Детерминированная выборка реплик

Каждая запись в подсистеме адресуется по идентификатору — например, `TabletId` для StateStorage или путь к схемному объекту для SchemeBoard. По этому идентификатору с помощью хеш-функции вычисляется фиксированный набор реплик, на которых хранится данная запись.

Это ключевое свойство: **любой узел кластера может самостоятельно вычислить, на каких репликах хранится нужная ему запись**. Это делает подсистемы масштабируемыми и устойчивыми к отказам.



Кворум

Для работы с каждой записью выбирается определенное количество реплик `nto_select`. Операции записи система пытается выполнять со всеми `nto_select` репликами, но при этом допускается отказ меньшинства реплик, и такой отказ не приводит к временной недоступности или приостановке работы.
Для успешного выполнения операции с записью достаточно получить ответ от **большинства** выбранных реплик — то есть от

`nto_select / 2 + 1` реплик, где `nto_select` — общее число реплик в выборке. Это означает, что часть реплик может быть недоступна, и подсистема продолжит работать.

Именно кворумный принцип позволяет безопасно выводить узлы на обслуживание и выполнять роллинг-рестарты кластера без потери работоспособности.

Ключевые понятия

Реплика

Реплика — это актор, работающий на узле кластера и хранящий часть метаданных подсистемы. Реплика принимает запросы на чтение и запись, участвует в формировании кворума.

Каждая реплика независима: она продолжает обслуживать запросы, даже если другие реплики временно недоступны. Данные, которые не удалось записать на недоступную реплику, будут переданы ей позже при восстановлении её работоспособности — либо явно через повторную попытку, либо автоматически через механизм синхронизации состояния.

Кольцо

Мотивация

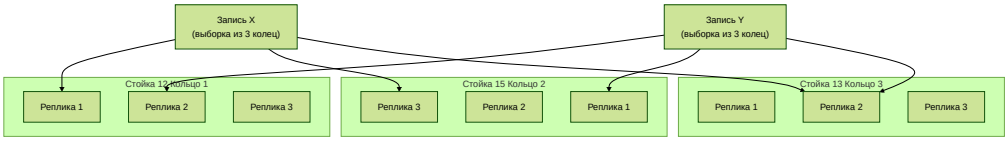
Для равномерного распределения нагрузки по узлам кластера подсистемам может потребоваться большое количество реплик: чем их больше, тем меньше запросов приходится на каждую реплику. Однако с ростом числа реплик возникает вопрос отказоустойчивости: при отказе домена отказа (серверной стойки) из строя одновременно может выйти несколько реплик, и это скажется на кворуме.

Определение и принцип работы

Кольцо — это группа реплик, из которых для каждой конкретной записи выбирается **не более одной** реплики. Выбор реплики внутри кольца детерминирован и основан на хеш-функции от идентификатора записи. Таким образом, разные записи обслуживаются разными репликами внутри одного кольца, что обеспечивает равномерное распределение запросов.

Кольца и домены отказа

Реплики каждого кольца размещаются в пределах одного **домена отказа** (fail domain) — одной, или нескольких серверных стойках. Реплики из разных колец при этом размещаются в разных стойках. Не допускается чтобы в одной стойке оказались реплики из разных колец. Это гарантирует, что при выходе из строя любой стойки из каждой выборки потеряется **не более одной реплики** — той, что принадлежит кольцу, чьи реплики находились в этой стойке.

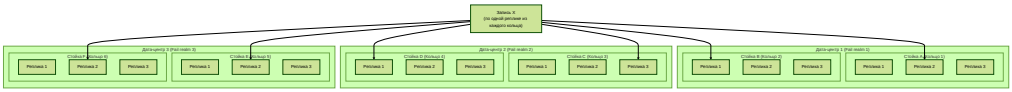


Кольца и области отказа

В кластерах с несколькими **областями отказа** (fail realm) — например, дата-центрами — применяются дополнительные ограничения на размещение реплик по кольцам:

- в одно кольцо не включают реплики из разных областей отказа;
- количество колец в каждой области отказа ограничивается.

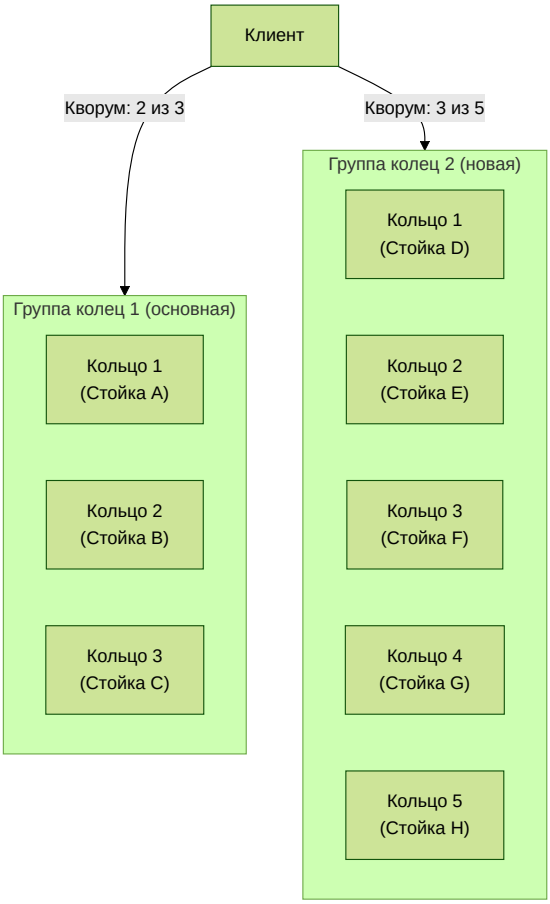
Эти ограничения позволяют контролировать, сколько колец затронет выход из строя целого дата-центра: если каждая область отказа содержит не более `k` колец, то при её отказе из строя выйдет не более `k` реплик из каждой выборки.



Группа колец

Группа колец — это набор колец, по которому собирается **отдельный кворум**. Подсистема может одновременно работать с несколькими группами колец, собирая кворум в каждой из них независимо.

Группы колец — это механизм **бесшовного изменения конфигурации**. Они позволяют вводить новые наборы реплик и выводить старые без остановки кластера. Подробнее об этом — в разделе Группы колец и смена конфигурации.



Группы колец и смена конфигурации

Бесшовная смена конфигурации

Изменение конфигурации подсистем — например, перенос реплик на другие узлы или изменение числа реплик — выполняется через добавление и удаление групп колец. Это позволяет менять конфигурацию без остановки кластера и без потери доступности.

Процесс смены конфигурации состоит из нескольких шагов:

1. Добавляется новая группа колец с нужной конфигурацией. На этом этапе она помечается флагом `write_only` — это означает, что новая группа получает все записи (синхронизируется с данными), но **не участвует в кворуме на чтение**. Запросы на чтение по-прежнему обслуживает старая группа.
2. После того как новая группа синхронизировалась и стала полноценной, флаг `write_only` снимается. Теперь обе группы участвуют в кворуме.
3. Старая группа помечается флагом `write_only`, а новая становится основной.
4. Старая группа удаляется.

Между шагами необходимо выдерживать паузу (не менее одной минуты), чтобы конфигурация успела распространиться по всем узлам кластера.

Подробные инструкции по ручной смене конфигурации приведены в разделе Конфигурирование подсистем распространения метаданных.

Автоматическая реконфигурация (SelfHeal)

В кластерах с конфигурацией V2 доступен механизм **SelfHeal подсистем распространения метаданных** — автоматическое управление конфигурациями State Storage, Board и SchemeBoard. Он отслеживает состояние узлов кластера и при необходимости:

- переносит реплики с вышедших из строя узлов на работоспособные;
- может добавлять новые реплики при расширении кластера с учётом настроек и доступных узлов.

SelfHeal работает через тот же механизм групп колец, что и ручная реконфигурация, но выполняет все шаги автоматически. Подробнее — в разделе SelfHeal подсистем распространения метаданных.

Группы колец в конфигурации двух дата-центров

В конфигурации с двумя дата-центрами (режим bridge) каждый дата-центр (pile) имеет **собственную группу колец**. Это позволяет:

- каждому дата-центру работать автономно, собирая кворум внутри своей группы;
- быстро и бесшовно переключать, какой дата-центр является основным (primary), без изменения конфигурации реплик.

Подробнее о режиме bridge описано в разделе Режим bridge.

Отказоустойчивость и модель отказа

Подсистемы распространения метаданных спроектированы с учётом модели отказа YDB, основанной на концепциях доменов отказа (обычно — серверных стоек) и областей отказа (обычно — дата-центров).

Правило размещения реплик (разные кольца — в разных стойках; реплики одного кольца могут быть в одной или нескольких стойках) обеспечивает следующие гарантии:

- **Отказ отдельного узла**: теряется не более одной реплики из каждой выборки. Кворум сохраняется.
- **Отказ целой стойки**: поскольку все реплики одного кольца находятся в одной стойке, теряется ровно одна реплика из каждой выборки. При `nto_select = 5` кворум (3 из 5) сохраняется даже при потере целой стойки.
- **Роллинг-рестарт**: последовательный перезапуск узлов не нарушает кворум, так как в каждый момент времени недоступна лишь часть реплик.

Связанные материалы

- Сервисы распространения метаданных — обзор для пользователей документации.
- Конфигурирование подсистем распространения метаданных — инструкции по ручному изменению конфигурации.
- SelfHeal подсистем распространения метаданных — автоматическое управление конфигурациями State Storage, Board и SchemeBoard.
- Режим bridge — конфигурация с двумя дата-центрами и роль групп колец в ней.
- Топология кластера — модель отказа, домены и области отказа.
- Глоссарий — определения терминов: StateStorage, Board, SchemeBoard, таблетка, ActorId.

LocalDB: персистентные незакомиченные изменения

Таблеткам может понадобиться сохранять большое количество изменений на протяжении длительного времени, а затем целиком закоммитить или отменить их. Для поддержки этого сценария LocalDB позволяет маркировать незакомиченные изменения в таблицах уникальным 64-битным идентификатором транзакции (`TxId`). Эти данные сохраняются в таблице рядом с закомиченными данными, но не видны для остальных запросов до коммита. Коммит или отмена транзакции атомарная и дешёвая, а со временем эти данные интегрируются в таблицу в обычном закомиченном виде. Таким образом, незакомиченные данные не хранятся в оперативной памяти длительное время и не ограничены её объемом.

Этот механизм используется в качестве основы для следующей функциональности:

- Сохранение незакомиченных изменений в долгих YQL транзакциях и возможность видеть эти изменения в той же транзакции до их коммита;
- Сохранение эффектов в волатильных распределённых транзакциях с возможностью отката;
- Межкластерная консистентная репликация, в которой изменения потоково применяются небольшими батчами, и периодически коммитятся, делая видимым очередной консистентный снимок исходной базы данных.

Ограничения

Этот механизм имеет следующие ограничения:

1. По одному ключу можно параллельно делать изменения в рамках разных `TxId` , но эти транзакции должны коммититься в том же порядке, в котором происходила запись изменений. Например, если по ключу K было изменение от транзакции tx1, а затем от транзакции tx2, то можно закоммитить транзакцию tx1, а затем tx2, и оба изменения будут видны. Однако, если сначала закоммитить tx2, то tx1 должна быть отменена.
2. Количество незакомиченных транзакций и объём данных в них по конкретному ключу должны быть ограничены вышестоящими уровнями.
3. `TxId` должен быть глобально уникальным и не должен переиспользоваться после коммита или отмены транзакции, в том числе между разными шардами.

Запись незакомиченных изменений

В логе восстановления (см. flat_redo_writer.h) есть следующие события:

- EvUpdateTx сохраняет изменения в таблице с указанием незакомиченного `TxId` . Это событие создаётся методом TDatabase::UpdateTx локальной базы данных (LocalDB).
- EvRemoveTx используется для удаленения указанного `TxId` , когда выполняется отмена транзакции. Это событие создаётся методом TDatabase::RemoveTx локальной базы данных.
- EvCommitTx используется для коммита указанного `TxId` с указанием MVCC версии коммита. Это событие создаётся методом TDatabase::CommitTx локальной базы данных.

Хранение незакомиченных изменений в MemTable

MemTable в LocalDB — это небольшое отсортированное по ключу дерево недавно сделанных изменений, которое хранится в памяти. В качестве ключа в этом дереве используется ключ таблицы, а в качестве значения — указатель на цепочку изменений для соответствующего ключа. Для каждого изменения указывается MVCC версия этого изменения (пара `Step` / `TxId` с глобальным временем коммита). Строки в цепочке изменений смержены в рамках одного MemTable. Например, представим что у нас были следующие операции для ключа K:

Версия	Операция
`v1000/10`	`UPDATE ... SET A = 1`
`v2000/11`	`UPDATE ... SET B = 2`
`v3000/12`	`UPDATE ... SET C = 3`

Тогда цепочка строк с изменениями для этого ключа в единственном MemTable будет выглядеть следующим образом:

Версия	Строка
`v3000/12`	`SET A = 1, B = 2, C = 3`
`v2000/11`	`SET A = 1, B = 2`
`v1000/10`	`SET A = 1`

В некоторых случаях MemTable может разделиться между операциями, что происходит, например, с началом компакшена. Тогда MemTable из примера выше выглядел бы так:

MemTable	Версия	Строка
Эпоха 2	`v3000/12`	`SET B = 2, C = 3`
Эпоха 2	`v2000/11`	`SET B = 2`
Эпоха 1	`v1000/10`	`SET A = 1`

Новые изменения применяются к текущему MemTable, даже если они ещё не были закомичены. При записи незакомиченные изменения помечаются специальной версией, с указанием максимального значения для `Step` (как бы подразумевая далёкое будущее) и `TxId` транзакции в качестве `TxId` . Мёрж нижестоящих изменений не производится. Например, представим, что далее мы выполнили следующие незакомиченные операции для ключа K:

TxId	Операция
15	`UPDATE ... SET C = 10`
13	`UPDATE ... SET B = 20`

Цепочка изменений в MemTable теперь будет выглядеть так:

Версия	Строка
`v{max}/13`	`SET B = 20`
`v{max}/15`	`SET C = 10`
`v3000/12`	`SET A = 1, B = 2, C = 3`
`v2000/11`	`SET A = 1, B = 2`
`v1000/10`	`SET A = 1`

Во время чтения для изменений с версией `Step == max`, итератор проверяет их `TxId` по таблице транзакций. Для закомиченных транзакций там указана MVCC версия коммита. Далее он применяет все закомиченные изменения, пока не найдёт и не применит первую смерженную запись, у которой версия `Step != max`.

Представим, что мы закоммитили транзакцию с `TxId = 13` в точке `v4000/20`. В этот момент в таблице транзакций появится запись `[13] => v4000/20`, и эта транзакция станет закомиченной. Все последующие чтения будут видеть изменения с `TxId = 13`, пропуская транзакцию с `TxId = 15`, так как она не была закоммичена до `TxId = 13`. Цепочка изменений для ключа K при этом не меняется.

Теперь представим, что мы выполнили `SET A = 30` в версии `v5000/21`. Итоговая цепочка изменений будет выглядеть следующим образом:

Версия	Строка
`v5000/21`	`SET A = 30, B = 20, C = 3`
`v{max}/13`	`SET B = 20`
`v{max}/15`	`SET C = 10`
`v3000/12`	`SET A = 1, B = 2, C = 3`
`v2000/11`	`SET A = 1, B = 2`
`v1000/10`	`SET A = 1`

Новая строка была добавлена в смёрженном состоянии, включая изменения для `TxId = 13`. Так как транзакция `TxId = 15` не закоммичена, её изменения были пропущены, что отразилось в смёрженном состоянии для `v5000/21`. Важно, чтобы вышестоящий код не закоммитил `TxId = 15` после этого, так как подобный коммит приведёт в аномалии: часть версий на чтении будут видеть транзакцию как закомиченную, а часть не будет.

Компакшен незакомиченных изменений

Компакшен берёт часть данных таблицы, мёржит их в отсортированном порядке и пишет в виде новой SST, которая заменяет собой скомпакченные данные. Если в компакшене участвует MemTable, то это подразумевает также компакшен части лога изменений. В скомпакшенном логе могут встречаться сообщения `EvRemoveTx` / `EvCommitTx`, которые меняют таблицу закомиченных транзакций — они также должны оказаться в персистентном сторадже. Это реализовано с помощью записи `TxStatus` блобов (см. flat_page_txstatus.h), которые LocalDB сохраняет наравне с записью SST с данными. В этих блобах сохраняется текущий список закомиченных и удалённых транзакций, который заменяют собой весь ранее записанный лог в части `EvRemoveTx` / `EvCommitTx` событий. На вход в компакшен передаётся ссылка на всю текущую таблицу, но во время записи таблица фильтруется, сохраняя только те транзакции, которые упоминались в компактящихся MemTable или предыдущих `TxStatus` страницах.

На уровне страниц с данными (см. flat_page_data.h) незакомиченные изменения из MemTable (или других SST) агрегируются по `TxId` с сохранением их относительного порядка и добавляются перед основной записью в виде незакомиченных дельта записей. У записи есть MVCC флаги (`HasHistory`, `IsVersioned`, `IsErased`), которые говорят о наличии по ключу MVCC истории и наличии полей с MVCC версией строки. Для дельта записей используется особый флаг IsDelta, который представляет из себя флаг `HasHistory` без указания других MVCC флагов. Для строк с дельта флагом после фиксированной части в полях расширения хранится структура TDelta, в которой указывается `TxId` незакомиченной на момент компакшена транзакции.

По одному ключу может быть несколько дельта записей, а также опционально самая свежая закомиченная запись. Так как по одному ключу на странице данных может быть только одна запись, таблица смещений указывает на первую дельта запись, а остальные записи при этом доступны через таблицу альтернативных свещений для этой записи:

Смещение	Описание
-X*8	смещение Main
...	...
-16	смещение Delta 2
-8	смещение Delta 1
0	заголовок Delta 0
...	...
смещение Delta 1	заголовок Delta 1
...	...
смещение Main	заголовок Main

Имея указатель на запись Delta 0, остальные записи для ключа можно получить через метод `GetAltRecord(size_t index)`, где `index` — номер записи в цепочке (1 для Delta 1). Цепочка заканчивается либо указателем на основную запись (без дельта флага), либо 0, если основная запись отсутствует.

Продолжая начатый выше пример, предположим, что после записи транзакции с `TxId = 13` текущий MemTable целиком скомпактили. Запись по 32-битному ключу K может выглядеть следующим образом (смещения указаны относительно указателя в списке записей на странице):

Смещение	Значение	Описание
-16	58	смещение Main
-8	29	смещение Delta 1
0	0x21	Delta 0: IsDelta + ERowOp::Upsert
1	0x00	.. ключевая колонка не-NULL
2	K	.. ключевая колонка (32-bit)
6	0x00	.. колонка A пустая
7	0	.. колонка A (32-bit)
11	0x01	.. колонка B = ECellOp::Set
12	20	.. колонка B (32-bit)
16	0x00	.. колонка C пустая
17	0	.. колонка C (32-bit)
21	13	.. TDelta::TxId
29	0x21	Delta 1: IsDelta + ERowOp::Upsert
30	0x00	.. ключевая колонка не-NULL
31	K	.. ключевая колонка (32-bit)
35	0x00	.. колонка A пустая
36	0	.. колонка A (32-bit)
40	0x00	.. колонка B пустая
41	0	.. колонка B (32-bit)
45	0x01	.. колонка C = ECellOp::Set
46	10	.. колонка C (32-bit)
50	15	.. TDelta::TxId
58	0x61	Main: HasHistory + IsVersioned + ERowOp::Upsert
59	0x00	.. ключевая колонка не-NULL
60	K	.. ключевая колонка (32-bit)
64	0x01	.. колонка A = ECellOp::Set
65	1	.. колонка A (32-bit)
69	0x01	.. колонка B = ECellOp::Set
70	2	.. колонка B (32-bit)
74	0x01	.. колонка C = ECellOp::Set
75	3	.. колонка C (32-bit)
79	3000	.. RowVersion.Step
87	12	.. RowVersion.TxId
95	-	Конец записи

При этом оставшиеся две записи будут в исторических данных в ключах `(RowId, 2000, 11)` и `(RowId, 1000, 10)`, соответственно. Об их наличии говорит флаг `HasHistory` в основной записи.

В процессе компакшена итератор работает в специальном режиме с перебором всех дельт и всех версий для каждого ключа. Реализация скана для компакшена (см. flat_ops_compact.h) сначала агрегирует незакомиченные на момент начала компакшена дельты по `TxId` с сохранением их относительного порядка. В случае, если изменения от разных `TxId` накладываются друг на друга, их порядок может поменяться произвольным образом. Такое изменение порядка валидно, так как такие транзакции пересекаются по порядку записи и вышестоящий уровень не имеет права коммитить их одновременно. Когда все незакомиченные дельты сагрегированы, они пишутся в правильном порядке в результирующую SST (см. flat_part_writer.h и flat_page_writer.h). После этого начинается перебор версий строк для ключа, которые пишутся в SST в порядке уменьшения версий.

Когда итератор встречает первую закомиченную дельту, версия коммита этой дельты берётся в качестве версии строки. В состояние строки мёржатся все закомиченные дельты ниже, включая первую закомиченную версию строки с каждого уровня

LSM, участвующего в компакшене. Для перехода к следующей версии итератор пропускает закомиченные дельты с версией большей либо равной последней версии и процесс повторяется.

Если по одному ключу записали и скомпактили большое количество дельт, после чего быстро закомитили с разными версиями, то формирование всех версий строки будет расти квадратично с ростом кол-ва дельт. Это одна из причин, по которой вышестоящие уровни должны контролировать количество дельт по ключу и не допускать их бесконтрольного роста. Другим ограничением сейчас является то, что все незакомиченные дельты необходимо держать в памяти, т.к. они могут быть закомиченны в любой момент и мы должны проверять это во время поиска. При этом все дельты для ключа должны находиться на одной странице.

Статистика по незакомиченным транзакциям

Оптимистично большинство пишущих транзакций в итоге коммитятся успешно, но иногда транзакции приходится отменять, в том числе уже после компакшена. Так как после их отмены большое количество данных может оказаться по факту удалено, для целей сборки мусора в SST сохраняется страница `TxIdStats` (см. flat_page_txidstat.h) со статистикой по количеству строк и байт, занимаемых каждым `TxId`. По мере того, как транзакции отменяются, стратегия компакшена агрегирует количество высвобождающихся байт, и может принять решение о запуске дополнительных компакшенов для сборки мусора.

Страницы `TxIdStats` также используются для постепенного уменьшения таблицы коммитов. После того, как транзакция оказывается закоммичена, запись об этом хранится в хеш-таблице в памяти до тех пор, пока на соответствующий `TxId` есть хотя бы одна ссылка. По мере компакшена SST незакомиченные дельты перезаписываются в виде закомиченных записей и ссылки на страницах `TxIdStats` пропадают. Если для соответствующего `TxId` был сделан коммит или отмена, плюс на него не остаётся ссылок и в MemTable и в SST, то эта запись удаляется из таблицы коммитов и освобождает память.

Размер таблицы коммитов в памяти ограничивается на уровне даташардов количеством открытых транзакций в каждый момент времени. Открытые транзакции — те, которые ещё не закоммичены или не отменены. Во время компакшена в SST могут быть записаны дельты только для открытых на момент начала компакшена транзакций, так как закомиченные дельты переписываются в обычные закомиченные строки, а от отменённых транзакций не остаётся каких-либо следов. Соответственно, размер таблицы коммитов ограничен количеством открытых транзакций помноженным на количество SST.

Одалживание SST с незакомиченными изменениями

При копировании таблиц, а также при сплитах и мёржах DataShard используется механизм «одалживания» SST в LocalDB. При этом SST из шарда-источника мёржится в соответствующую таблицу на шарде-приёмнике. С добавлением незакомиченных изменений там могут быть как изменения из ещё открытых транзакций (актуально в случае репликации), так и закомиченные, но ещё не скомпакченные изменения. Для того, чтобы на шардах-приёмниках данные читались в том же виде, как и на шарде-источнике, одалживаются и мёржатся в том числе `TxStatus` блобы. Информациюя из них добавляется в таблицу коммитов.

Информация по транзакциям может отличаться на разных шардах. Рассмотрим новый пример:

1. Транзакция пишет изменение по ключу K с `TxId` на шард S, при этом данные компактятся и изменения оказывается в крупной SST.
2. Шард S начинает превышать порог по размеру и сплитится на два шарда L и R.
3. SST, в котором есть изменения по ключу K, попадают на оба шарда с наложением фильтра.
4. Ключ K оказывается в шарде L, но транзакция также существует на шарде R, так как на неё есть ссылка из одолженной SST.
5. Транзакция коммитит изменение на шарде L, но при этом коммита на шарде R не происходит, така как логически и с точки зрения писателя на шарде R не было изменений.
6. Так как транзакция завершается, `TxId` на шарде R через какое-то время откатывается, так какв реальности в данных шарда при этом ничего не меняется.
7. Допустим, в дальнейшем шарды L и R снова смержились.
8. В такой ситуации в `TxStatus` от шарда L будет коммит, а в `TxStatus` от шарда R будет отмена для того же `TxId`. Так как транзакции успешно коммитят все изменения, а отмена может быть в том числе из-за наложенного фильтра, в случае мержа конфликтующей информации по `TxId` коммит «выигрывает» над отменой.

На практике DataShard перед очередным сплитом или мержем полностью компактятся, так что конфликтующая информация по статусу коммита не должна возникнуть. Например, это означает, что транзакции не могут решить закоммитить только часть изменений, все изменения для конкретного `TxId` должны быть закоммичены вместе. Это также означает, что идентификаторы транзакций нельзя переиспользовать, в том числе между разными шардами. В качестве `TxId` для незакомиченных изменений также могут использоваться только глобально уникальные идентификаторы.

Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

Hive

Hive — таблетка, отвечающая за управление другими таблетками, а именно за выбор узлов базы для их запуска, принятие решений о необходимости перебалансировки таблеток и т.п.

Создание и удаление таблеток инициируется таблеткой SchemeShard. При создании таблетки Hive присваивает ей уникальный TabletId, заполняет TabletStorageInfo, выбирает наиболее подходящий узел и отправляет на него команду на запуск таблетки. В некоторых нестандартных ситуациях отдельная таблетка может прервать работу, тогда узел, на котором она была запущена, отправляет сообщение в Hive. Также Hive предполагает, что если связь с некоторым узлом потеряна, то запущенные на нём таблетки прекратили работу. В таких случаях Hive перезапускает таблетки на других узлах с увеличением поколения.

Логика запуска таблеток подробно описана в статье Запуск таблеток.

В кластере YDB есть корневой Hive, который отвечает за системные таблетки всех баз данных кластера. Hive конкретной базы данных, в свою очередь, отвечает за таблетки, обслуживающие пользовательскую нагрузку этой базы. Все узлы кластера регистрируются в корневом Hive, а в Hive конкретной базы регистрируются только вычислительные узлы этой базы. При регистрации узел сообщает в Hive, какие типы таблеток и в каком количестве могут быть на нём запущены.

Метрики потребления ресурсов

Для распределения таблеток по узлам Hive учитывает потребление ресурсов. Для каждой таблетки отслеживается потребление четырёх типов ресурсов:

1. *CPU* — потребление процессора, рассчитывается как число микросекунд, затраченных на работу таблетки за последнюю секунду, и для визуализации переводится в проценты ядра.
2. *Memory* — потребление таблеткой оперативной памяти.
3. *Network* — генерируемый таблеткой объём трафика.
4. *Counter* — фиктивный ресурс, используемый для реализации равномерного распределения в штуках. Если у таблетки ненулевое значение потребления любого другого ресурса, то её значение Counter равно 0, иначе оно равно 1. Таким образом, Counter применяется для любых таблеток, для которых нет данных о реальном потреблении, а также для таблеток, для которых учёт реального потребления отключён: по умолчанию это только таблетки колоночных таблиц.

Дополнительно для определения перегруженных узлов используются метрики потребления ресурсов узла целиком: потребление оперативной памяти и ресурсов процессора в пулах акторной системы. Эти значения переводятся в относительную величину (число от 0 до 1), и их максимум используется как значение общего потребления ресурсов узла — *Node usage*. Также для всех метрик на стороне Hive применяется агрегация по окну, чтобы учитывать всплески нагрузки.

Информация о ресурсах используется при выборе узла для таблетки: к примеру, если про некоторую таблетку известно только её потребление CPU, то будет выбран узел из числа наименее загруженных по CPU. Если известно несколько ресурсов, то используется максимум из их относительных потреблений.

Автобалансировка

В определённые моменты Hive может запустить процесс автобалансировки, перемещающий таблетки между узлами для улучшения распределения нагрузки. Ситуации, в которых это происходит, перечислены ниже. Автобалансировщик работает итеративно, принимая решения о перевозе таблеток последовательно и по одной. Он выбирает самый загруженный узел, взвешенно-случайным образом выбирает на нём таблетку и находит для неё более подходящий узел. Этот процесс повторяется, пока сбалансированность не будет восстановлена. То, как именно определяется загруженность узла, зависит от типа балансировки: например, при дисбалансе потребления CPU учитывается потребление CPU, а при неравномерном распределении колоночной таблицы — количество таблеток.

Порогами, при которых запускается балансировка, можно управлять с помошью конфигурации Hive.

Дисбаланс потребления ресурсов

Для оценки сбалансированности потребления используется метрика *Scatter*, вычисляемая отдельно для каждого ресурса по формуле:

и — максимум и минимум потребления данного ресурса среди всех узлов, соответственно. Для нормировки потребления на каждом узле используется число доступных ресурсов на узле, которое может различаться между узлами. При низких нагрузках эта величина может сильно колебаться. Чтобы этого избежать, при вычислении считается, что потребление ресурса не может быть ниже 30%. Если превышает порог, запускается балансировка.

Максимальное из значений по разным ресурсам доступно в виде сенсора `MAX(Hive/BalanceScatter)` в подгруппе `tablets`.

Перегруженность узла

Наличие сильно загруженного узла может негативно сказываться на работе YDB: загруженность по CPU приводит к голоданию и увеличению задержки, а загруженность по памяти может привести к падению узла по out-of-memory. Балансировка запускается, если загрузка самого загруженного узла превышает 90%, а наименее загруженного — ниже 70%.

Этот вариант балансировки отличается от других более аггресивными перевозами таблеток.

Максимальное потребление на узле доступно в виде сенсора `MAX(Hive/BalanceUsageMax)` в подгруппе `tablets`.

Равномерное распределение конкретного объекта

Для таблеток, которые используют ресурс Counter, также отслеживается равномерность распределения таблеток каждого объекта (каждой таблицы) с помощью метрики `ObjectImbalance`, аналогичной описанной выше . При рестартах узлов равномерность может нарушаться, и тогда запускается балансировка.

Сенсор `MAX(Hive/ImbalancedObjects)` показывает максимальное среди Hive в подгруппе `tablets` количество объектов с ненулевым `ObjectImbalance`. Для каждого Hive значение `WorstObjectVariance` равно дисперсии числа таблеток по узлам у объекта с наибольшим `ObjectImbalance`; сенсор `MAX(Hive/WorstObjectVariance)` показывает максимум этих значений.

Смотрите также

- Настройка Hive
- Hive web-viewer
- Запуск таблеток

Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

Запуск таблеток

В данной статье описывается процесс запуска таблеток со стороны Hive. Hive принимает решение о запуске таблетки в различных ситуациях:

- Таблетка только что была создана.
- Утрачена связь с узлом, на котором была запущена таблетка.
- Узел, на котором была запущена таблетка, прислал сообщение о том, что она прекратила работу (например, из-за потери связи с дисковой подсистемой кластера).
- При перевозе таблетки в рамках автобалансировки.

В любой из этих ситуаций происходит следующее:

1. Таблетка добавляется в очередь на запуск.
2. При обработке очереди для неё выбирается подходящий узел.
3. На узел отправляется команда о запуске таблетки.

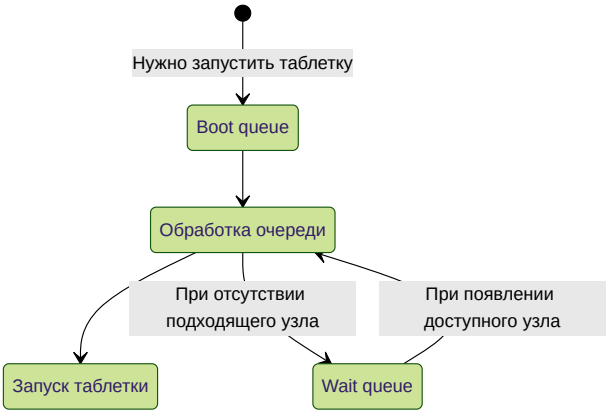
Очередь

Очередь на запуск, или *Boot queue*, хранится в памяти Hive и является приоритетной. Приоритет таблетки определяется следующими факторами:

1. Тип таблетки — системные таблетки имеют более высокий приоритет, чем пользовательские.
2. Метрики потребления ресурсов — таблетки с бóльшим потреблением имеют бóльший приоритет.
3. Таблетки, которые часто перезапускаются, имеют пониженный приоритет.

При обработке очереди за раз обрабатывается ограниченное количество таблеток (`max_boot_batch_size` в конфигурации). Это нужно для того, чтобы при запуске большого числа таблеток Hive не переставал на долгое время отвечать на другие запросы.

Если при обработке очередной таблетки оказывается, что её нельзя запустить ни на одном из узлов, то эта таблетка откладывается в отдельную очередь *Wait queue*. Когда доступность узла меняется (подключается новый узел, или с узла снимается ограничение в Hive UI), Hive возвращается к этим таблеткам и при обработке очереди на запуск чередует таблетки из Boot Queue и таблетки из Wait Queue.



Важно

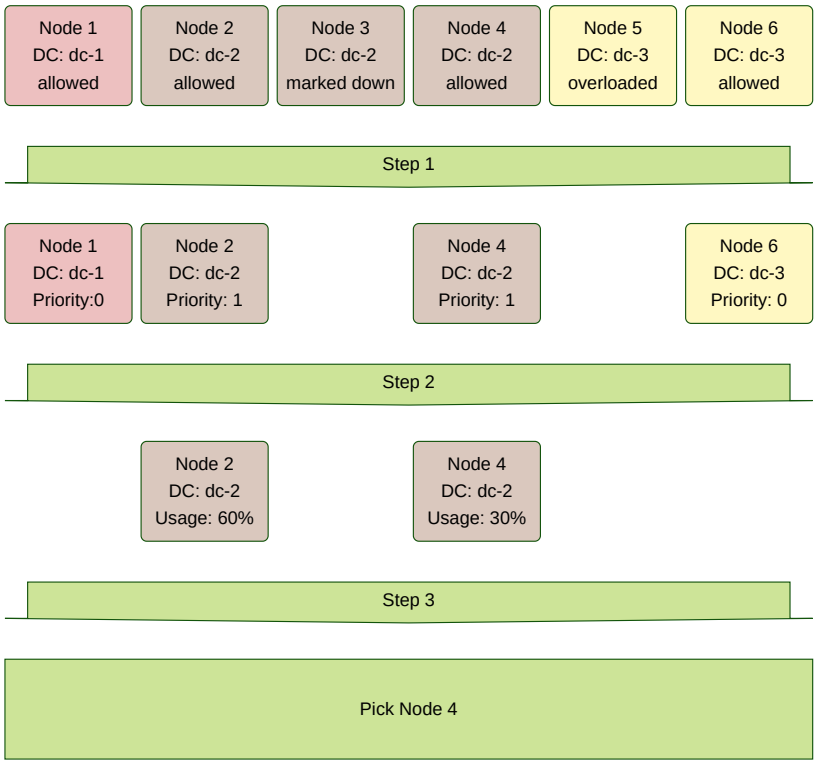
Одновременный запуск множества таблеток может создавать повышенную нагрузку на узел. Поэтому максимальное число одновременно запускаемых таблеток на одном узле ограничено значением `max_tablets_scheduled` из конфигурации. При этом, если один из узлов упирается в это ограничение, Hive останавливает запуск новых таблеток и на других узлах тоже, чтобы это не повлияло на равномерность распределения. Это поведение можно регулировать с помощью параметра `boot_strategy`.

Выбор узла

Существуют строгие ограничения на то, на каких узлах разрешено запускать таблетку: не каждый узел может запускать каждый тип таблеток; таблетки некоторой базы данных могут быть запущены только на узлах этой базы данных. Дополнительно при **перевозе** таблеток не рассматриваются перегруженные узлы.

1. Из всех подходящих узлов отбираются узлы с максимальным приоритетом. Приоритет определяется исходя из датацентра, в котором находится узел. Явно указать приоритеты датацентров можно в подсекции `default_tablet_preference` в конфигурации. Для координаторов и медиаторов приоритеты определяются динамически так, чтобы по возможности поддерживать их в одном датацентре. Дополнительно, если таблетка завершает работу с ошибкой на некотором узле, на следующий запуск этой таблетки приоритет этого узла понижается.
2. Для узлов с максимальным приоритетом считается целевая метрика, которая почти совпадает с метрикой Node usage. Она отличается тем, что учитываются только те ресурсы, которые потребляет данная таблетка, а также наличием штрафа за количество таблеток того же объекта схемы.
3. Наконец, на основе этой метрики делается выбор узла. Алгоритм выбора на этом этапе определяется параметром `node_select_strategy` в конфигурации. По умолчанию выбирается случайный узел из 7% узлов с минимальным значением метрики.

Ниже приведена иллюстрация процесса.



Процесс запуска

На каждом узле запущен сервис Local, отвечающий за взаимодействие с Hive. При запуске таблетки Hive отправляет в Local нужного узла команду старта таблетки, содержащую всю необходимую для запуска информацию: TabletID, поколение, историю каналов и режим запуска (лидер или подписчик). После того, как таблетка запускается, Local сообщает об этом в Hive. С этого момента таблетка считается запущенной с точки зрения Hive и остаётся такой, пока:

- Local не сообщит об остановке её работы;
- связь с Local не будет нарушена.

В этих ситуациях вновь начнётся процесс запуска уже для следующего поколения таблетки.

Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

DataShard: блокировки и видимость изменений в транзакциях

YQL запросам может требоваться прочитать данные, которые были записаны предыдущими запросами в рамках той же транзакции, ещё до коммита транзакции. В этом случае YDB необходимо поддерживать консистентную видимость данных с учётом сделанных изменений. Для реализации этого сценария DataShard поддерживает запись незакомиченных изменений в рамках транзакции, а затем включает эти данные в последующих чтениях в рамках той же транзакции. Также он позволяет атомарно коммитить все накопленные изменения с учётом ограничений изоляции serializable.

Для реализации этой возможности DataShard использует поддержку транзакций с большим количеством изменений в нижестоящей LocalDB. Эти транзакции не ограничены размером одного сообщения между акторами в распределённой системе.

Высокоуровневая схема работы

YQL транзакции могут разбиваться на несколько «фаз» в Query Processor, используя результат одной фазы в качестве входа для последующих фаз. Это происходит либо для «сложных» транзакций, состоящих из нескольких подзапросов, либо «интерактивных», где клиент начинает транзакцию и выполняет запросы без коммита. Например, если YQL запрос содержит `JOIN`, первая фаза может прочитать левую таблицу, а затем во второй фазе использовать этот результат для выполнения точечных чтений по правой таблице.

Для читающих запросов QP использует глобальный MVCC снимок для обеспечения консистентности между подзапросами. Для пишуших запросов нужно также убедиться во время коммита, что serializable изоляция не нарушается. Это обеспечивается с помощью Оптимистического управления параллелизмом, где чтения добавляют оптимистические блокировки для прочитанных диапазонов, а запись в других транзакциях «ломает» эти блокировки во время их коммита. Транзакция может успешно закоммититься только если ни одна из её блокировок не была сломана другой транзакцией. В противном случае транзакция завершится с ошибкой «transaction locks invalidated».

Транзакция может сделать несколько чтений используя временные метки чтения, пока другие транзакции параллельно пишут в те же таблицы и шарды. Это может быть как один глобальный MVCC снимок, так и несколько временных меток, разные для каждого чтения. Когда транзакция коммитится ей назначается единое время коммита в глобальном порядке выполнения. Если все чтения можно повторить в точке времени коммита, без изменения видимых результатов, такая транзакция могла бы целиком выполниться в точке с временем коммита. Не сломанная оптимистическая блокировка показывает транзакции, что эти чтения можно переместить в более позднюю точку с временем коммита.

Незакомиченные изменения немногим отличаются от чтений с точки зрения сериализации. Если DataShard может запомнить эти изменения и затем «переместить» их в финальную точку с временем коммита, такая транзакция может быть закоммичена, иначе она должна быть отменена. В отличии от блокировок чтения (которые хранятся в памяти), незакомиченные изменения персистентны, и их нужно правильно удалять когда в них больше нет необходимости.

Как блокировки используются во время чтений

Подготавливаемым на DataShard'ах операциям обычно назначают глобально уникальный 64-битный `TxId`. Они выделяются достаточно большими батчами с помощью глобальных таблеток TxAllocator. Когда QP начинает первое чтение в многофазной транзакции, он также использует этот `TxId` в качестве идентификатора блокировки `LockTxId`, который в дальнейшем используется во всех запросах в той же YQL транзакции. DataShard добавляет новые блокировки, если в операции указан ненулевой `LockTxId`:

- См. поле LockTxId в читающих `TEvRead` запросах
- См. поле LockTxId в пишущих `TEvWrite` запросах
- См. поле LockTxId в сообщениях `TDataTransaction` (используется для кодирования транзакций, которые работают с данными таблиц)

Также используется поле `LockNodeId`, в котором указывается идентификатор ноды с источником блокировки. DataShard использует это поле для подписки на статус блокировки и зачистки блокировок, которые перестают использоваться.

`LockTxId` — это просто уникальное число, которое используется множеством операций в одной и той же YQL транзакции, в то время как `TxId` уникальный для каждой операции на одном DataShard'е. Использование первого `TxId` в качестве `LockTxId` не обязательно, но т.к. QP уже получил глобально уникальное число с выделением `TxId`, а `LockTxId` не пересекается с `TxId` по использованию, это экономит аллокацию.

Таблица блокировок (см. datashard_locks.h и datashard_locks.cpp) индексирует блокировки по диапазонам первичного ключа через дерево регионов, позволяя находить и «ломать» их по точечным ключам. В простом случае читающие операции добавляют диапазон с помощью метода SetLock, а пишущие операции ломают другие блокировки по записанному ключу используя метод BreakLocks.

Когда блокировка добавляется впервые, ей назначается монотонно возрастающий `Counter` в текущем `Generation` таблетки (см. сообщение TLock). Также строка с этими числами появляется в виртуальной таблице `/sys/locks` (сама таблица больше не используется). Информация о взятых блокировках возвращается в результирующих сообщениях (например см. TEvReadResult).

В успешном сценарии, когда пока блокировка существует и не сломана, поля `Generation` и `Counter` не меняются.
В неуспешных сценариях ранее установленные блокировки инвалидируются. Например, в случае инвалидации блокировки при рестарте изменяется значение `Generation`, а `Counter` может менять при явном статусе ошибки или при инвалидации и пересоздании блокировки в том же поколении.

Первые `Generation` и `Counter` для каждого шарда запоминаются в Query Processor. Если во время жизни транзакции они изменились, это сигнализируют о неконсистентности и нарушении serializable изоляции. Пишущие транзакции, либо транзакции которые не используют глобальный MVCC снимок для эффективности, проверяют значения `Generation` и `Counter` блокировок при финальный коммите. Коммит может завершиться успехом только если все значения всех блокировок совпадают.

При использовании глобального MVCC снимка все чтения с этим снимком гарантированно консистентны. Тем не менее, они происходят со взятием блокировок на случай, если транзакция в дальнейшем окажется пишущей. DataShard при этом дополнительно проверяет наличие конфликтующих данных, закомиченных поверх читаемого снимка. При обнаружении конфликта, чтение не завершается с ошибкой, однако поле `Counter` блокировки выставляется равным ErrorAlreadyBroken. Это сигнализирует о том, что блокировку взять не удалось и записать что-то в этой транзакции в дальнейшем не получится. Если окажется, что вся транзакция целиком выполняла только чтения, то она завершится успешно. Однако, при любой попытке что-либо записать такая транзакция сразу же завершится с ошибкой, так как коммит будет гарантированно не возможен.

Как блокировки используются для записи изменений

Когда Query Processor необходимо сделать незакомиченные изменения в рамках YQL транзакции, в операции записи на DataShard'е также указывается ненулевой `LockTxId`. Помимо установки блокировок DataShard использует этот `LockTxId` в качестве `TxId` для записи незакомиченных изменений. Эти изменения будут храниться пока блокировка остаётся валидной. Блокировки с незакомиченными изменениями называются пишущими блокировками (write locks), а информация по ним сохраняется в специальных таблицах (см. таблицу Locks и связанные таблицы ниже), что позволяет им переживать рестарты DataShard'ов.

При использовании операций записи незакомиченных изменений есть несколько ограничений:

- Операция должна выполняться в режиме immediate, таким образом нельзя записать незакомиченные изменения распределённой транзакцией. Вместо этого изменения пишутся независимо на каждый шард.
- В операции должен быть указан `LockNodeId` с нодой, через которую DataShard подписывается на статус блокировки и автоматически удаляет изменения, если блокировка пропадает. Например, блокировка пропадает, если транзакция отменяется нештатно или нода с состоянием транзакции не отвечает.
- У операции должен быть указан правильный MVCC снимок, который используется для чтений и поиска конфликтов. Ожидается, что в рамках одной YQL транзакции используется один и тот же MVCC снимок во всех операциях чтения и записи.
- Существующая блокировка должна быть валидна и не сломана. В противном случае для указанного `LockTxId` не должно быть уже имеющихся нескомпакченных данных в LocalDB. Это защищает от нештатных ситуаций, когда транзакция отменяется на шарде (например, на основе статуса блокировки), а QP пытается продолжить в ней писать.

При чтениях YQL транзакция должна указывать тот же `LockTxId`. При этом чтения через LocalDB будут использовать особую карту транзакций, в которой `LockTxId` как-будто уже закоммичен. Это позволяет транзакции видеть собственные изменения, но не изменения других незакомиченных транзакций. В карте транзакций добавляется особая запись `[LockTxId] => v{min}`, позволяя видеть собственные изменения независимо от значения конкретного MVCC снепшота.

Незакомиченные записи требуют дополнительного анализа конфликтов (см. CheckWriteConflicts в реализации minikql engine host). Когда несколько незакомиченных транзакций пытаются писать в один и тот же ключ, DataShard должен убедиться, что в случае коммита соответствующие им конфликтующие блокировки будут сломаны. Объекты для отслеживания транзакций (например TLockedWriteTxObserver) используются для обнаружения таких конфликтов. LocalDB вызывает соответствующие методы этих объектов перед пропуском или применением изменений. Перед записью DataShard выполняет чтение в специальном режиме для поиска всех незакомиченных изменений по ключу. В случае обнаружения конфликта они добавляются в граф конфликтов между блокировками. Таким образом, каждая блокировка знает какие блокировки должны быть сломаны в случае коммита.

Если на шарде есть незакомиченные изменения, то чтения также должны дополнительно проверять конфликты. Это необходимо, так как чтения должны быть не только консистентными в рамках текущей транзакции, но и учитывать согласованность состояния с будущей точкой времени коммита. Здесь также используются объекты для отслеживания, чтобы собирать списки незакомиченных транзакций, пропущенных во время чтения. На их основе добавляются связи в графе между пишущими и читающими блокировками.

Особой обработки требует ситуация, когда транзакция делает чтение с учётом собственных незакомиченных изменений, которые были поверх ранее закомиченных изменений, но сделанных уже после получения MVCC снимка в транзакции. Пример подобной ситуации:

1. Ключ K изначально имеет `A = 1` в версии `v4000/100`. Два числа здесь — это `Step` (4000) и `TxId` (100) времени коммита.
2. Tx1 открывается и получает MVCC снимок `v5000/max`.
3. Tx2 коммитит слепую запись `UPSERT` с `B = 2` в версии `v6000/102`.
4. Tx1 делает слепую запись `UPSERT` с `C = 3` и `TxId = 101`.
5. В этот момент Tx1 ещё могла бы закоммититься успешно, так как она не читала ключ K и изменения с `C = 3` всё ещё может переместиться в будущее время коммита.
6. Tx1 выполняет чтение по ключу K, которое выполняется с использованием снимка `v5000/max`:
    - На время чтения будет использоваться временная запись `[101] => v{min}` в карте транзакций;
    - Итерация спозиционируется на первой дельте с изменением `C = 3` (т.к. `v{min} <= v5000/max`), которое будет добавлено в состояние строки;
    - Все прочие закомиченные дельты будут также добавлены, т.е. состояние строки будет включать `B = 2` которое в этот момент уже закоммичено;
    - Однако, здесь возникает конфликт, т.к. `B = 2` было закоммичено уже после MVCC снимка (`v6000/102 > v5000/max`);
    - Это обнаружится в методе OnApplyCommitted, который вызывает CheckReadConflict;
    - Так как подобное состояние строки не соответствует изоляции, блокировка транзакции будет автоматически сломана, а также будет взведён флаг неконсистентности чтения.
7. С ошибкой завершится не только данное чтение, но также Tx1 не сможет быть закоммичена, т.к. произошло нарушение изоляции.
8. Клиентское приложение получит ошибку «transaction locks invalidated» и может принять решение о повторе транзакции с начала.

Взаимодействие с накоплением изменений

У DataShard'ов есть так называемые накопители изменений (change collectors), которые логируют происходящие изменения и потоково отправляют их в другие подсистемы. В частности, это используется для поддержки Change Data Capture и Асинхронных вторичных индексов. В зависимости от режима работы, накопителям изменений может требоваться знать предыдущее состояние строки, и для этого они выполняют чтение строки перед её записью.

Накопителям изменений также нужно обрабатывать запись незакомиченных изменений. Однако эти записи нельзя отправлять в другие системы до коммита. Также порядок отправки изменений должен совпадать с финальным порядком их применения. Для поддержки этих требований изменения сначала аккумулируются в отдельной таблице LockChangeRecords, а с случае успешного коммита транзакции они атомарно добавляются в выходной поток с помощью добавления одной строки в таблице ChangeRecordCommits. Таким образом, обработка изменений происходит постепенно, что избавляет от дорогостоящей массовой обработки во время коммита. Объем накапливаемых изменений не ограничен.

Взаимодействие с распределёнными транзакциями

Когда распределённая транзакция начинает выполнение, она валидирует блокировки на текущем шарде и рассылает результат на других участников транзакции. Если все результаты получены и оказались успешными, то тело транзакции

может продолжить выполнение и применить эффекты. Другими словами, если все чтения со всех участвующих шардов удаётся переместить на время коммита без нарушения гарантий изоляции, то транзакция может применить все накопленные изменения. В противном случае тело транзакции не выполняется, возвращается соответствующий код ошибки, а все накопленные изменения отменяются. Результаты валидации блокировок запоминаются персистентно. Корректность выполнения обеспечивается с помощью анализа конфликтов между транзакциями.

В отличии от обычных транзакций, коммит блокировки с незакомиченными изменениями может затрагивать произвольные ключи, которые не хранятся в памяти DataShard'а. Соответственно, проанализировать конфликты по пересечению ключей не представляется возможным. После валидации пишущей блокировки, она «замораживается» и больше не может сломаться. Если конфликтующая транзакция попытается сломать замороженную блокировку, она вместо этого ставится на паузу и дожидается пока транзакция с замороженной блокировкой завершится. DataShard следит за тем, коммит каких блокировок может сломать другие блокировки. Для защиты от тупиковой ситуации он также проверяет может ли коммит транзакции А изменить результат валидации транзакции Б и наоборот. Если транзакция А должна выполниться до транзакции Б в глобальном порядке транзакций, DataShard не будет начинать валидацию транзакции Б пока транзакция А не завершится. Пока транзакции не конфликтуют они могут выполняться в любом порядке.

Коммит изменений

Для коммита ранее записанных изменений, Query Processor запускает на шардах транзакцию, которая коммитит ранее захваченные блокировки. У такой транзакции не указывается `LockTxId`, так как такой коммит не захватывает новые блокировки. В списке Locks перечисляются все ранее установленные блокировки, а в поле `Op` указывается Commit. Сама транзакция коммита на шардах может быть либо immediate (если затрагивает только один шард), либо подготовлена как распределённая транзакция на нескольких участниках.

Для поддержки распределённых транзакций все шарды, которые проверяют блокировки, включаются в список SendingShards, а все шарды, применяющие эффекты, включаются в список ReceivingShards. Во время валидации отправляющие шарды подготавливают персистентные readset'ы и отправляют их на все принимающие шарды. Принимающие же шарды после валидации ждут получения readset'ов перед тем как начинать выполнение транзакции. Во время выполнения, в случае если все результаты валидации были успешными, блокировки коммитятся через вызов KqpCommitLocks. В противном случае, тело транзакции не выполняется, а блокировка удаляется вместе с отменой всех незакомиченных изменений через вызов KqpEraseLocks. Так как в случае ошибки коммита изменения отменяются, отдельно коммит ретраить нельзя.

> **Примечание**
>
> Все незакомиченные изменения с одинаковым `LockTxId` должны быть включены в транзакцию коммита, и транзакцию нельзя коммитить частично. Например, если транзакция сделала несколько записей, и на одной из записей произошла ошибка (например в случае кратковременной недоступности шарда, когда нельзя однозначно понять была ли запись произведена), то будет некорректно пропустить эту запись и закоммитить транзакцию частично. Если в будущем произойдёт мерж шардов, то несовпадение статуса транзакции может приводить к аномалиям.

В транзакции коммита также могут быть дополнительные изменения, которые атомарно применяются сразу после коммита блокировки. Для уменьшения задержек QP старается аккумулировать изменения в памяти либо до момента их коммита, либо до необходимости чтения этих изменений в той же транзакции. Далее QP отправляет накопленные изменения вместе с транзацией коммита. В случае, если удаётся накопить все изменения до момента коммита, незакомиченных изменений не происходит, а блокировки при необходимости берутся только на чтение.

При необходимости явной отмены изменений YQL транзакция запускает пустую транзакцию на шардах с указаем Rollback в качестве операции блокировки. Если по какой-то причине этого не происходит, при разрушении TLockHandle все подписанные даташарды автоматически чистят блокировки через внутреннюю транзакцию TxRemoveLock. Явная отмена предпочтительна, т.к. незакомиченные изменения потребляют ресурсы и в случае асинхронной очистки через `TLockHandle` нет гарантии, что эти ресурсы освободятся до того, как новые транзакции попытаются записать ещё больше изменений.

Ограничения

Наличие незакомиченных изменений приводит к тому, что всем новым попыткам записи на тот же шард приходится дополнительно проверять конфликты, из-за чего запись перестаёт быть «слепой». Чтение перед каждой записью делает их дороже, увеличивает время выполнения транзакций и уменьшает пропускную способность. Для максимальной производительности рекомендуется выполнять транзакции таким образом, что сначала выполняются все чтения, а затем небольшое количество слепых записей. В этом случае незакомиченные изменения не используются и производительность шардов остаётся оптимальной.

Из-за ограничений LocalDB даташарды также вынуждены гарантировать, что не будет образовываться слишком много открытых транзакций. Сейчас таблица блокировок ограничена примерно 10 тысячами записей, что включает в себя пишущие блокировки. Даташарды также вынуждены подсчитывать количество уже имеющихся незакомиченных изменений перед каждой новой записью, что реализовано через подсчёт пропуском в объекте для отслеживания транзакций. В случае превышения лимита кидается исключение TLockedWriteLimitException.

Персистентные блокировки переживают рестарты даташардов и восстанавливают своё последнее состояние. В этом механизме поддерживается возможность сохранения диапазонов, но на практике она не используются, так как даташард был бы вынужден записывать новые диапазоны во время каждого чтения, что слишком дорог. Вместо этого читающие блокировки, ставшие персистентными из-за наличия записи, восставливаются с диапазоном «весь шард». Это лучше, чем инвалидация на любом рестарте, но конфликтует с любой записью и увеличивает риск ошибки «transaction locks invalidated» (TLI).

> **Хотите присоединиться к команде разработки YDB?**
>
> Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

DataShard: распределённые транзакции

На основе идей Calvin в YDB реализованы распределённые транзакции. Они состоят из множества произвольных (не обязательно детерминированных) операций, выполняемых над заранее известным набором участников, например DataShard'ами. В процессе выполнения транзакции, она подготавливается на каждом из участников, и получает позицию (временную метку) в глобальном порядке выполнения через одну из таблеток координатора в базе данных. Каждый участник получает часть потока транзакций с их участием, с сохранением их относительного порядка выполнения. Далее каждый участник выполняет свою часть распределённых транзакций в соответствии с этим порядком, и несмотря на то, что разные участники могут выполнять эти транзакции с разной скоростью и не одновременно, с точки зрения временных меток они выполняются в одной и той же точке, учитывая все эффекты, произошедшие с более ранними временными метками.

В случае, когда выполнение транзакции зависит от состояния других участников, участники обмениваются данными между собой с помощью так называемых ReadSet'ов, представляющих из себя персистентное сообщение от одного участника другому, с семантикой доставки "как минимум один раз". В этом случае выполнение транзакции на каждом из участников делится на три дополнительные фазы:

1. **Фаза чтения**. Участник производит чтение, фиксацию и отправку данных, которые необходимы другим участникам. Сейчас на коммите QP транзакций (тип транзакции `TX_KIND_DATA` с заполненным полем `TDataTransaction.KqpTransaction` и типом `KQP_TX_TYPE_DATA`) в этой фазе происходит только проверка оптимистичных блокировок, если результат этой проверки нужен другим участникам. В более старых MiniKQL транзакциях (тип транзакции `TX_KIND_DATA` с заполненным полем `TDataTransaction.MiniKQL`) выполняется чтение и отправка произвольных данных. Другой пример, где происходит чтение в этой фазе - распределённая транзакция удаления строк по TTL, которая используется для фильтрации подходящих под условие строк и одновременного их удаления из основной таблицы и индексов.
2. **Фаза ожидания**. Участник ждёт получения необходимых ему данных от других участников.
3. **Фаза выполнения**. Участник на основе локальных и полученных с других участников данных принимает решение о выполнении, либо отмене транзакции, формирует и применяет эффекты в соответствии с телом транзакции. Тело транзакции написано таким образом, что участники принимают решение на одинаковых данных, и либо все принимают решение о выполнении, либо все принимают решение об отмене транзакции.

Для эффективности участники могут выполнять операции в другом порядке, но важно, что с точки зрения наблюдаемых эффектов (других операций чтения и записи) подобного изменения порядка не было видно. Если все транзакции в базе данных выполняются по данной схеме, то это соответствует уровню изоляции strict serializable. На практике одношардовые транзакции не проходят через механизм координации, и каждый участник самостоятельно выбирает в какой точке глобального порядка выполнить транзакцию, что из-за разной скорости получения потока транзакций и их выполнения снижает уровень изоляции до serializable.

Начиная с версии 24.1 в YDB появилась поддержка "волатильных" распределённых транзакций. Основное отличие "волатильных" транзакций в том, что все участники (в том числе координатор), могут хранить информацию о транзакциях в памяти (и терять любое упоминание о них в случае рестарта) до момента их выполнения и записи эффектов в персистентное хранилище, в том числе отказываться от выполнения любой такой транзакции с гарантированной отменой на других участниках. Это позволяет исключить ожидание ответа от персистентного хранилища до выполнения распределённой транзакции на участниках и уменьшает время отклика.

В текущих версиях YDB, в процессе выполнения пользовательских YQL транзакций, в виде распределённых транзакций выполняется только финальный коммит, применяющий эффекты в read-write транзакциях. Запросы до коммита YQL транзакции выполняются как одношардовые с использованием оптимистичных блокировок, а консистентность обеспечивается с помощью глобального MVCC снимка.

Базовый протокол выполнения распределённых транзакций

Типы операций и участников, которые можно выполнять в виде распределённых транзакций, в YDB постепенно расширяются. При этом любые распределённые транзакции выполняются по примерно одинаковому базовому протоколу, а существенные отличия есть разве что у схемных транзакций, к которым сейчас предъявляются дополнительные требования идемпотентности.

В качестве примеров акторов, которые выполняют оркестрацию распределённых транзакций, можно привести:

- TKqpDataExecutor выполняет DML запросы, в том числе распределённые коммиты
- SchemeShard выполняет распределённые схемные транзакции
- TDistEraser выполняет распределённые транзакции удаления строк в таблицах с индексами для целей TTL

Протокол выполнения распределённых транзакций в YDB немного напоминает двухфазный коммит, хотя и не является таковым. Общую схему работы актора, который оркестрирует выполнение распределённой транзакции, можно разделить на следующие фазы:

1. **Определение списка участников**. Выбирается набор конкретных шардов (`TabletId`), которые будут участвовать в её выполнении. Например, таблица может состоять из множества шардов (таблеток `DataShard` с уникальными `TabletId`), но если для конкретной транзакции набор затрагиваемых ключей попадает на один шард, такая транзакция будет одношардовой с единственным участником. В дальнейшем этот набор участников не может быть изменён.
2. **Фаза подготовки**. На участников отправляется специальное сообщение (обычно оно называется `TEvProposeTransaction`, в случае DataShard таблеток с недавнего времени появился специализированный вариант `TEvWrite`), в котором указывается уникальный в рамках кластера `TxId` (идентификатор транзакции), и которое описывает необходимые операции и их параметры (тело транзакции). Участники валидируют возможность выполнения транзакции, выбирают для неё допустимый диапазон позиций в глобальном порядке (процедура выбора `MinStep` и `MaxStep` описана ниже), в случае успеха запоминают транзакцию и отвечают статусом `PREPARED`.
   - Если транзакция одношардовая, в сообщении указывается режим "немедленного выполнения" (`Immediate`). Шард выполняет такую транзакцию как можно скорее (с учётом требований консистентности), а вместо `PREPARED` отвечает результатом выполнения. Фаза планирования при этом пропускается. В некоторых специализированных вариантах одношардовых операций (например `TEvUploadRowsRequest` для реализации `BulkUpsert`) допускается также отсутствие `TxId`.
   - Тело персистентной транзакции надёжно сохраняется в локальной базе и участник гарантирует её выполнение в будущем, В некоторых случаях (например "слепой" `UPSERT` в несколько шардов) участник даже должен гарантировать успешность выполнения.
   - Тело "волатильной" транзакции (`Volatile`) запоминается в памяти и участник сразу отвечает `PREPARED`. При этом выполнение или успешное выполнение этой транзакции не гарантируется.
   - Процесс переходит к следующей фазе только после того, как будут получены ответы от всех участников.
   - В общем случае (за исключением схемных транзакций), сообщения `TEvProposeTransaction` нельзя отправлять на участников повторно.

3. **Фаза планирования**. После того, как от всех участников был получен ответ `PREPARED`, на основе ответов выбирается общий `MinStep` и `MaxStep`, а также выбирается координатор, через которого будет происходить планирование транзакции. На координатор отправляется сообщение `TEvTxProxy::TEvProposeTransaction` с указанием `TxId` транзакции и списком её участников.
    - Выполнение транзакции возможно только между участниками одной базы данных. Каждый участник в ответе из фазы подготовки сообщает свои ProcessingParams, и в случае если все участники в одной базе данных, наборы координаторов и медиаторов будут идентичны.
    - Выбор координатора происходит на основе полученных `ProcessingParams` по историческим причинам, т.к. запросы выполнялись без указания базы данных, и узнать список координаторов можно было только от самих участников.
    - В случае, если сообщение `TEvTxProxy::TEvProposeTransaction` отправляется на координатор повторно, транзакция может быть запланирована на несколько временных меток одновременно. Это не является проблемой (и является нормой для схемных транзакций), в этом случае транзакция будет выполнена в минимальной временной метке, а остальные временные метки будут проигнорированы участниками, т.к. к этому моменту транзакция уже выполнена и более не существует.
4. **Фаза выполнения**. В этой фазе актор ожидает ответы от координатора и участников, собирая общий результат выполнения транзакции.
    - В некоторых случаях (временная потеря связи, рестарт таблеток) актор может восстанавливать подключение к участникам и продолжать ожидание результата, если к этому моменту транзакция ещё не была выполнена.
    - В случае потери связи и невозможности получить результат выполнения от какого-либо участника, пишущая транзакция завершается со статусом `UNDETERMINED`.

Фаза подготовки в таблетке DataShard

Распределённые транзакции в таблетке DataShard начинаются с фазы подготовки, где транзакция предлагается на выполнение одним из следующих сообщений:

- TEvDataShard::TEvProposeTransaction точка входа для транзакций разных типов
- TDataEvents::TEvWrite специальная точка входа для операций записи данных и коммитов

Если в сообщении не указан режим работы `Immediate` (немедленного выполнения), то оно запускает фазу подготовки распределённой транзакции. Тело транзакции валидируется на возможность выполнения (например, CheckDataTxUnit в случае обычных транзакций с подпрограммами), при этом выбирается допустимый диапазон планирования:

- `MinStep` выбирается на основе текущего медиаторного/настенного времени
- `MaxStep` выбирается с добавлением таймаута планирования, который сейчас равен 30 секундам для обычных транзакций

После валидации и выбора параметров планирования, транзакция сохраняется на диск (в случае обычных транзакций), либо в памяти (в случае волатильных транзакций), шард отвечает статусом `PREPARED` и начинает ожидание плана с указанием `PlanStep` для данного `TxId`. Таймаут планирования используется на случай нештатного завершения актора, запускающего распределённую транзакцию. Шард не может достоверно выяснить, успел ли актор запустить фазу планирования до своего завершения, однако после успешной валидации транзакции должен гарантировать её потенциальное выполнение без возможности отмены по инициативе шарда (за исключением волатильных транзакций). Это блокирует многие конфликтующие операции (например, изменение схемы или партиционирования), и необходим таймаут, после которого гарантируется отсутствие возможности запланировать эту транзакцию. Когда медиаторное время превышает `MaxStep` без появления плана, гарантируется, что транзакция не сможет быть запланирована, и такая транзакция может быть безопасно удалена.

За хранение транзакций в памяти и на диске отвечает класс TTransQueue. В случае персистентных транзакций основная информация о транзакциях сохраняется в таблице TxMain и загружается целиком на старте шарда. Потенциально большое тело транзакции сохраняется в таблице TxDetails и может не задерживаться в памяти на время ожидания. Тело транзакции загружается в память повторно перед добавлением в очередь на выполнение и начала анализа конфликтов с другими транзакциями.

В отличие от персистентных транзакций, волатильные распределённые транзакции сохраняются исключительно в памяти, в текущей реализации исчезая вместе с рестартом шарда (в будущем может появиться возможность их миграции во время штатных рестартов). Протокол предусматривает, что волатильные транзакции могут быть отменены в любой момент по инициативе любого участвующего шарда до их окончательного выполнения на этом шарде. Например, шарды пользуются этим для быстрой отмены транзакций с началом изменения партиционирования, или при появлении схемной операции, не дожидаясь окончания таймаута планирования.

В теле распределённой транзакции должно быть достаточно информации о других участниках этой транзакции, для формирования и отправки ReadSet'ов, а также для их правильного ожидания. В используемых сейчас QP транзакциях это так или иначе связано с проверкой блокировок и передаётся через структуру TKqpLocks, которая заполняется актором `TKqpDataExecutor`. Там указываются наборы шардов:

- `SendingShards` - шарды из этого множества отправляют ReadSet'ы на все шарды из множества `ReceivingShards`
- `ReceivingShards` - шарды из этого множества ожидают получение ReadSet'ов от всех шардов из множества `SendingShards`

В случае волатильных транзакций предполагается, что все шарды указываются в `SendingShards` (любой шард может отменить транзакцию и должен сообщить об этом на другие шарды), а шарды, применяющие эффекты, обязательно указываются в `ReceivingShards` (успешность применения эффектов зависит от других шардов). При этом через ReadSet'ы перадаются сериализованные сообщения TReadSetData с указанием единственного поля `Decision`.

В качестве обратного примера, когда ReadSet'ы не используются совсем, можно привести персистентную транзакцию слепой записи в несколько шардов. В этом случае, после планирования, транзакция не может быть отменена, и шарды обязаны гарантировать будущую успешность этой записи ещё в фазе подготовки.

Фаза планирования

После ответов `PREPARED` от всех участников, оркестрирующий актор выбирает максимум среди `MinStep`, минимум среди `MaxStep`, выбирает конкретный экземпляр координатора, и отправляет сообщение `TEvTxProxy::TEvProposeTransaction` с указанием `TxId` и списком участников (исторически для каждого участника также указывается тип операции над шардов - чтение или запись, но на практике это никак не используется). Соответствующий координатор выбирает ближайший подходящий `PlanStep` (шаг планирования) и добавляет в него указанный `TxId` со списком участников. Для персистентных транзакций это соответствие надёжно сохраняется на диске, обычно выбирая шаг планирования не чаще, чем каждые 10 миллисекунд. Для волатильных транзакций выбирается ближайший `PlanStep` среди тех, которые надёжно зарезервированы за текущим поколением координатора, а очередной шаг планирования (при наличии транзакций, либо при наличии запросов на продвижение медиаторного времени) выделяется каждую миллисекунду, и происходит без какой-либо записи на диск.

Когда очередной `PlanStep` оказывается запланирован (в каждом из которых может быть ноль и более транзакций), участвующие в нём шарды (и как результат транзакции) распределяются по медиаторам. Сейчас распределение происходит через хеширование идентификаторов таблеток по модулю от количества медиаторов. Каждому медиатору отправляется только соответствующая ему часть плана, в виде потока шагов с сохранением порядка. Часть плана может оказаться пустой, даже если шаг планирования не был пустым. Если происходит рестарт медиатора или если с ним временно теряется связь, координатор автоматически переподключается и повторяет все шаги, которые ещё не были подтверждены.

Для персистентных транзакций переотправка шагов планирования гарантирует, что транзакция будет доставлена до медиатора как-минимум один раз. Волатильные транзакции хранятся только в памяти, и могут пропадать в случае сбоя координатора. По этой причине начиная с версии 24.1 изменены гарантии доставки, повторная отправка волатильных шагов планирования не гарантируется. Это также включает в себя пустые шаги планирования, координаторы оставляют в памяти и переотправляют только последний из них.

Для уменьшения ошибок в случае штатных рестартов, координатор запускает специальный актор состояния, который хранит состояние координатора в памяти, и продолжает какое-то время работать даже после того как экземпляр координатора останавливается. Адрес этого актора надёжно сохраняется в сторадже после завершения инициализации и до обработки новых запросов, и автоматически доступен более поздним поколениям. Новое поколение координатора связывается с последним известным актором и переносит состояние в памяти, включая списки запланированных волатильных транзакций, как часть своей инициализации. Актор состояния также используется для переноса резерва шагов для планирования без записи на диск, и также позволяет переносить неиспользованный резерв в новое поколение координатора без дополнительных ожиданий.

Медиаторы получают потоки шагов планирования через сообщения `TEvTxCoordinator::TEvCoordinatorStep` от каждого из координаторов и объединяют их по совпадению `PlanStep`. Объединённые шаги планирования, которые меньше или равны минимуму среди последних полученных шагов от каждого координатора, считаются полными и направляются на участников через сообщения `TEvTxProcessing::TEvPlanStep`. При этом на каждого участника приходит сообщение с указанием `PlanStep`, а также список `TxId`, которые должны быть выполнены в этом `PlanStep`. Транзакции в рамках одного `PlanStep` упорядочиваются по увеличению `TxId`. Пары `(Step, TxId)` в дальнейшем используется как глобальная MVCC версия в системе.

Участники подтверждают получение шагов планирования (и их надёжную запись в случае персистентных транзакций) через отправку сообщения `TEvTxProcessing::TEvPlanStepAccepted` отправителю (в таблетку медиатора) и сообщений `TEvTxProcessing::TEvPlanStepAck` в таблетки координатора. После получения этих сообщений соответствующие шаги планирования считаются доставленными на участника, удаляются из памяти и из локальной базы координатора, и не будут повторно доставляться.

На основе сообщений `TEvTxProcessing::TEvPlanStepAccepted` медиаторы также запоминают, до какого `PlanStep` включительно все шаги планирования были доставлены до всех участников. Такой максимальный `PlanStep` называется медиаторным временем и распространяется на ноды с участниками по подпискам через сервис `TimeCast`. В общем случае медиаторное время показывает, что до соответствующего `PlanStep` все сообщения `TEvTxProcessing::TEvPlanStep` были получены и подтверждены участником, и участнику должно быть известно обо всех транзакциях до этого времени включительно. Полезность медиаторного времени прежде всего в том, что позволяет шардам узнавать о продвижении времени в системе, даже когда они не участвуют в распределённых транзакциях. Для эффективности шарды сейчас распределяются по нескольким timecast бакетам на каждом медиаторе, и время в бакете продвигается когда все участники в этом бакете подтверждают получение `TEvTxProcessing::TEvPlanStep`. Само медиаторное время предоставляется шардам в виде подписки (через отправку TEvRegisterTablet в сервис time cast на старте шарда) на атомарную переменную (адрес которой приходит в сообщении TEvRegisterTabletResult после успешной регистрации в сервисе time cast) и позволяет избегать дорогих веерных рассылок сообщений на простаивающие шарды.

DataShard обрабатывает сообщения `TEvTxProcessing::TEvPlanStep` в транзакции TTxPlanStep, при этом найденным по `TxId` транзакциям назначается соответствующий `Step` и в дальнейшем они по очереди добавляются в `Pipeline` через `PlanQueue` в рамках ограничений на количество одновременно работающих активных транзакций.

Фаза выполнения в таблетке DataShard

Шаг выполнения PlanQueue разрешает запуск следующей распределённой транзакции в соответствии с увеличением `(Step, TxId)` и после опциональной загрузки тела транзакции с диска (для персистентных транзакций) QP транзакция финализирует план выполнения и попадает в шаг выполнения BuildAndWaitDependencies. На этом шаге анализируются ключи, которые заявлены для чтения и записи в теле транзакции, на основе которых между транзакциями возникают зависимости. Например, если транзакция `A` пишет по ключу `K`, а более поздняя транзакция `B` читает по ключу `K`, то у транзакции `B` возникает зависимость на транзакцию `A` и транзакция `B` не может быть выполнена до завершения выполнения транзакции `A`. Транзакция выходит из шага `BuildAndWaitDependencies` только когда у неё не остаётся прямых зависимостей от других транзакций.

Перед выполнением персистентная QP транзакция выполняет фазу чтений с формированием исходящих OutReadSet'ов (шаг BuildKqpDataTxOutRS), куда входит в том числе проверка оптимистичных блокировок. Далее на шаге StoreAndSendOutRS исходящие OutReadSet'ы и лог проверки блокировок сохраняются на диск. Если блокировки содержали незакомиченные изменения, у них выставляется флаг `Frozen` для предотвращения их отмены до завершения транзакции, в противном случае корректность обеспечивается через правильный порядок выполнения конфликтующих транзакций. Операция, которая формирует исходящие OutReadSet'ы либо проверяет блокировки, добавляется в `Incomplete` множество: конфликтующие транзакции должны обеспечить корректность уже произошедших чтений (например, выбирая MVCC версию для новых пишущих операций, которая больше чем версия операции), но при этом новые чтения могут не блокироваться на ожидании завершения этой операции и продолжать читать с версией, которая меньше чем версия операции.

Наконец, персистентные QP транзакции подготавливают структуры данных для ожидания входящих InReadSet'ов (шаг PrepareKqpDataTxInRS) и переходят к ожиданию всех необходимых ReadSet'ов от других участников (шаг LoadAndWaitInRS). В некоторых случаях, например слепая запись в несколько шардов без проверки блокировок, может происходить без обмена ReadSet'ами, и в этом случае шаги по обработки ReadSet'ов ничего не делают и пропускаются. В качестве исключения для волатильных транзакций шаги работы с ReadSet'ами также ничего не делают и пропускаются.

Наконец, операция переходит к фазе выполнения (шаг ExecuteKqpDataTx), ещё раз проверяет локальные блокировки (на этот раз используя сохранённый ранее `AccessLog`), проверяет ReadSet'ы, полученные от других участников, и в случае успеха выполняет тело транзакции и завершается успешно. Если проверка блокировок (локальных или удалённых) не проходит успешно и обнаруживается конфликт, то операция завершается с ошибкой `ABORTED`.

Особенности выполнения волатильных транзакций

В отличие от персистентных транзакций, волатильные транзакции хранятся только в памяти, пропадают на рестартах шарда, но при этом обязаны гарантировать, что транзакция будет либо закоммичена на всех участниках, либо оперативно отменена на всех участниках в случае ошибки на любом их них. Их преимущество при этом, что они не разделяются на длительные фазы чтения, ожидания и выполнения, а атомарно выполняются в единственной фазе выполнения за 1RTT стораджа на пути от начала запроса до ответа клиентам, при этом они чаще всего не задерживаются в очереди выполнения и позволяют

увеличивать пропускную способность шардов. Благодаря тому, что любой шард имеет возможно отменить транзакцию в любой момент, это также уменьшает недоступность шардов во время изменения партиционирования или выполнения схемных операций.

В основе волатильных транзакций лежит поддержка персистентных незакомиченных изменений в локальной базе. На фазе выполнения все эффекты записываются как незакомиченные (при этом используется глобально уникальный `TxId` распределённой транзакции), с добавлением транзакции в VolatileTxManager в состоянии ожидания решения о коммите от всех участников. Успешный ответ отправляется клиенту только после того как эффекты надёжно записаны, и все участники приняли решение о коммите. Последующие чтения используют `TxMap` чтобы видеть ещё ожидающие изменения, но проверяют их статус с помощью `TxObserver`. Если оказывается, что результат чтения зависит от волатильных изменений в состоянии ожидания, то такая операция подписывается на принятие решения о коммите волатильной транзакции и в конечном итоге перезапускается (либо прочитав теперь уже закомиченные изменения, либо пропустив их после отмены).

Наличие ещё незакомиченных и не отменённых волатильных транзакций накладывает ограничения на выполнение слепых операций на шарде. Из-за того, что незакомиченные изменения в локальной базе должны обязательно коммититься в том же порядке, в котором они происходили по каждому ключу, а информация о ключах транзакции не остаётся в памяти после её выполнения, DataShard вынужден делать чтение по ключу перед каждой записью для поиска конфликтов. Эти конфликты могут возникать как из-за незакомиченных изменений в рамках оптимистичных блокировок (в этом случае такие оптимистичные блокировки ломаются), так и принадлежать ещё ожидающим волатильным транзакциям. Чтобы не блокировать запись на время ожидания, даже не волатильная операция может переключиться на волатильный коммит. Для этого при необходимости выделяется `GlobalTxId` (уникальный в рамках кластера идентификатор транзакции, необходимые в случае отсутствия `TxId` в запросе, например `BulkUpsert`), а запись изменений по конфликтующим ключам переходит на волатильный коммит. Изменения при этом записываются как незакомиченные с указанием `GlobalTxId`, а в `VolatileTxManager` транзакция добавляется без указания других участников, и сразу оказывается закомиченной. Такие транзакции не блокируют последующие чтения, и после того как нижестоящие транзакции закоммитятся или отменятся, эти изменения также будут закоммичены в локальной базе.

Для уменьшения задержек в продвижении очереди транзакций используется кеш конфликтов для ключей по которым возможна запись в распределённых транзакциях. Пока транзакция ожидает своей очереди, по ключам записи заранее происходит чтение и кеширование конфликтов через вызов RegisterDistributedWrites, а любые операции записи в дальнейшем поддерживают информацию по ключам в актуальном состоянии. В этом случае пишущие распределённые транзакции могут быстро получить списки конфликтов по ключу перед каждой записью, даже если нужных страниц с данными нет в оперативной памяти, и такие транзакции выполняются быстрее.

Стоит заметить, что на таблице может быть настроена отгрузка потока изменений (асинхронные индексы и/или CDC). Формирование потока изменений в этом случае ничем не отличается от незакомиченных изменений в транзакциях. Для волатильной транзакции происходит формирование незакомиченного потока изменений, который в дальнейшем либо атомарно добавляется в общий поток, либо удаляется. В зависимости от настроек, для формирования потока изменений может потребоваться чтение текущего состояния строки. Если предыдущее состояние строки зависит от других ожидающих волатильных транзакций, образуется зависимость и последующий рестарт транзакции как и с обычным чтением. Это может сильно замедлять очередь распределённых транзакций, что впрочем мало чем отличается от возникновения зависимостей между персистентными распределёнными транзакциями с проверкой блокировок.

Одновременно с записью эффектов волатильная транзакция формирует и пишет на диск OutReadSet'ы для других участников распределённой транзакции из множества `ReceivingShards`. А после того, как ещё незакомиченные изменения успешно записаны на диск эти ReadSet'ы отправляются другим участникам.

Шарды после рестарта забывают о существовании волатильных транзакций, которые не успели сохранить эффекты на диске, и никогда уже не отправят ReadSet'ы другим участникам по своей инициативе. Чтобы даже в этом случае оперативно получать решение, шарды после успешного выполнения тела транзакции заранее отправляют другим участникам специальное сообщение, которое информирует об их ожидании входящего ReadSet'а (это называется `ReadSet Expectation`). Перезапустившиеся шарды увидят запрос на ReadSet по транзакции, про которую они ничего не знают, и отправят специальный ReadSet без данных. Благодаря таким ReadSet'ам без данных все участники узнают об отмене транзакции и также отменят её.

Гарантии на волатильные транзакции

Как упоминалось ранее, волатильные транзакции хранятся только в памяти шардов, в том числе координаторов, и по этой причине могут завершиться неудачно по самым разным причинам. Важно показать, почему волатильная транзакция либо успешно коммитится на всех шардах, либо отменяется на всех шардах в случае ошибок, а ситуация при которой транзакции применилась лишь на части шардов - невозможна.

Возможные источники ошибок при выполнении транзакций:

1. На любом из шардов может произойти нештатный рестарт, в том числе в ситуации, когда новый экземпляр шарда запускается до полного завершения работы старого экземпляра. Так как транзакция хранится только в памяти, у нового экземпляра может не быть никакой информации по транзакциям из старого экземпляра, которые ещё не были записаны на диск. В случае успешного выполнения транзакции, от неё также может не оставаться никаких следов, кроме её эффектов.
2. На координаторе может произойти нештатный рестарт, при этом план транзакции может оказаться отправлен лишь на часть шардов.
3. Один из шардов может по любой причине завершить выполнение транзакции с ошибкой.

Важно гарантировать, что если любой из шардов успешно закоммитил первоначальные незакомиченные эффекты транзакции и получил `DECISION_COMMIT` от всех других участников, то такая транзакция гарантированно закоммитится на всех участниках и не может быть отменена. Также важно гарантировать, что если любой из участников завершает транзакцию с ошибкой (в том числе забывает про неё после рестарта), то такая транзакция никогда не сможет успешно закоммититься ни на одном из участников.

Это достигается следующим образом:

1. ReadSet'ы с `DECISION_COMMIT` отправляются другим участникам только после успешной записи незакомиченных изменений, описания транзакции и исходящих ReadSet'ов на диск. Таким образом сообщения с `DECISION_COMMIT` являются персистентными и продолжат отправляться на других участников до момента их подтверждения, независимо от того коммитится ли в итоге транзакция.
2. ReadSet'ы подтверждаются только после успешной записи чего-либо на диск, либо после подтверждения readonly lease на момент получения сообщения (актуально в случае получения более не актуальных ReadSet'ов). Таким образом невозможна ситуация, когда предыдущее (устаревшее) поколение таблетки подтверждает получение ReadSet'а для неизвестной транзакции, которая была подготовлена на более позднем поколении таблетки. Локальная база гарантирует, что новое поколение таблетки запускается после того как readonly lease на предыдущих поколениях истекает (при условии, что частота монотонных часов не отличается больше чем в два раза). Успешная запись на диск также подтверждает актуальность текущего лидера.

- Для волатильных транзакций используется оптимизация, когда приходящие ReadSet'ы для них не записываются на диск. Подтверждение о получении отправляется только после завершения коммита или отмены транзакции, когда итоговое состояние надёжно записано на диск.
- Таким образом, в случае, если все участники сформировали ReadSet'ы с `DECISION_COMMIT`, шарды гарантированно продолжат их получать, пока окончательно не закоммитят все эффекты транзакции и не отправят подтверждение.

3. Любые сообщения, связанные с отменой транзакции, отправляются только после записи этой отмены на диск. Это нужно на случай гонок между несколькими поколениями одного из участников. Если новое поколение успешно коммитит транзакцию, то более старое поколение (без доступа к диску и с истёкшей readonly lease) может получить `NO_DATA` на запрос о ReadSet'е (т.к. получение уже было подтверждено более новым поколением). Однако в этом случае более старое поколение не сможет записать отмену на диск, и не сможет ошибочно ответить ошибкой на закомиченную транзакцию.
4. Успешный ответ после получения всех `DECISION_COMMIT` можно отправлять не дожидаясь окончательного коммита транзакции, но требует чтобы её эффекты были записаны на диск и не потерялись в случае рестарта. Это позволяет отвечать на успешно выполненные транзакции за 1RTT стораджа, при этом результат выполнения транзакции остаётся стабильным:
   - В случае рестарта эффекты транзакции записаны на диск, и в худшем случае она восстановится в состоянии ожидания (`Waiting`), т.к. перед ответом мы как-минимум дождались записи этого состояния на диск.
   - С точки зрения MVCC эта транзакция уже выполнена, и любое новое чтение будет выполняться с версией, которая включает в себя изменения из этой транзакции. Любая новая попытка чтения этих изменений будет запускать ожидание решения по этой транзакции.
   - Неподтверждённые ReadSet'ы будут переотправлены, и т.к. мы ранее получили `DECISION_COMMIT` от всех участников, мы снова получим тот же успешный результат.
5. Пока транзакция находится в очереди, и особенно пока для неё неизвестен шаг планирования, входящие ReadSet'ы запоминаются в памяти, но по ним не отправляются ответы. Ответы отправляются уже после её выполнения, или если транзакция отменяются досрочно (с гарантией, что она не была и не будет записана на диск).

Волатильные транзакции, в отличие от персистентных, гарантируют стабильность видимости изменений на всех шардах, в которых участвовала распределённая транзакция. Например, для персистентных транзакций возможна следующая аномалия:

1. Многошардовая транзакция `Tx1` осуществляет слепую запись в два шарда, по ключам `x` и `y`.
2. Одношардовое чтение `Tx2` читает ключ `x` и видит значение записанное в `Tx1`.
3. Одношардовое чтение `Tx3` строго после завершения `Tx2` читает ключ `y` и не видит значение записанное в `Tx1`.

Данная аномалия возникает, если шард с ключом `x` быстро получает шаг планирования с транзацией `Tx1` и выполняет запись. Поступившее на него чтение из транзакции `Tx2` выберет MVCC версию, которая включает изменение по ключу `x`. Однако, шард с ключом `y` может находиться на ноде, которая немного отстаёт в продвижении медиаторного времени, и данный шард мог ещё не получать шаг планирования с транзакцией `Tx1` и ещё не знает, будет ли выполняться данная транзакция и когда. Чтение из транзакции `Tx3` в этом случае может выбрать MVCC версию, которая не включает в себя изменения по ключу `y`. Несмотря на то, что транзакция `Tx3` выполнялась строго после `Tx2`, с точки зрения глобального порядка транзакция `Tx3` выполнилась до `Tx2`.

С волатильными транзакциями подобная аномалия невозможна, т.к. для завершения чтения по ключу `x` шард должен получить `DECISION_COMMIT` от шарда с ключом `y`. Это означает, что транзакция выполнена и записана на шарде с ключом `y`, и последующее чтение в `Tx3` обязательно дождётся положительного решения по данной транзакции и также прочитает изменение по ключу `y`.

Другими словами, если клиент увидел результат выполнения волатильной транзакции, то все последующие чтения продолжат видеть результат этой транзакции и стабильность видимости изменений не нарушается.

Косвенное планирование волатильных транзакций

Можно заметить, что одна из возможных проблем, это отправка шага планирования лишь на часть шардов. В этом случае может возникать ситуация, когда часть участников выполнит транзакцию и начнёт ожидания решения от шардов, которые ждут её планирования. Так как таймаут планирования составляет 30 секунд это могло бы приводить к излишним задержкам в 30 секунд до отмены транзакции.

Чтобы избежать таких задержек, после получения шага планирования хотя бы одним из участников и выполнения транзакции, этот шаг упоминается в запросах на ReadSet'ы и отправляемых ReadSet'ах. Участники, получив такое сообщение, косвенно узнают в каком `PlanStep` должна быть выполнена транзакция и запоминают его как `PredictedStep`. Даже если шаг планирования оказался потерян, DataShard добавляет транзакцию в PlanQueue на соответствующем шаге при получении не меньшего шага планирования по другим транзакциям, либо при продвижении медиаторного времени. В случае же если полученный таким образом шаг планирования оказался в прошлом, транзакция быстро отменяется, аналогично таймауту планирования.

Важно упомянуть, что т.к. транзакция может быть запланирована несколько раз, разные участники могут пропускать разные шаги планирования, и в итоге транзакция может оказаться запланированной на несовпадающие шаги между разными участниками. По этой причине, сообщения с `DECISION_COMMIT` считаются валидными только при совпадении `PlanStep` транзакции на разных участниках. В противном случае такой ReadSet приводит к отмене транзакции.

Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

Устройство механизма конфигурации V2

Конфигурация V2 в YDB реализует единый подход к управлению настройками кластера. В то время как страница Обзор конфигурации V2 в разделе Администрирование кластеров описывает как пользоваться этим механизмом, эта статья сфокусирована именно на его техническом устройстве. Она будет полезна разработчикам и контрибьюторам YDB, желающим внести изменения в этот механизм, а также всем, кто хочет более глубоко разобраться в происходящем на кластере YDB при изменении конфигурации.

Конфигурация кластера YDB хранится в нескольких местах и различные компоненты кластера координируют синхронизацию между ними:

- в виде набора файлов конфигурации узла в файловой системе каждого узла YDB (необходимы для использования в момент запуска узла для подключения к остальному кластеру);
- в специальной области хранения метаданных на каждом PDisk (кворум из PDisk считается единственным источником правды о конфигурации);
- в локальной базе таблетки DS Controller (для нужд распределённого хранилища);
- в локальной базе таблетки Console.

Часть параметров вступает в силу на узле сразу после того, как изменённая конфигурация будет доставлена туда, а часть — только после перезапуска узла.

Распределённая система конфигурации (Distconf)

Distconf — это система управления конфигурацией V2 кластера YDB, основанная на Node warden, узлах хранения и их PDisk'ах. Все изменения конфигурации V2, включая первоначальную инициализацию, проходят через неё.

Роль лидера

Центральным элементом системы Distconf является **лидер** — единственный узел в кластере, который на текущий момент имеет право инициировать и координировать изменения конфигурации. Наличие единой точки принятия решений устраняет гонки и конфликты, гарантируя последовательное применение изменений. Если текущий лидер выходит из строя, кластер автоматически запускает процесс выборов нового лидера. Далее в этом разделе мы рассмотрим, как именно происходят выборы и как лидер управляет кластером.

Кворум и источник правды

Ключевым элементом надёжности Distconf является хранение конфигурации непосредственно на PDisk'ах. Любое изменение считается успешно применённым только тогда, когда оно записано в область метаданных на **кворуме** дисков. Это означает, что даже при выходе из строя части узлов или дисков система сможет восстановить актуальную и целостную конфигурацию, прочитав её с оставшихся. Именно кворумное хранилище на PDisk'ах, а не состояние какого-либо одного узла, является единственным источником правды о конфигурации кластера. Этот механизм реализован в `distconf_persistent_storage.cpp`.

Стоит отметить, что запись в область метаданных PDisk'а для Distconf — это специальная низкоуровневая операция, которая происходит напрямую через `NodeWarden` и **не использует** стандартный путь работы с данными через `DSProxy`.

Сам механизм кворума является иерархическим и реализован в `distconf_quorum.h`. Базовый принцип — «большинство из большинств». Для принятия решения (например, о применении новой конфигурации) требуется:

1. Наличие большинства ответивших дисков внутри каждого дата-центра. Дата-центр, выполнивший это условие, считается «работоспособным».
2. Наличие большинства «работоспособных» дата-центров в кластере.

Для конфигураций, затрагивающих статические группы хранения, требование еще строже: каждая статическая группа должна подтвердить наличие собственного внутреннего кворума VDisk'ов в соответствии со своей схемой отказоустойчивости.

Процесс Binding и выборы лидера

Выборы лидера в Distconf осуществляются в процессе построения связей (`Binding`), описанном в `distconf_binding.cpp`.

1. **Обнаружение**: Узлы хранения получают полный список статических узлов кластера через стандартный механизм `TEvInterconnect::TEvNodesInfo`.
2. **Построение дерева связей**: Каждый узел инициирует подключение (`bind`) к случайному узлу из общего списка, которому он ещё не подчинён. Этот процесс формирует ациклический граф, который на промежуточных стадиях представляет собой лес (набор несвязанных деревьев), а в конечном итоге — единое дерево, покрывающее все узлы.
3. **Предотвращение циклов и обмен топологией**: Механизм защиты от циклов основан на постоянном обмене информацией о текущей топологии. Когда узел `А` пытается подключиться к узлу `Б`, он отправляет `Б` всё своё известное поддерево. Узел `Б` отклоняет запрос, если `А` уже является его потомком. В случае одновременной попытки взаимного подключения конфликт разрешается в пользу узла с большим `NodeId`.
4. **Определение лидера**: В процессе слияния деревьев неизбежно остаётся один узел, который не может ни к кому подключиться, потому что все остальные узлы уже находятся в его поддереве. Этот узел становится корнем итогового дерева и объявляется **лидером**.

Если текущий лидер выходит из строя, процесс `Binding` запускается заново, и кластер выбирает нового лидера.

Scatter/Gather — механизм распространения команд

Для управления кластером лидер использует механизм `Scatter/Gather` (`distconf_scatter_gather.cpp`), который работает поверх дерева, построенного в процессе `Binding`.

- **Scatter (рассылка)**
  Когда лидеру нужно отдать команду всем узлам (например, предложить новую конфигурацию), он отправляет её своим прямым потомкам в дереве. Каждый узел, получив команду, ретранслирует её своим потомкам. Так команда эффективно распространяется по всему дереву до самых листьев.
- **Gather (сбор)**
  После выполнения команды узлы должны отчитаться о результате. Ответы собираются в обратном порядке: листья отправляют результаты своим родителям, те, в свою очередь, агрегируют их и отправляют выше. В итоге лидер получает обобщённый результат от всего кластера.

Этот механизм используется для распределённых операций, включая двухфазный коммит при изменении конфигурации. Задачи рассылки и сбора (`TScatterTasks`) отслеживаются лидером для контроля выполнения операций.

Конечный автомат (FSM) и жизненный цикл изменений

Весь процесс изменения конфигурации на лидере управляется конечным автоматом (finite-state machine или FSM), реализованным в `distconf_fsm.cpp`. FSM гарантирует, что в один момент времени выполняется только одна операция изменения конфигурации, предотвращая гонки и конфликты.

Когда поступает запрос на изменение, FSM переходит в состояние `IN_PROGRESS`, блокируя новые запросы.

Лидер использует `Scatter/Gather` для выполнения двухфазного коммита: сначала рассылает предложение (Propose), а после получения кворума подтверждений — команду на применение (Commit).

После успешного завершения или отмены операции FSM возвращается в состояние `RELAX`, разрешая обработку следующих запросов.

Управление конфигурацией через InvokeOnRoot

Запросы `TEvNodeConfigInvokeOnRoot` представляют собой унифицированный механизм для любых изменений конфигурации кластера. Эти команды могут быть инициированы как **администратором системы** (например, через CLI), так и **другими компонентами YDB** в автоматическом режиме (например, таблеткой `BlobStorageController` в процессе `SelfHeal`).

Независимо от источника, любой такой запрос обрабатывается по единому сценарию:

1. Запрос доставляется на узел-лидер Distconf (если был отправлен не на него, то перенаправляется автоматически).
2. Лидер запускает FSM и выполняет процесс Scatter/Gather, как описано выше.

Этот механизм гарантирует, что любое изменение конфигурации, вне зависимости от его природы, проходит через строгую процедуру валидации и кворумного подтверждения.

Основные команды, поддерживаемые этим механизмом:

- `UpdateConfig`: Внести частичные изменения в текущую конфигурацию. Изменения передаются в виде protobuf-сообщения `TStorageConfig`.
- `QueryConfig`: Запросить текущую и предлагаемую конфигурацию. Ответ содержит protobuf-сообщения `TStorageConfig`.
- `ReplaceStorageConfig`: Заменить текущую конфигурацию новой, переданной в виде YAML.
- `FetchStorageConfig`: Вернуть загруженную YAML конфигурацию кластера.
- `ReassignGroupDisk`: Заменить диск в статической группе хранения.
- `StaticVDiskSlain`: Обработать событие выхода из строя VDisk'а в статической группе.
- `DropDonor`: Удалить диски-доноры после завершения миграции данных.
- `BootstrapCluster`: Инициировать первоначальное создание кластера.

Интеграция и дополнительные механизмы

Помимо основных процессов, Distconf тесно взаимодействует с другими компонентами системы:

- **Контроллер распределённого хранилища**

  DS-controller получает от Distconf изменения в конфигурации и использует их для работы распределённого хранилища.
- **Узлы баз данных**

  Узлы баз данных (database nodes) подписываются на события `TEvNodeWardenDynamicConfigPush`, чтобы получать обновления конфигурации в реальном времени.
- **SelfHeal**

  При использовании Distconf для статической группы работает SelfHeal по аналогии с динамическими группами.
- **Локальные YAML-файлы на узлах**

  Хранятся в директории, указанной аргументом запуска сервера `ydbd --config-dir`, и обновляются при получении информации о каждом изменении конфигурации. Эти файлы нужны при старте узла, чтобы обнаружить PDisk'ы и другие узлы кластера, а также установить начальные сетевые соединения. Данные в этих файлах могут быть устаревшими, особенно при выключении узла на длительное время.

Базовый порядок работы Distconf

1. При старте узла Distconf пытается прочитать конфигурацию с локальных PDisk'ов.
2. Подключается к случайному узлу хранения для проверки актуальности конфигурации.
3. Если не удаётся подключиться, но есть кворум подключённых, становится лидером.
4. Лидер пытается инициировать первоначальную настройку кластера, если она разрешена.
5. Лидер рассылает актуальную конфигурацию всем узлам через Scatter/Gather.
6. Узлы сохраняют полученную конфигурацию в локальной области PDisk для метаданных и в директориях `--config-dir`.

Итоговое распределение конфигурации

Место хранения	Содержит
`TPDiskMetadataRecord` на кворуме из PDisk'ов	Истинную актуальную конфигурацию
Локальная директория `--config-dir`	Исходный YAML для старта (может быть устаревшим)
Console	Актуальная копия (с минимальной задержкой)
DS-controller	Подмножество конфигурации, необходимое для распределённого хранилища

Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

Устройство режима bridge

> **Функциональность Корпоративной СУБД Яндекса**
>
> Данная функциональность доступна только в Корпоративной СУБД Яндекса. В open-source версии YDB она отсутствует.

Общее описание режима и сценарии использования см. в статье Режим работы кластера Bridge. Эта статья предназначена для разработчиков и контрибьюторов YDB и содержит технические детали реализации режима bridge.

YDB поддерживает произвольное количество pile, однако для простоты изложения далее рассматривается случай с двумя pile, назовём их pile А и pile Б.

Хранение конфигураций

Distconf хранит конфигурацию двух статических групп, двух наборов state storage, а также scheme board и board. В нём же сохраняется режим работы кластера.

Конфигурация статических групп, state storage и бордов использует схему хранения с двумя комплектами кворумов:

- Кворум А — включает только узлы из pile А. Запись считается успешной, если она прошла на большинстве узлов pile А и на кворуме дисков статической группы pile А;
- Кворум Б — включает только узлы из pile Б. Запись считается успешной, если она прошла на большинстве узлов pile Б и на кворуме дисков статической группы pile Б.

При отказе одного из pile запись в соответствующий кворум становится невозможной.

Distconf использует оба кворума (А и Б) для хранения конфигурации:

- Конфигурация с режимом `PRIMARY/DISCONNECTED` записывается только в кворум А;
- Конфигурация с режимом `DISCONNECTED/PRIMARY` записывается только в кворум Б;
- Все остальные режимы записываются в оба кворума — А и Б.

При выполнении failover администратор переводит кластер в режим `PRIMARY/DISCONNECTED` или `DISCONNECTED/PRIMARY`. В этом режиме конфигурация общих компонентов сохраняется только на кворуме из `PRIMARY` pile.

В кластере из более чем двух pile аварийное переключение может перевести несколько недоступных pile в `DISCONNECTED` одним запросом `UpdateClusterState`, одновременно выбрав новый `PRIMARY` из уже синхронизированных pile. Повторяемый параметр `--pile` команды failover формирует это единое обновление. При отдельных переходах промежуточная конфигурация может требовать участия другого недоступного pile, что не позволит собрать кворум. Кворумы проверяются по новой конфигурации: отключённые pile исключаются, а требования к кворумам узлов и хранилища внутри остальных участвующих pile не меняются.

Для начала восстановления администратор переводит пару из живого и отказавшего pile в режим `PRIMARY/NOT_SYNCHRONIZED`. Такая конфигурация сохраняется на всех кворумах.

Запись конфигурации запрещена, если новая конфигурация несовместима с последней сохранённой на узле. Например, переход из `PRIMARY/DISCONNECTED` в `DISCONNECTED/PRIMARY` недопустим. Это предотвращает ситуацию, при которой pile А и Б одновременно переключаются в режимы, претендующие на роль `PRIMARY`.

Для защиты от взаимодействия pile А и pile Б в случае конфликта конфигураций блокируется установка сессий interconnect при обнаружении подобных несоответствий на узлах.

У Distconf при работе в режиме bridge в каждом pile создаётся поддерево, и корни этих поддеревьев присоединяются к лидеру Distconf. Это позволяет при потере pile минимизировать необходимое перестроение связей узлов Distconf.

Запуск статических групп

Конфигурация всех статических групп хранится в Distconf. Корень поддерева Distconf в каждом pile может получить конфигурацию и принять решение о необходимом для применения этой конфигурации кворуме на основании самой конфигурации. `DISCONNECTED` pile не участвует в кворуме, поэтому для применения конфигурации без `DISCONNECTED` pile необходимы кворумы А и Б, для применения конфигурации `PRIMARY/DISCONNECTED` необходим только кворум А, для применения конфигурации `DISCONNECTED/PRIMARY` необходим только кворум Б.

Когда необходимый кворум собран, Distconf применяет конфигурацию, а Node Warden запускает VDisk и PDisk статических групп.

Для доступа к статической группе используется сервис `dsproxy-proxy`, который предоставляет таблеткам интерфейс `dsproxy` и выполняет операции синхронно с обеими группами в паре.

Dsproxy-proxy статической группы меняет режим работы в зависимости от конфигурации, полученной от distconf. Конфигурация dsproxy-proxy хранится в конфигурации общих компонент.

Поведение dsproxy-proxy

В режиме `PRIMARY/SYNCHRONIZED` запись выполняется синхронно в две группы. Чтение также отправляется в обе группы, и ответ синхронизированной группы, в которой есть все запрошенные блобы, принимается сразу. Dsproxy-proxy не опрашивает группы ping-запросами: о разрыве связи он узнаёт от interconnect, а о смене режима — из конфигурации, получаемой от distconf.

В режиме `PRIMARY/DISCONNECTED` запись и чтение выполняются только в `PRIMARY`-группе.

Узел узнаёт об отключении pile, когда до него доходит новая конфигурация. До этого его dsproxy-proxy продолжает писать в отключённый pile и читать из него, а в отключённом pile могут отсутствовать записи, уже выполненные узлами, знающими об отключении. Номера поколений не позволяют отсечь такой pile, поскольку он не узнаёт новые значения. Поэтому dsproxy-proxy никогда не считает отсутствие блоба в одном pile ответом: чтение, не нашедшее блоб, и любое чтение диапазона ждут ответов всех pile, которым был отправлен запрос, и возвращают объединение того, что в них есть. Блоб, найденный в любом pile, возвращается; отсутствие блоба сообщается, только если его нет ни в одном pile, а ошибка от какого-либо pile приводит к ошибке чтения, а не к сообщению об отсутствии блоба. При плановом отключении, когда pile остаётся доступным, окно ограничено скоростью доставки конфигурации до каждого узла; получив её, узел также разрывает interconnect-сессии с отключённым pile. При внеплановом переключении отключённый pile недоступен, поэтому такое окно не возникает.

`Assimilate` опрашивает участвующие pile, части группы в которых достигли `WRITE_KEEP_BARRIER_DONOTKEEP_DATA` или `SYNCED`, включая ещё синхронизируемые части. Как минимум одна выбранная часть должна считаться полной в конфигурации, с которой принят запрос. Запрос ждёт всех успешных ответов и объединяет блокировки, барьеры и флаги блобов только в пределах общего префикса перечня, покрытого каждой страницей ответа, чтобы при переходе к следующей странице не пропустить ещё не полученные записи другого pile. Ошибка от pile не позволяет вернуть успешный объединённый перечень.

`CheckIntegrity` опрашивает каждую участвующую часть, считающуюся полной: часть группы в стадии `SYNCED` в pile со статусом `SYNCHRONIZED`, а если у группы назначен авторитетный pile — только его часть в стадии `SYNCED`. Успешный результат ждёт всех выбранных ответов и содержит худшие статусы размещения и данных, а диагностические сообщения помечены идентификаторами pile. Поэтому деградация любой выбранной копии видна, даже если другая копия исправна; ещё синхронизируемые части не считаются деградировавшими копиями. Неудачная проверка возвращает ошибку.

В режиме `PRIMARY/NOT_SYNCHRONIZED` dsproxy-proxy поддерживает несколько подрежимов, определяющих, какие виды записи выполняются в `NOT_SYNCHRONIZED`-группе:

- `WRITE_KEEP` — выполняется только запись `KEEP`-флагов;
- `WRITE_KEEP_BARRIER_DONOTKEEP` — выполняется запись барьеров, `KEEP`- и `DO_NOT_KEEP`-флагов;
- `WRITE_KEEP_BARRIER_DONOTKEEP_DATA` — выполняется запись данных, барьеров, `KEEP`- и `DO_NOT_KEEP`-флагов.

Конфигурация bridge-группы содержит для каждого pile:

- идентификатор группы, которую хранит pile;
- подрежим этой группы, определяющий, какие записи она получает при синхронизации;
- номер поколения конфигурации этой группы.

Также в ней указан номер поколения состояния кластера, для которого она составлена. Dsproxy-proxy, у которого более старое состояние кластера, ждёт более нового, прежде чем отправить запрос, поэтому режимы pile всегда берутся вместе с составленной для них конфигурацией группы.

Dsproxy-proxy отправляет каждый запрос в группу pile с номером поколения этой группы из своей конфигурации (`ForceGroupGeneration`). Dsproxy группы отвечает `RACE`, если поколение его конфигурации другое, а VDisk выполняют только запросы, сделанные для их точного поколения группы. При каждом изменении конфигурации bridge-группы, в том числе при смене подрежима, контроллер (для статической группы — distconf) увеличивает номера поколений всех её групп, поэтому запросы dsproxy-proxy, пропустившего изменение, не выполняются: они возвращаются с `RACE` и перезапускаются с более новой конфигурацией. Отключённый pile не узнаёт новых номеров поколений, поэтому отключение так отсечь нельзя.

Для перехода из режима `PRIMARY/NOT_SYNCHRONIZED` в режим `PRIMARY/SYNCHRONIZED` выполняется синхронизация. В ходе синхронизации недостающие данные переносятся из `PRIMARY`-группы в `NOT_SYNCHRONIZED`-группу, а также выполняется двустороннее копирование информации о блокировках.

Bootstrap BSController и других системных таблеток

В списке узлов для запуска таблетки достаточно указать один набор узлов, включающий достаточное количество узлов из каждого pile. Bootstrapper должен анализировать конфигурацию state storage proxy-proxy и на её основе определять, можно ли запускать таблетку на данном узле. Если запуск таблетки на узле невозможен (поскольку узел не находится в `PRIMARY`-pile), Bootstrapper должен пройти по дереву подключённых к нему Bootstrapper-ов и назначить в качестве запускающего Bootstrapper из `PRIMARY`-pile. Если такого Bootstrapper-а нет, текущий Bootstrapper не должен запускать таблетку.

Конфигурация и запуск остальных таблеток

Динамическими группами хранения управляет BS Controller. Вместо каждого обычного storage-пула каждый раз создаётся три storage-пула: в пуле А — группы из pile А, в пуле Б — группы из pile Б, в пуле Ц — прокси-прокси группы, составленные из пар групп из пулов А и Б. Управляемые хайвом таблетки используют только прокси-прокси группы из пула Ц.

Запуск таблеток выполняется только в `PRIMARY` или `PROMOTED`-pile. Перевоз таблеток из `PRIMARY` в `PROMOTED` выполняется по возможности мягко, перевоз таблеток из `DISCONNECTED` в `PRIMARY` происходит максимально быстро.

Поведение interconnect

В interconnect разделяются обмен метаданными и установка сессии.
Обмен метаданными происходит до установления сессии и позволяет паре узлов определить используемый набор функций interconnect и обменяться конфигурацией.
Interconnect с узлами из отключённой половины кластера разрывается, и попытки установления соединения не предпринимаются. Установление соединения возможно только после перевода pile в состояние `PRIMARY/NOT_SYNCHRONIZED`.

Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

Spilling Service

Обзор

Spilling Service — это акторный сервис, который предоставляет временное хранилище для блобов данных в системе YDB. Сервис работает по принципу key-value хранилища, где клиенты могут сохранять данные по уникальному идентификатору и получать их обратно по этому идентификатору.

Архитектура

Основные компоненты

- **Task queue**: Сервис поддерживает внутреннюю очередь операций записи и чтения. Все запросы на спиллинг помещаются в данную очередь и обрабатываются асинхронно.
- **Thread pool**: Для выполнения операций ввода-вывода используется пул рабочих потоков.
- **File management**: Сервис автоматически создает, удаляет и управляет файлами на диске.
- **Resource monitoring**: Сервис осуществляет мониторинг использования дискового пространства, количества активных операций и других метрик производительности.

Хранение данных

Данные сохраняются в файлах на локальной файловой системе. Spilling Service обеспечивает:

- распределение записей между файлами;
- удаление файлов;
- управление жизненным циклом данных.

При непредвиденном перезапуске устаревшие файлы удаляются автоматически.

Взаимодействие с компонентами

Компоненты системы интегрированы с Spilling Service и взаимодействуют с ним через события акторной системы:

Мониторинг состояния памяти

Вычислительные узлы осуществляют постоянный мониторинг состояния памяти через аллокатор. Аллокатор информирует узлы о снижении объема свободной памяти. Однако, система не дожидается полного исчерпания памяти, поскольку процесс спиллинга также требует дополнительных ресурсов памяти для сериализации и буферизации данных.

Отправка событий

При обнаружении необходимости спиллинга данных вычислительный компонент (подсистема передачи данных или вычислительное ядро) выполняет следующие действия:

1. Сериализует данные в блоб
2. Генерирует уникальный идентификатор для блоба
3. Создает запрос на спиллинг с блобом и сгенерированным идентификатором
4. Отправляет запрос в Spilling Service
5. Освобождает ресурсы и переходит в режим ожидания, давая возможность другим задачам использовать вычислительные ресурсы

Ожидание результатов

После отправки запроса вычислительный компонент освобождает ресурсы для других задач и переходит в режим ожидания, позволяя системе оптимально использовать вычислительные мощности кластера до завершения записи во внешнее хранилище.

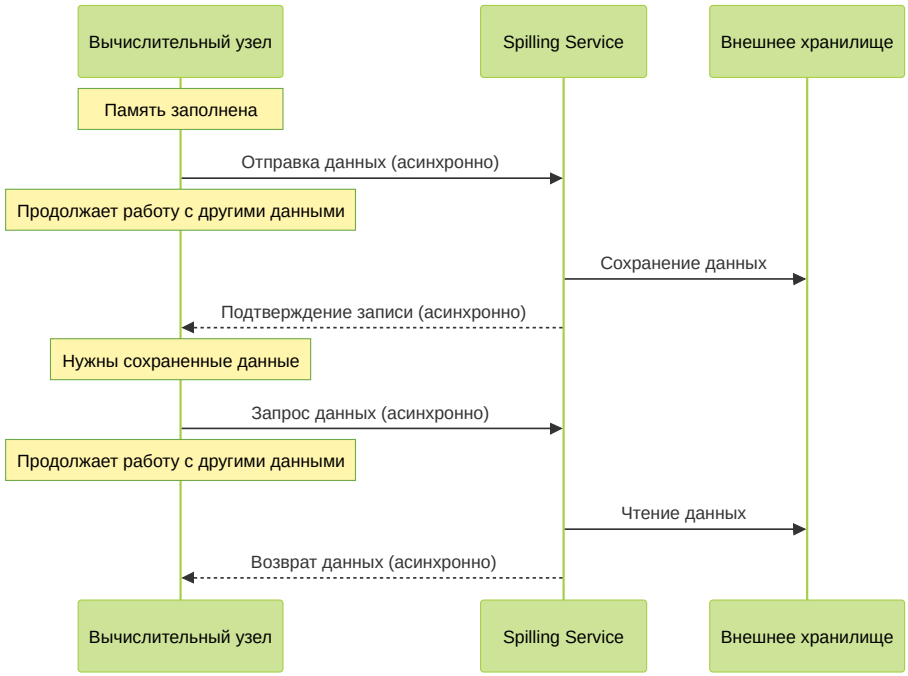
Обработка ответов

Spilling Service обрабатывает запрос и возвращает подтверждение записи по указанному идентификатору или сообщение об ошибке. Продолжение работы вычислительного компонента возможно только после получения подтверждения.

Чтение данных

При необходимости восстановления данных компонент отправляет запрос чтения с идентификатором блоба. Spilling Service читает данные из внешней памяти и возвращает ответ с восстановленными данными. Во время ожидания загрузки данных освободившиеся вычислительные ресурсы используются для обработки других задач.

Схема работы спиллинга



Конфигурация

Подробная информация о настройке Spilling Service доступна в разделе Конфигурация спиллинга.

См. также

- Концепция спиллинга
- Конфигурация спиллинга
- Мониторинг YDB
- Диагностика производительности

Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

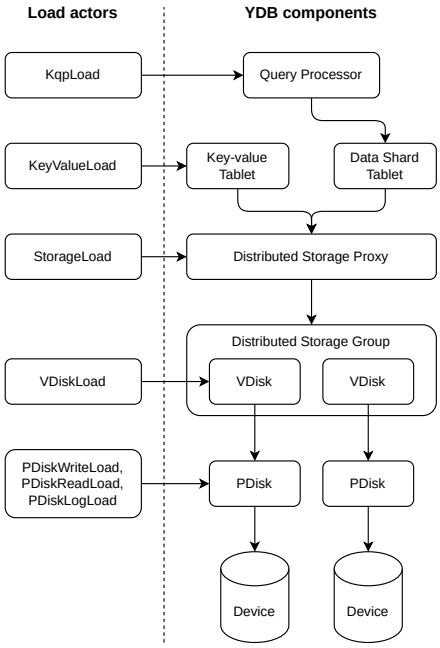
Обзор тестирования с помощью нагружающих акторов

Тестирование производительности системы является важным этапом добавления изменений в ядро YDB. С помощью нагрузочного тестирования вы можете:

- определить показатели производительности и сравнить их со значениями до изменений;
- проверить работоспособность системы на высоких или пиковых нагрузках.

Сама по себе задача нагрузить высокопроизводительную распределенную систему является нетривиальной. Часто разработчику приходится запускать несколько инстансов клиента, чтобы добиться требуемой загрузки сервера. YDB реализует простой и удобный механизм подачи нагрузки — нагружающие акторы. Акторы создаются и запускаются в самом кластере, таким образом не нужно привлекать дополнительные ресурсы для запуска клиентов. Нагружающие акторы могут быть запущены на произвольных узлах кластера — на одном, на всех или только на выбранных. Вы можете создать любое количество акторов на любом из узлов.

Нагружающие акторы позволяют тестировать как систему в целом, так отдельные ее компоненты:



Например, вы можете подать нагрузку на Distributed Storage без задействования слоев таблеток и Query Processor. Таким образом можно изолированно тестировать отдельные слои системы и эффективно находить узкие места. Комбинация акторов разных типов позволяет запускать множество видов нагрузки.

> **Примечание**
>
> Функциональность находится в разработке, исходный код доступен в ветке *main* репозитория YDB. О том, как собрать YDB из исходного кода, читайте в инструкции.

Типы акторов

Тип	Описание
KqpLoad	Подает нагрузку на слой Query Processor и нагружает все компоненты кластера.
KeyValueLoad	Нагружает Key-value таблетку.
StorageLoad	Нагружает Distributed Storage без задействования слоев таблеток и Query Processor.
NbsDbgLikeLoad	Генерирует блочную нагрузку, подобную NBS, через персистентную нагружающую таблетку, группы прямого блочного доступа (DBG), PersistentBuffer и DDisk.
VDiskLoad	Тестирует производительность записи на VDisk.
PDiskWriteLoad	Тестирует производительность записи на PDisk.
PDiskReadLoad	Тестирует производительность чтения с PDisk.
PDiskLogLoad	Тестирует корректность вырезания из середины лога PDisk.
MemoryLoad	Аллоцирует память, полезен при тестировании логики.
Stop	Останавливает все акторы, либо только указанные.

Запуск нагрузки

Запустить нагрузку можно с помощью следующих инструментов:

- YDB UI кластера — позволяет создать по конфигурации и запустить нагружающий актор либо на текущем узле, либо сразу на всех узлах тенанта.

- Утилита `ydbd` — позволяет на любой узел кластера отправить конфигурацию актора с указанием, на каких узлах необходимо создать и запустить актор.

В качестве примера рассмотрим создание и запуск актора KqpLoad. Актор обращается к БД `/slice/db` как к key-value хранилищу в 64 потока, длительность нагрузки — 30 секунд. Перед началом работы актор создает необходимые таблицы, после окончания удаляет их. При создании актору будет автоматически присвоен тег. Этот же тег будет присвоен и результату теста.

YDB UI

1. Откройте страницу управления нагружающими акторами на узле (например, `http://<address>:8765/actors/load`, где `address` — адрес узла кластера, на котором нужно запустить нагрузку).
2. В поле ввода/вывода вставьте конфигурацию актора:

```
KqpLoad: {
    DurationSeconds: 30
    WindowDuration: 1
    WorkingDir: "/slice/db"
    NumOfSessions: 64
    UniformPartitionsCount: 1000
    DeleteTableOnFinish: 1
    WorkloadType: 0
    Kv: {
        InitRowCount: 1000
        PartitionsByLoad: true
        MaxFirstKey: 18446744073709551615
        StringLen: 8
        ColumnsCnt: 2
        RowsCnt: 1
    }
}
```

3. Чтобы создать и запустить актор, нажмите кнопку:
   - **Start new load on current node** — нагрузка будет запущена на текущем узле.
   - **Start new load on all tenant nodes** — нагрузка будет запущена на всех узлах тенанта.

В поле ввода/вывода появится следующее сообщение:

```
{"status":"OK","tag":1}
```

- `status` — статус запуска нагрузки;
- `tag` — тег, который был присвоен нагрузке.

CLI

1. Создайте файл с конфигурацией актора:

```
KqpLoad: {
    DurationSeconds: 30
    WindowDuration: 1
    WorkingDir: "/slice/db"
    NumOfSessions: 64
    UniformPartitionsCount: 1000
    DeleteTableOnFinish: 1
    WorkloadType: 0
    Kv: {
        InitRowCount: 1000
        PartitionsByLoad: true
        MaxFirstKey: 18446744073709551615
        StringLen: 8
        ColumnsCnt: 2
        RowsCnt: 1
    }
}
```

2. Запустите актор:

```
curl <endpoint>/actors/load -H "Content-Type: application/x-protobuf-text" --data mode=start --data all_nodes
=<start_on_all_nodes> --data config="$(cat proto_file)"
```

- `endpoint` — http-эндпоит узла (например, `http://<address>:<port>`, где `address` — адрес узла, `port` — http-порт узла).
- `proto_file` — путь к файлу с конфигурацией актора.
- `start_on_all_nodes` – `true`, чтобы запустить нагрузку на всех узлах тенанта, `false`, чтобы запустить нагрузку только на узле с данным `endpoint`.

Просмотр результата тестирования

Результаты тестирования можно просмотреть с помощью YDB UI. Описание выводимых параметров смотрите в документации соответствующего актора.

YDB UI

1. Откройте страницу управления нагружающими акторами на узле (например, `http://<address>:<port>/actors/load`, где `address` — адрес узла, `port` — http-порт мониторинга узла, на котором была запущена нагрузка).

2. Нажмите кнопку **Results**.

Будут отображены результаты завершенных тестирований. Найдите результат с соответствующим тегом.

Tag# 1

Finish reason# OK called StartDeathProcess
Finish time# 2022-12-02T10:03:45.207002Z

Window	Txs	Txs/Sec	Errors	p50(ms)	p95(ms)	p99(ms)	pMax(ms)
total	333324	11110.8	0	4.895	6.303	11.455	64.511
30	10377	10377	0	4.927	10.111	18.303	34.559
29	11101	11101	0	5.055	6.015	6.527	8.319
28	7618	7618	0	5.023	7.903	19.711	64.511
27	10394	10394	0	4.895	10.623	18.815	36.351
26	11134	11134	0	5.055	5.983	6.463	7.519
25	11203	11203	0	5.023	6.111	7.871	13.567

Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

KqpLoad

Тестирует производительности кластера YDB в целом, нагружая все компоненты через слой Query Processor. Нагрузка, аналогична нагрузке от подкоманды workload YDB CLI, но запускается изнутри кластера.

Вы можете запустить два вида нагрузки:

- **Stock** — симулирует работу склада интернет-магазина: создает заказы из нескольких товаров, получает список заказов по клиенту.
- **Key-value** — использует БД как key-value хранилище.

Перед началом работы создаются необходимые таблицы, после завершения они удаляются.

Параметры актора

Ниже описаны основные параметры актора. Полный список параметров смотрите в файле load_test.proto Git-репозитория YDB.

Параметр	Описание
`DurationSeconds`	Продолжительность нагрузки в секундах.
`WindowDuration`	Размер окна для агрегации статистики.
`WorkingDir`	Путь директории, в которой будут созданы тестовые таблицы.
`NumOfSessions`	Количество параллельных потоков, подающих нагрузку. Каждый поток пишет в свою сессию.
`DeleteTableOnFinish`	Если `False`, то созданные таблицы не удаляются после завершения работы нагрузки. Может быть полезно в случае, когда при первом запуске актора создается большая таблица, а при последующих выполняются запросы к ней.
`UniformPartitionsCount`	Количество партиций, создаваемых в тестовых таблицах.
`WorkloadType`	Тип нагрузки. В случае Stock: • `0` — InsertRandomOrder; • `1` — SubmitRandomOrder; • `2` — SubmitSameOrder; • `3` — GetRandomCustomerHistory; • `4` — GetCustomerHistory. В случае Key-Value: • `0` — UpsertRandom; • `1` — InsertRandom; • `2` — SelectRandom.
`Workload`	Вид нагрузки. `Stock`: • `ProductCount` — количество видов товаров. • `Quantity` — количество товаров каждого вида на складе. • `OrderCount` — первоначальное количество заказов в БД. • `Limit` — минимальное количество шардов для таблиц. `Kv`: • `InitRowCount` — до начала нагрузки нагружающий актор запишет в таблицу указанное количество строк. • `StringLen` — длина строки `value`. • `ColumnsCnt` — сколько столбцов использовать в таблице. • `RowsCnt` — сколько строк вставлять или читать в одном SQL запросе.

Примеры

Следующий актор запускает stock-нагрузку БД `/slice/db`, выполняя простые UPSERT-запросы в `64` потока в течение `30` секунд.

```
KqpLoad: {
    DurationSeconds: 30
    WindowDuration: 1
    WorkingDir: "/slice/db"
    NumOfSessions: 64
    UniformPartitionsCount: 1000
    DeleteTableOnFinish: 1
    WorkloadType: 0
    Stock: {
        ProductCount: 100
        Quantity: 1000
        OrderCount: 100
        Limit: 10
    }
}
```

Результатом теста является количество успешных транзакций в секунду, количество повторных попыток исполнения транзакций и количество ошибок.

Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

KeyValueLoad

Нагружает Key-value таблетку.

Конфигурация актора

```
message TKeyValueLoad {
    message TWorkerConfig {
        optional string KeyPrefix = 1;
        optional uint32 MaxInFlight = 2;
        optional uint32 Size = 11; // data size, bytes
        optional bool IsInline = 9 [default = false];
        optional uint32 LoopAtKeyCount = 10 [default = 0]; // 0 means "do not loop"
    }
    optional uint64 Tag = 1;
    optional uint64 TargetTabletId = 2;
    optional uint32 DurationSeconds = 5;
    repeated TWorkerConfig Workers = 7;
}
```

Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

StorageLoad

О блочной нагрузке, подобной NBS, через группы прямого блочного доступа (DBG), PersistentBuffer и DDisk см. NbsDbgLikeLoad. `StorageLoad` проверяет путь хранения блобов через DSProxy/VDisk.

Тестирует производительность записи и чтения с Distributed Storage. Нагрузка подается непосредственно на Distributed Storage без задействования слоев таблеток и Query Processor. При тестировании производительности записи актор записывает данные в указанную группу хранения. Для тестирования чтения актор предварительно записывает данные в указанную группу хранения, а потом читает их. После снятия нагрузки все данные, записанные актором, удаляются.

Вы можете подать нагрузку трех видов:

- *Постоянная* — актор следит, чтобы одновременно было запущено указанное число запросов. Чтобы подать постоянную нагрузку, задайте нулевую паузу между запросами (например, `WriteIntervals: { Weight: 1.0 Uniform: { MinUs: 0 MaxUs: 0 } }`) и отличный от нуля `MaxInFlightWriteRequests`, при этом не задавайте `WriteHardRateDispatcher`.
- *Интервальная* — актор запускает запросы через заданные промежутки времени. Чтобы подать интервальную нагрузку, задайте ненулевую паузу между запросами (например, `WriteIntervals: { Weight: 1.0 Uniform: { MinUs: 50000 MaxUs: 50000 } }`) и не задавайте `WriteHardRateDispatcher`. Максимальное число одновременно выполняемых запросов задается параметром `MaxInFlightWriteRequests`. Если его значение равно `0`, то ограничения нет.
- *С фиксированной частотой* — актор запускает запросы через определенные промежутки времени, величина которых меняется таким образом, чтобы поддерживать определенную частоту запросов в секунду. Если продолжительность нагрузки ограничена параметром `DurationSeconds`, то эта частота может отличаться в момент старта и в момент завершения нагрузки и будет изменяться равномерно на протяжении всего основного цикла нагрузки. Чтобы подать нагрузку этого вида, задайте описание параметров нагрузки с фиксированной частотой (параметр `WriteHardRateDispatcher`). Обратите внимание, что если в параметрах актора задан `WriteHardRateDispatcher`, то запустится нагрузка с фиксированной частотой, вне зависимости значений `WriteIntervals`. Максимальное количество одновременно выполняемых запросов задается параметром `MaxInFlightWriteRequests`, если его значение равно `0`, то ограничения нет.

Параметры актора

Ниже описаны основные параметры актора. Полный список параметров смотрите в файле load_test.proto Git-репозитория YDB.

Параметр	Описание
`DurationSeconds`	Продолжительность нагрузки. Таймер запускается после завершения начальной записи данных.
`Tablets`	Нагрузка подается от имени таблетки со следующими реквизитами: • `TabletId` — идентификатор таблетки. Должен быть уникальным для каждого нагружающего актора в кластере. Этот параметр и `TabletName` – взаимоисключающие. • `TabletName` — имя таблетки. Если задан этот параметр, идентификаторы таблеток назначаются автоматически, таблеткам, запущенным на одной и той же ноде с одним и тем же именем, присваиваются одинаковые идентификаторы, таблеткам на разных нодах присваиваются различные идентификаторы. • `Channel` — канал таблетки. • `GroupId` — идентификатор группы хранения, на которую будет подана нагрузка. • `Generation` — поколение таблетки.
`WriteSizes`	Размер записываемых данных. Для каждого запроса выбирается случайным образом из интервала `Min`-`Max`. Вы можете задать несколько диапазонов `WriteSizes`, и тогда выбор значения из конкретного диапазона будет определяться его `Weight`.
`WriteHardRateDispatcher`	Описание параметров нагрузки с фиксированной частотой для запросов записи. Если задан этот параметр, то значение `WriteIntervals` игнорируется.
`WriteIntervals`	Описание параметров вероятностного распределения временных интервалов между записями для интервальной нагрузки в микросекундах. Вы можете задать несколько диапазонов `WriteIntervals`, и тогда выбор значения из конкретного диапазона будет определяться его `Weight`.
`MaxInFlightWriteRequests`	Максимальное количество одновременно обрабатываемых запросов на запись.
`MaxInFlightWriteBytes`	Максимальный суммарный объём данных в одновременно выполняемых запросах записи, в байтах; ноль означает отсутствие ограничения.
`ReadSizes`	Размер читаемых данных. Для каждого запроса выбирается случайным образом из интервала `Min`-`Max`. Вы можете задать несколько диапазонов `ReadSizes`, и тогда выбор значения из конкретного диапазона будет определяться его `Weight`.
`ReadIntervals`	Описание параметров вероятностного распределения временных интервалов между запросами для интервальной нагрузки в микросекундах. Вы можете задать несколько диапазонов `ReadIntervals`, и тогда выбор значения из конкретного диапазона будет определяться его `Weight`.
`ReadHardRateDispatcher`	Описание параметров нагрузки с фиксированной частотой для запросов чтения. Если задан этот параметр, то значение `ReadIntervals` игнорируется.
`MaxInFlightReadRequests`	Максимальное количество одновременно обрабатываемых запросов на чтение.
`MaxInFlightReadBytes`	Максимальный суммарный объём данных в одновременно выполняемых запросах чтения, в байтах; ноль означает отсутствие ограничения.
`FlushIntervals`	Описание параметров вероятностного распределения временных интервалов между запросами на удаление записанных основным циклом StorageLoad данных в микросекундах. Вы можете задать несколько диапазонов `FlushIntervals`, и тогда выбор значения из конкретного диапазона будет определяться его `Weight`. Одновременно будет обрабатываться только один запрос на удаление записанных данных.

`PutHandleClass`	Класс записи данных в дисковую подсистему. В случае `TabletLog` запись выполняется с максимальным приоритетом.
`GetHandleClass`	Класс чтения данных с дисковой подсистемы. В случае `FastRead` чтение выполняется с максимальной скоростью.
`InitialAllocation`	Описание параметров начальной записи данных. Определяет объем данных, который будет записан до старта основной нагрузки, и которые затем могут быть прочитаны запросами чтения, наряду с данными, записанными в основном цикле нагрузки.

Write requests class

Class	Description
`TabletLog`	Самый высокий приоритет запросов записи.
`AsyncBlob`	Используется для записи таблиц SSTable и их частей.
`UserData`	Используется для записи пользовательских данных отдельными блобами.

Read requests class

Class	Description
`AsyncRead`	Используется для чтения данных таблеток, прошедших процесс компакшена.
`FastRead`	Используется для быстрого чтения пользовательских данных.
`Discover`	Чтения запросов Discover.
`LowRead`	Низкоприоритетные чтения, выполняемые на фоне.

Параметры вероятностного распределения

Интервал, записанный в виде repeated-поля `TIntervalInfo`, вычисляется по следующему алгоритму:

- случайным образом выбирается элемент из массива `TIntervalInfo` с вероятностью, пропорциональной весу;
- для элемента типа `TIntervalUniform` равновероятно выбирается значение в интервале `Min-Max` (если указано `MinMs/MaxMs`, то значение в миллисекундах; если `MinUs/MaxUs` — в микросекундах);
- для элемента типа `TIntervalPoisson` выбирается интервал по формуле `Min(log(-x / Frequency), MaxIntervalMs)`, где `x` — случайное значение в интервале `[0, 1]`, что обеспечивает соответствие интервалов распределению Пуассона с частотой `Frequency`, но с интервалом не больше `MaxIntervalMs`.

Аналогичный механизм применяется и для вероятностного распределения размеров записываемых данных, за исключением того, что в том механизме есть только равновероятное распределение размера `[Min, Max]`.

Параметры нагрузки с фиксированной частотой

Параметр	Описание
`RequestsPerSecondAtStart`	Частота запросов в секунду в момент старта нагрузки. Если продолжительность нагрузки не задана, то частота запросов остается постоянной и равной величине этого параметра.
`RequestsPerSecondOnFinish`	Частота запросов в секунду в момент завершения нагрузки.

Параметры начальной записи данных

Параметр	Описание
`TotalSize`	Суммарный размер записанных данных. Этот параметр и `BlobsNumber` – взаимоисключающие.
`BlobsNumber`	Количество записанных блобов.
`BlobSizes`	Размер записываемых блобов. Для каждого запроса выбирается случайным образом из интервала `Min` - `Max`. Вы можете задать несколько диапазонов `BlobSizes`, и тогда выбор значения из конкретного диапазона будет определяться его `Weight`.
`MaxWritesInFlight`	Максимальное количество одновременно обрабатываемых запросов на запись. Если парамаетр не задан, то ограничения на количество одновременно обрабатываемых запросов нет.
`MaxWriteBytesInFlight`	Максимальный суммарный объем данных одновременно обрабатываемых запросов на запись. Если парамаетр не задан, то ограничения на суммарный объем данных одновременно обрабатываемых запросов нет.
`PutHandleClass`	Класс записи данных в дисковую подсистему.
`DelayAfterCompletionSec`	Время в секундах, которое актор ждет от момента завершения начальной записи данных до момента старта основной нагрузки. Если параметр не задан, нагрузка начнется сразу же после завершения начальной записи данных.

Примеры

Нагрузка на запись

Следующий актор будет писать в группу с идентификатором `2181038080` в течение `60` секунд. Размер одной записи `4096` байт, число одновременно выполняемых запросов не более `256` (постоянная нагрузка):

```
StorageLoad: {
    DurationSeconds: 60
    Tablets: {
        Tablets: { TabletId: 1000 Channel: 0 GroupId: 2181038080 Generation: 1 }
        WriteSizes: { Weight: 1.0 Min: 4096 Max: 4096 }
        WriteIntervals: { Weight: 1.0 Uniform: { MinUs: 0 MaxUs: 0 } }
        MaxInFlightWriteRequests: 256
        FlushIntervals: { Weight: 1.0 Uniform: { MinUs: 10000000 MaxUs: 10000000 } }
        PutHandleClass: TabletLog
    }
}
```

При просмотре результата тестирования наибольший интерес представляют следующие значения:

- `Writes per seconds` — количество записей в секунду, например `28690.29`.
- `Speed@ 100%` — 100 перцентиль скорости записи в МБ/с, например `108.84`.

Нагрузка на чтение

Чтобы подать нагрузку на чтение, необходимо сначала записать данные. Данные записываются запросами по `4096` байт каждые `50` мс, при этом число одновременно выполняемых запросов не более `1` (интервальная нагрузка). Если запрос не успеет завершиться за `50` мс, актор дождется его завершения и через `50` мс запустит следующий запрос. Данные старше `10` с удаляются. Чтение данных выполнятся запросами по `4096` байт, число одновременно выполняемых запросов `16` (постоянная нагрузка):

```
StorageLoad: {
    DurationSeconds: 60
    Tablets: {
        Tablets: { TabletId: 5000 Channel: 0 GroupId: 2181038080 Generation: 1 }
        WriteSizes: { Weight: 1.0 Min: 4096 Max: 4096}
        WriteIntervals: { Weight: 1.0 Uniform: { MinUs: 50000 MaxUs: 50000 } }
        MaxInFlightWriteRequests: 1

        ReadSizes: { Weight: 1.0 Min: 4096 Max: 4096 }
        ReadIntervals: { Weight: 1.0 Uniform: { MinUs: 0 MaxUs: 0 } }
        MaxInFlightReadRequests: 16
        FlushIntervals: { Weight: 1.0 Uniform: { MinUs: 10000000 MaxUs: 10000000 } }
        PutHandleClass: TabletLog
        GetHandleClass: FastRead
    }
}
```

При просмотре результата тестирования наибольший интерес представляют следующие значение:

- `ReadSpeed@ 100%` — 100 перцентиль скорости чтения МБ/с, например `60.86`.

Нагрузка только на чтение

До старта основной нагрузки записывается блок данных, объемом `1 GB`, состоящий из блобов размером от `1 MB` до `5 MB`. Чтобы не перегрузить систему запросами записи, количество одновременно выполняемых запросов ограничивается `5`. После завершения записи этого блока запускается readonly нагрузка с возрастающей частотой: с `10` до `50` запросов в секунду, частота возрастает равномерно на протяжении `300` секунд.

```
StorageLoad: {
    DurationSeconds: 300
    Tablets: {
        Tablets: { TabletId: 5000 Channel: 0 GroupId: 2181038080 Generation: 1 }

        MaxInFlightReadRequests: 10
        GetHandleClass: FastRead
        ReadHardRateDispatcher {
                RequestsPerSecondAtStart: 10
                RequestsPerSecondOnFinish: 50
        }

        InitialAllocation {
                TotalSize: 1000000000
                BlobSizes: { Weight: 1.0 Min: 1000000 Max: 5000000 }
                MaxWritesInFlight: 5
        }
    }
}
```

Рассчитанные перцентили скорости чтения/записи относятся только к запросам основного цикла нагрузки, запросы начальной записи в них не учитываются. Представляют интерес графики в Monitoring, например, по ним можно проследить, как ухудшается время отклика системы по мере роста нагрузки.

Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

NbsDbgLikeLoad

`NbsDbgLikeLoad` измеряет производительность упрощённого пути записи и чтения Network Block Store (NBS) через группы прямого блочного доступа (DBG). Персистентная нагружающая таблетка владеет DBG и отправляет запросы в PersistentBuffer (PB) и DDisk. Отдельные нагружающие акторы генерируют запросы и измеряют время их выполнения.

Используйте этот актор для сравнения репликации PB, фонового копирования в DDisk, пакетной обработки и параллельного ввода-вывода. Для проверки пути хранения блобов через DSProxy/VDisk используйте StorageLoad.

Предварительные требования

Используйте тестовый кластер с пулами DDisk и PB, настроенными в BlobStorage Controller (BSC). При выделении ресурсов должно возвращаться число пар DDisk/PB, соответствующее `HostsPerDbg`. Нагружающей таблетке также нужны обычное хранилище для каналов таблетки и работающий Hive; пулы DDisk это хранилище не предоставляют.

Откройте `http://<node>:8765/actors/load?mode=tablet` на узле нужной базы данных. Эта страница позволяет создавать таблетки, просматривать их идентификаторы Hive и размещение, а также запускать нагрузку. Порт мониторинга в вашем кластере может отличаться.

> **Важно**
>
> Запускайте не более одной нагрузки одновременно на каждой нагружающей таблетке и завершайте её перед удалением таблетки. Во время запуска таблетка остаётся в фазе `Ready` и не обеспечивает исключительный доступ. Следующий запуск закрывает приём ввода-вывода, завершает принятые операции и перенастраивает те же акторы отдельных DBG до старта генераторов нагрузки.

Автоматизация через dstool

Для скриптов и возобновляемых запусков используйте `ydb-dstool cluster workload nbs-dbg-like`. Команда вызывает существующий legacy-метод gRPC `TestShardControl` и не переключается на HTTP. Сначала разверните серверную поддержку: сервис нагрузки на координаторе и узлах генераторов, а также таблетки с версией протокола автоматизации 1. Каждой команде нужны явно указанная база данных (или сохранённый дескриптор с ней) и права администратора. Проверка возможностей отклоняет неподдерживаемые серверы и устаревшие инкарнации координатора.

Legacy-сервис gRPC по умолчанию выключен на TLS-эндпоинтах и при явном списке сервисов, который его не включает. Добавьте `legacy` в `GRpcConfig.ServicesEnabled` в textproto или объедините следующую настройку с серверной конфигурацией YAML:

```
grpc_config:
  services_enabled:
    - legacy
```

Сохраните остальные включённые сервисы и убедитесь, что `legacy` отсутствует в `services_disabled`: этот список имеет приоритет. `test_shard` — отдельный сервис, который не включает данный legacy-RPC. Автоматизация не меняет настройки включения сервисов по умолчанию.

Используйте выделенные таблетки нагрузки. Сериализуйте создание и удаление для каждого индекса владельца в Hive, включая операции через HTTP. Резервируйте разные индексы для баз данных, использующих общий Hive. Одновременные изменения жизненного цикла независимыми контроллерами не поддерживаются. Координатор запрещает известные ему пересекающиеся запуски и удаление таблетки с известной активной нагрузкой; это не блокировка на уровне кластера. Не запускайте нагрузку через другой координатор одновременно на тех же таблетках.

Конфигурация и команды

Команда gRPC `create` строит `TAllocConfig` из параметров; файл выделения ресурсов не нужен. `--pool-name` задаёт пул для DDisk и persistent buffer, по умолчанию `ddp1`. Значения по умолчанию: `--num-groups` — 32, `--target-num-vchunks` — 1 vChunk на группу, `--vchunk-size-bytes` — 134217728 (128 МиБ), `--hosts-per-dbg` — 5. `TargetNumVChunks` также задаёт запрашиваемое у BSC число vChunk на группу; адресное пространство нагрузки на группу равно `TargetNumVChunks * VChunkSizeBytes`. Используйте пул и геометрию, подходящие вашему кластеру.

Если у базы нет пулов каналов таблетки по умолчанию, повторите `--tablet-storage-pool POOL` в порядке каналов. Для трёх каналов в одном обычном пуле хранения таблеток передайте `--tablet-storage-pool tablet-storage` трижды. Этот параметр отличается от `--pool-name`, который задаёт пулы DDisk/PB. Повторное создание использует существующие ресурсы только при совпадении базы/домена, эффективной конфигурации и фактических упорядоченных привязок; иначе возвращает конфликт. Предварительная проверка не делает создание атомарным относительно другого контроллера. Операция отправляет не более одного запроса создания или удаления; после неоднозначной ошибки транспорта проверьте индекс владельца и сообщённый идентификатор таблетки, прежде чем повторять операцию. Частичная ошибка сопровождается идентификатором таблетки для восстановления. Повторное удаление уже отсутствующей таблетки допустимо.

Команда `run` строит `TEvLoadTestRequest` из параметров; `--config` ей не нужен. Для существующей таблетки укажите `--tablet-id ID`, для нескольких повторите `--target TABLET_ID[@NODE_ID]`. Без цели `run` создаёт выделенную таблетку с теми же параметрами выделения ресурсов, что и `create`, ждёт готовности, выполняет нагрузку и удаляет таблетку после подтверждённого завершения. `--pool-name` задаёт общий пул DDisk/PB; `--ddisk-pool-name` и `--pb-pool-name` позволяют выбрать пулы раздельно, как на странице мониторинга. Без `@NODE_ID` используется текущее размещение Hive; `@0` запускает генератор на координаторе, а ненулевой ID явно задаёт узел. Идентификаторы таблеток должны быть уникальными. По умолчанию каждый запуск или испытание длится 10 секунд. Длительность должна быть положительной и превышать задержку перед измерениями. Для длительности, задержки, inflight и предела LSN CLI использует значения формы мониторинга; остальные пропущенные параметры нагрузки получают значения protobuf по умолчанию.

Без параметров inflight выполняется один запуск со значением 2048 на таблетку. Укажите `--inflight N` для другого одиночного значения или оба параметра `--inflight-from A --inflight-to B` для серии. Значения серии: A, 2A, 4A и так далее, пока значение не превысит B; промежуточное B не добавляется. `--trials` по умолчанию равно 1 и при значении больше 1 должно быть нечётным, как на странице мониторинга. CLI поддерживает те же цели на нескольких таблетках, долю чтений, размер ввода-вывода, префикс DBG, последовательные адреса, предел LSN, контрольные суммы, отключение репликации и выбор медианного испытания по write IOPS. Служебные теги назначаются сервисом.

Параметр	Поле protobuf	По умолчанию

`--duration-seconds`	`DurationSeconds`	10 секунд на запуск или испытание
`--delay-before-measurements-seconds`	`DelayBeforeMeasurementsSeconds`	0 секунд
`--num-groups-to-use`	`NumDirectBlockGroupsToUse`	0 (все группы)
`--inflight` или `--inflight-from` / `--inflight-to`	Значения `MaxInFlight` для испытаний	2048 на таблетку для одиночного запуска
`--read-ratio`	`ReadRatio`	0 чтений на 100 записей
`--sequential`	`Sequential`	Случайные адреса
`--read-write-size-kib`	`ReadWriteSizeKiB`	4 КиБ
`--stop-on-writes-done-count`	`StopOnWritesDoneCount`	0 (только длительность)
`--max-inflight-lsns`	`TabletConfig.MaxInflightLsns`	65536
`--flush-batch-size` / `--erase-batch-size`	Размеры пакетов таблетки	10000 каждый
`--sync-requests-batch-size`	`TabletConfig.SyncRequestsBatchSize`	10
`--pbuffer-reply-timeout-us`	`TabletConfig.PBufferReplyTimeoutMicroseconds`	50000 мкс
`--disable-replication`	`TabletConfig.DisableReplication`	Репликация включена; с этим флагом чтение запрещено
`--disable-checksums`	`TabletConfig.EnableChecksums`	Контрольные суммы включены

`--read-ratio` задаёт число чтений на 100 записей, а не процент всех операций; допустимые значения — от 0 до 100. `--read-write-size-kib` должен быть не меньше 4, кратен 4 и без остатка помещаться в выделенный vChunk. Ненулевое значение `--stop-on-writes-done-count` добавляет условие остановки; длительность по-прежнему ограничивает запуск.

Используйте обычные параметры TLS и аутентификации dstool и явный эндпоинт `grpc://` или `grpcs://`. Замените базу, эндпоинт, индекс владельца и идентификатор таблетки в примерах:

```
DSTOOL_ENDPOINT=grpcs://node.example:2135
ydb-dstool -e "$DSTOOL_ENDPOINT" cluster workload nbs-dbg-like create --database /Root/test --owner-index 1 --pool-name ddp1 --format json
ydb-dstool -e "$DSTOOL_ENDPOINT" cluster workload nbs-dbg-like list --database /Root/test --format json
ydb-dstool -e "$DSTOOL_ENDPOINT" cluster workload nbs-dbg-like describe --database /Root/test --owner-index 1 --format json
ydb-dstool -e "$DSTOOL_ENDPOINT" cluster workload nbs-dbg-like run --database /Root/test --tablet-id 72057594000000001 --duration-seconds 60 --inflight 32 --output-dir ./run-01 --format json
ydb-dstool -e "$DSTOOL_ENDPOINT" cluster workload nbs-dbg-like results --handle ./run-01 --wait --format json
ydb-dstool -e "$DSTOOL_ENDPOINT" cluster workload nbs-dbg-like stop --handle ./run-01 --format json
ydb-dstool -e "$DSTOOL_ENDPOINT" cluster workload nbs-dbg-like run --database /Root/test --duration-seconds 60 --inflight-from 1 --inflight-to 32 --trials 3 --output-dir ./sweep-01 --format jsonl
ydb-dstool -e "$DSTOOL_ENDPOINT" cluster workload nbs-dbg-like delete --database /Root/test --owner-index 1 --format json
```

`create` ожидает готовности. `run` по умолчанию ожидает завершения; `--no-wait` доступен только для одного запуска на явно заданной таблетке и возвращает дескриптор и путь к артефактам. `results` получает текущее состояние, а `--wait` ожидает конечного результата. `stop` останавливает один запуск и ожидает подтверждения завершения. Для отдельного испытания серии используйте `results` или `stop` с `--database`, `--node-id`, `--incarnation` и `--request-id` из checkpoint вместо `--handle`.

По умолчанию координатором становится принимающий узел; `--node-id` выбирает другой. Перед отправкой клиент фиксирует узел координатора и инкарнацию сервиса. Последующие запросы могут поступать через другой шлюз и перенаправляются координатору. Отсутствие сервиса нагрузки или контекста базы приводит к ошибке. Каждый новый запуск, включая каждое испытание sweep, запрашивает текущее размещение в Hive. Отсутствующий `NodeId` цели размещает генератор рядом с таблеткой; явно заданное ненулевое значение переопределяет размещение, а явный ноль выбирает узел координатора. Принятый запуск сохраняет снимок размещения при повторах и автоматически не переносится и не перезапускается.

Автоматизация ожидает готовности всего запрошенного префикса DBG и подтверждения применения конфигурации его акторами до генерации нагрузки. Она не использует HTTP-подстановку при нулевой готовности и не уменьшает нагрузку молча. Значение `NumDirectBlockGroupsToUse`, превышающее выделенное число DBG, отклоняется. Значения по умолчанию: тайм-аут RPC 90 секунд, бюджет запуска 60 секунд и интервал опроса 2 секунды; параметры — `--rpc-timeout`, `--startup-timeout` и `--poll-interval`. Бюджет запуска — единый срок, отсчитываемый с момента приёма START, который покрывает поиск размещения, проверку возможностей, готовность DBG и подтверждение конфигурации; повторный START его не продлевает, а его максимум составляет 3600 секунд. Тайм-аут RPC также передаётся шлюзу gRPC в поле `RpcTimeoutMs`. Общее ожидание runner по умолчанию равно бюджету запуска + длительности нагрузки + 210 секундам на завершение запросов и ограниченный транспортный запас; его переопределяет `--wait-timeout`. Подтверждённое завершение означает ожидание клиентских запросов, а не фоновых сбросов или удалений. Проверяйте `Run.TerminationConfirmed`: без подтверждения завершения запросов сервис оставляет запуск в `STOPPING` с `ExecutionError`, `TerminationConfirmed: false` и без `FinishedAtMs`, даже если watchdog генератора сообщил об ошибке. Такие неразрешённые запуски продолжают блокировать повторное использование своих таблеток на том же координаторе. Для таких запусков тайм-аут или прерывание клиента возвращает ненулевой код и неразрешённый исход; неуспешный или неподтверждённый исход не должен разрешать следующее испытание.

Один бюджет RPC действует на все попытки через шлюзы и ограничивается остатком общего бюджета операции; остаток передаётся и в gRPC, и в `RpcTimeoutMs`. Бюджет испытания начинается до START или восстановительного GET, ожидания

результата — до первого GET, готовности — до DESCRIBE, остановки — до STOP. Поздний конечный ответ сохраняется, но вердикт остаётся тайм-аутом. Отмена получает отдельный ограниченный бюджет.

Результаты, восстановление и sweep

`--format json` сохраняет машиночитаемый stdout; прогресс и диагностика поступают в stderr. Для серии формат pretty один раз показывает размещение таблеток, затем для каждого испытания выводит `MaxInFlight`, направление, IOPS, p50, p95, p99 в микросекундах, число измеренных ошибок и их процент. Строки чтения показываются, если чтения настроены или измерены. Результаты JSON и JSONL содержат `measured_io_errors` по направлениям: число ошибок и общее число операций строками, процент числом; при неуспешном вердикте добавляется `failure_reason`. Процент равен числу ошибок, делённому на сумму успешных и неуспешных измеренных операций данного направления. Protobuf JSON представляет 64-битные идентификаторы и счётчики строками. Результат CLI содержит `response.Run` и отдельный вердикт `passed` (для незавершённых результатов окончательного вердикта нет). Состояния выполнения: `IN_PROGRESS`, `STOPPING`, `SUCCEEDED`, `FAILED`, `CANCELLED`. `SUCCEEDED` означает завершение нагрузки. Частичные измеренные ошибки остаются видимыми, но не меняют вердикт CLI; направление не проходит проверку только при наличии ошибок и отсутствии успешных измеренных операций (100% ошибок). Ошибка выполнения, отмена, тайм-аут и потерянные/отсутствующие результаты по-прежнему приводят к ненулевому коду выхода. `--allow-io-errors` ослабляет только проверку 100% ошибок.

Типизированный результат содержит ошибки выполнения, эффективную конфигурацию с разрешённым размещением, сведения о сборке, временные метки, длительность измерения в миллисекундах, счётчики, байты, скорости, гистограммы задержек в микросекундах и статистику по таблеткам. Для общей задержки объединяются гистограммы, а не усредняются перцентили. Поля метрик браузера сохраняются как отдельное совместимое представление.

По умолчанию `run` сохраняет артефакты в `./nbs-dbg-like-load-results/<client-generated-uuid>/`. Явно заданный каталог не должен существовать, кроме случая возобновления. Созданный CLI файл `config.json` и атомарно записываемый `checkpoint.json` сохраняют настройки, координатор, идентификаторы запросов, цели и состояния испытаний до отправки. При автоматическом создании таблетки `auto.json` сохраняет индекс владельца и параметры выделения до CREATE. Каждый `result-NNNN.json` сохраняется до отметки о завершении испытания в checkpoint. Учётные данные в артефакты не попадают.

```
ydb-dstool -e "$DSTOOL_ENDPOINT" cluster workload nbs-dbg-like run --resume ./run-01 --format json
ydb-dstool -e "$DSTOOL_ENDPOINT" cluster workload nbs-dbg-like run --resume ./sweep-01 --format jsonl
```

Возобновление использует сохранённые настройки запуска; новые конфликтующие настройки отклоняются. `results --handle` и `stop --handle` также сохраняют конечный результат до отметки о завершении в checkpoint. Они восстанавливают прерванную запись результата и checkpoint и читают сохранённый конечный результат без сети, в том числе после потери истории координатора. Dry-run не изменяет артефакты; незавершённый ответ не помечается завершённым. Возобновление пропускает сохранённые завершённые испытания, запрашивает ранее отправленные и запускает только ещё не отправленные. Полностью завершённый запуск или sweep можно возобновить по сохранённым результатам без работающего координатора. Прерванная отправка неоднозначна: неизвестный или истёкший дескриптор никогда не должен автоматически приводить к новому запуску взамен старого. Принятая ограниченная нагрузка принадлежит сервису и продолжается после отключения клиента. Завершённые результаты и записи дедупликации хранятся в памяти сервера 24 часа; предел координатора — 1024 активные/сохранённые записи, после чего новые запуски отклоняются. Перезапуск координатора теряет историю и меняет инкарнацию. Потеря координатора не доказывает остановку удалённых генераторов; перед новой нагрузкой на те же таблетки выясните их состояние.

Серия сохраняет все значения удвоения inflight и все испытания, меняя только `WorkloadConfig.MaxInFlight`. Испытания последовательны, останавливаются при первом неуспешном вердикте и сохраняют `summary.json` с медианным испытанием по write IOPS. При тайм-ауте или прерывании runner запрашивает отмену конкретного запуска и подтверждает завершение до возможности продолжения; неподтверждённый исход остаётся неразрешённым. Автоматически созданная таблетка удаляется только после сохранения конечных результатов всех отправленных запусков с подтверждённым завершением; иначе её идентификатор остаётся в `auto.json` для восстановления. Существующие таблетки CLI автоматически не удаляет. Глобальный параметр `--dry-run` проверяет и показывает предполагаемые запросы без изменений и вымышленных результатов.

Для работы с агентом используйте специализированный навык ydb-nbs-dbg-like-load. Остальные разделы описывают совместимый HTTP-сценарий и общие параметры нагрузки.

Создание таблетки

Выберите отдельный `owner_idx` для каждой таблетки в выбранном Hive. Hive сопоставляет этот индекс вместе с фиксированным идентификатором владельца сервиса нагрузки с фактическим идентификатором таблетки. Сохраните оба значения: для удаления используется `owner_idx`, а для запуска нагрузки — идентификатор таблетки, назначенный Hive.

Сохраните следующую конфигурацию выделения ресурсов в файл `nbs-dbg-alloc.txt`. Замените имена пулов на настроенные в вашем кластере. Это сообщение `TAllocConfig` без внешнего блока `NbsDbgLikeLoad`.

```
DDiskPoolName: "ddp1"
PersistentBufferDDiskPoolName: "ddp1"
NumDirectBlockGroups: 1
TargetNumVChunks: 1
VChunkSizeBytes: 134217728
HostsPerDbg: 5
```

Создайте таблетку через форму на странице или отправьте тот же запрос:

```
curl --fail-with-body 'http://<node>:8765/actors/load' \
  -H 'Content-Type: application/x-protobuf-text' \
  --data-urlencode 'mode=tablet_create' \
  --data-urlencode 'owner_idx=1' \
  --data-urlencode 'config@nbs-dbg-alloc.txt'
```

Ответ представляет собой HTML с `TabletId=<id>`. Повторное создание инициализированной таблетки возвращает HTTP 409 с сообщением «Tablet was already initialized»; уже выделенные ресурсы не изменяются.

Чтобы явно задать привязки каналов таблетки, передайте в поле формы `storage_pools` список имён пулов, разделённых переводами строк, — по одному на канал. Например, добавьте `--data-urlencode 'storage_pools@tablet-storage-pools.txt'`. Если привязки не заданы явно, вспомогательный актор использует доступные пулы хранения тенанта для трёх

каналов, а при их отсутствии — привязку Hive по умолчанию. HTTP-обработчик читает это отдельное поле формы; одного `TabletStoragePools` в тексте конфигурации выделения ресурсов недостаточно, чтобы задать привязки Hive.

Параметры выделения ресурсов

Выделенные ресурсы сохраняются между запусками. Чтобы изменить их конфигурацию, удалите нагружающую таблетку и создайте её заново.

На странице таблетки поле `NumDirectBlockGroups` предварительно заполнено значением `32`. В таблице ниже указаны значения по умолчанию в protobuf, которые применяются, если поле не задано в сообщении конфигурации.

Параметр	По умолчанию	Значение
`DDiskPoolName`	`"ddp1"`	Пул BSC для данных DDisk.
`PersistentBufferDDiskPoolName`	`"ddp1"`	Пул BSC для размещения PB.
`NumDirectBlockGroups`	`1`	Число выделяемых логических DBG; должно быть положительным.
`TargetNumVChunks`	`1`	Число vChunk на DBG в адресном пространстве нагрузки и запрашиваемое число учитываемых чанков на стороне данных. Используйте положительное значение.
`VChunkSizeBytes`	`134217728`	Размер vChunk в байтах; используйте размер чанка, поддерживаемый кластером. Значение должно быть положительным и кратным 4096.
`HostsPerDbg`	`5`	Ожидаемое число пар DDisk/PB в каждом ответе на выделение ресурсов — от 3 до 5. Для реплицируемых записей и сброса данных нагрузка использует первые три пары. Поле проверяет ответ, а не настраивает геометрию пула BSC.
`TabletId`	Назначается таблеткой	Таблетка перезаписывает это поле собственным идентификатором, назначенным Hive, и использует его как идентификатор владельца хранилища. Не задавайте его во входных данных.
`TabletStoragePools`	Пусто	Сохраняемое поле конфигурации. При создании через HTTP передавайте привязки в отдельном поле формы `storage_pools`, как описано выше.

Проверка выделенных ресурсов

Обновите страницу таблеток или запросите их список:

```
curl --fail-with-body 'http://<node>:8765/actors/load?mode=tablet_list'
```

HTML-список объединяет информацию Hive с ответом `TEvNbsLoadTabletGetSummary` каждой таблетки. Он показывает размещение таблеток, пулы и число DBG. Это работает и после создания нескольких таблеток; отдельного сообщения для выделения ресурсов сразу нескольким таблеткам нет.

Сводка протокола акторов содержит число выделенных DBG, геометрию vChunk и `NumReadyDirectBlockGroups` — длину максимального непрерывного начального участка списка DBG, в каждой из которых готовы как минимум три PB. Для каждого PB рабочий актор выполняет `TEvConnect`, получает через `TEvGetPersistentBufferRegistrationToken` одноразовый токен, привязанный к таблетке, поколению и DBG, отправляет `TEvRegisterPersistentBuffer` с этим токеном и проверяет регистрацию через `TEvListPersistentBuffer`. Только успешное получение списка делает PB готовым; эта проверка также подтверждает продолжение обслуживания после отказа в повторной регистрации при переподключении. См. регистрацию PB. Создание завершается до окончания этих действий. Положительное число готовых DBG ограничивает адресное пространство запуска. Для совместимости текущий прокси нагрузки использует общее число выделенных DBG, если число готовых равно нулю, поэтому запуск сразу после создания всё ещё может приводить к ошибкам «peers not ready». Готовность PB также не гарантирует готовность всех соединений DDisk, необходимых для чтения и сброса данных.

Запуск нагрузки

Замените идентификатор таблетки в следующем примере значением, полученным при создании, и сохраните этот полный запрос к сервису нагрузки в файл `nbs-dbg-run.txt`:

```
NbsDbgLikeLoad {
  NbsDbgLikeTabletId: 72057594000000001
  WorkloadConfig {
    DurationSeconds: 60
    DelayBeforeMeasurementsSeconds: 15
    MaxInFlight: 32
    ReadRatio: 0
    Sequential: false
    ReadWriteSizeKiB: 4
    TabletConfig {
      MaxInflightLsns: 4096
      FlushBatchSize: 10000
      EraseBatchSize: 10000
      SyncRequestsBatchSize: 10
      PBufferReplyTimeoutMicroseconds: 50000
      EnableChecksums: true
    }
  }
}
```

Вставьте его в общую форму конфигурации нагружающих акторов и запустите на текущем узле или отправьте напрямую:

```
curl --fail-with-body 'http://<node>:8765/actors/load' \
  -H 'Content-Type: application/x-protobuf-text' \
```

```
  --data-urlencode 'mode=start' \
  --data-urlencode 'config@nbs-dbg-run.txt'
```

Убедитесь, что JSON-ответ содержит `"status": "ok"`, и сохраните назначенные `tag` и `uuid`. Этот обработчик возвращает HTTP 200 даже при ошибках разбора или запуска, поэтому одной проверки `curl --fail-with-body` недостаточно для подтверждения успеха. Ошибки инициализации рабочих акторов могут появиться позже в итоговом результате.

Запрашивайте завершённые результаты по полученному UUID:

```
curl --fail-with-body \
  -H 'Accept: application/json' \
  'http://<node>:8765/actors/load?mode=results&uuid=<uuid>'
```

Пустой массив результатов означает, что завершённый результат для этого UUID пока недоступен. Пока нагрузка выполняется, используйте JSON: прокси нагрузки NBS не обрабатывают запросы текущего состояния в HTML. После завершения тот же URL без JSON-заголовка показывает итоговый HTML-отчёт. Завершив запуск, можно начать следующий с другими настройками нагрузки, повторно используя выделенные ресурсы.

Параметры нагрузки

Эти поля находятся в `NbsDbgLikeLoad.WorkloadConfig`.

На странице таблетки поля `MaxInFlight` и `MaxInflightLsns` предварительно заполнены значениями `2048` и `65536`. HTTP-обработчик `tablet_run` использует те же значения, если параметры не заданы. В таблицах ниже указаны значения по умолчанию в protobuf для сообщений конфигурации.

Параметр	По умолчанию	Значение
`DurationSeconds`	`0`	Время от старта нагрузки до прекращения генерации запросов, включая прогрев. Задайте положительное значение здесь или в `StopOnWritesDoneCount`.
`DelayBeforeMeasurementsSeconds`	`15`	Интервал прогрева, исключаемый из результатов измерения. Должен быть меньше положительного `DurationSeconds`.
`MaxInFlight`	`32`	Общий предел одновременно выполняемых записей и чтений на целевую таблетку. Используйте положительное значение. При случайном доступе большие лимиты делятся между рабочими акторами; при последовательном доступе используется один рабочий актор.
`ReadRatio`	`0`	Число отправленных чтений на 100 записей, рассчитанное по числу отправленных запросов. `100` означает примерно одно чтение на запись, то есть половину всех запросов. Значения выше `100` допустимы: `200` означает примерно два чтения на запись. Этот параметр не задаёт нагрузку только на чтение.
`Sequential`	`false`	Последовательный обход плоского адресного пространства при true; иначе — равномерный случайный выбор.
`ReadWriteSizeKiB`	`4`	Фиксированный размер ввода-вывода, не менее 4 КиБ и кратный 4 КиБ. Он не должен превышать размер vChunk и должен делить его без остатка.
`NumDirectBlockGroupsToUse`	`0`	Использовать начальную часть списка доступных DBG. Ноль или значение больше числа доступных DBG означает использование всех доступных DBG.
`StopOnWritesDoneCount`	`0`	Целевое число успешно завершённых записей, распределяемое между рабочими акторами. Ноль отключает ограничение по числу записей. Также задавайте длительность: в текущей реализации неуспешные записи могут исчерпать лимит отправки до достижения целевого числа успешных записей.
`TabletConfig`	См. ниже	Конфигурация, устанавливаемая в акторах отдельных DBG таблетки перед генерацией нагрузки.

Завершения запросов во время прогрева и ожидания оставшихся ответов учитываются в счётчиках за всё время работы, но исключаются из измеряемых пропускной способности и задержки. Случайное чтение может обращаться к незаписанным адресам, а генератор не сравнивает возвращённые данные с ожидаемым содержимым блока. Для проверки корректности используйте интеграционные тесты.

Запросы PB и DDisk используют уникальные 64-битные идентификаторы vChunk: `ui64(DbgIndex) * TargetNumVChunks + localVChunkIndex`. Поэтому разные DBG хранят разные данные даже на общем DDisk при одинаковых локальных смещениях. Плоское адресное пространство клиента не меняется, а геометрия выделенных ресурсов остаётся неизменной между запусками. При переходе с прежнего отображения с локальными индексами DBG создайте новые нагружающие таблетки и выделите ресурсы заново: существующие данные не переносятся. См. описание реализации нагрузки.

Параметры таблетки

Эти поля находятся в `WorkloadConfig.TabletConfig`.

Параметр	По умолчанию	Значение
`MaxInflightLsns`	`4096`	Лимит отслеживаемых порядковых номеров записей журнала (LSN). Каждый актор DBG независимо применяет предел `max(1, budget / allocated_DBG_count)`. В делитель входят и неактивные DBG. Ноль запрещает записи; единого общего счётчика для лимита нет.

`FlushBatchSize`	`10000`	Максимальное число LSN за один проход сброса для каждого получателя, суммарно по пакетам, сгруппированным по vChunk; приводится к значению не менее одного.
`EraseBatchSize`	`10000`	Максимальное число LSN, включаемых в один пакет удаления для каждого получателя; приводится к значению не менее одного.
`SyncRequestsBatchSize`	`10`	Порог числа записей в текущей ещё не принятой когорте, готовой к сбросу или удалению, в отдельной DBG. Принимаются ровно её текущие участники. Более позднее завершение, в том числе с меньшим LSN, ждёт новую когорту или очистку при отсутствии активности, а уже принятые записи не учитываются в следующем пороге. Приводится к значению не менее одного. Установите `1`, чтобы остаток обрабатывался без задержки. Общий таймер очистки с задержкой в одну секунду принимает готовые записи в vChunk без активности независимо от порога, в том числе после остановки генерации нагрузки. Перенастройка, удаление и poison немедленно принимают все уже готовые записи независимо от числа и оставляют незавершённые записи PB вне когорты.
`PBufferReplyTimeoutMicroseconds`	`50000`	Тайм-аут, передаваемый координатору записи в несколько PB.
`DisableReplication`	`false`	Записывать только в PB-координатор, возвращать ответ после его подтверждения и удалять запись без копирования в DDisk. Требует `ReadRatio: 0`.
`EnableChecksums`	`true`	Вычислять в нагружающем акторе одну контрольную сумму на каждый блок полезной нагрузки размером 4 КиБ и передавать контрольные суммы через нагружающую таблетку в PersistentBuffer. Значение должно соответствовать режиму контрольных сумм DDisk/PersistentBuffer.

Прокси устанавливает `TabletConfig.IoSizeBytes` из `ReadWriteSizeKiB`, а `TabletConfig.NumDirectBlockGroupsToUse` — из выбранного числа DBG. Настраивайте их через поля нагрузки, а не через внутренние поля таблетки.

Задавайте лимит LSN на DBG со значительным запасом относительно порога синхронизации: LSN остаются под наблюдением на этапах записи, сброса и удаления, поэтому обычным когортам нужен запас ёмкости для прохождения порога без ожидания очистки при отсутствии активности.

Запуск на нескольких таблетках

Создайте каждую таблетку отдельно с собственным индексом владельца, затем получите их идентификаторы и текущие идентификаторы узлов из списка. Замените идентификаторы в примере:

```
NbsDbgLikeLoad {
  Targets { TabletId: 72057594000000001 NodeId: 1 }
  Targets { TabletId: 72057594000000002 NodeId: 2 }
  WorkloadConfig {
    DurationSeconds: 60
    DelayBeforeMeasurementsSeconds: 15
    MaxInFlight: 64
    ReadRatio: 100
    ReadWriteSizeKiB: 4
    TabletConfig { SyncRequestsBatchSize: 1 }
  }
}
```

Отправьте запрос тем же способом с `mode=start`. Непустой список `Targets` включает режим координатора и имеет приоритет над `NbsDbgLikeTabletId`. Каждая целевая таблетка получает полную конфигурацию нагрузки: в этом примере каждой задаётся предел в 64 одновременно выполняемых операции, то есть 128 в сумме. Если задан `StopOnWritesDoneCount`, он также применяется к каждой целевой таблетке отдельно.

`NodeId` выбирает узел для запуска прокси нагрузки соответствующей таблетки. Ноль или отсутствие значения означает узел координатора; актор не запрашивает и не отслеживает размещение в Hive. Используйте узел, на котором сейчас работает таблетка, чтобы генерация нагрузки выполнялась рядом с ней. Сервис нагрузки на целевом узле должен быть доступен. Указывайте каждую таблетку только один раз и запускайте этот координатор на одном узле.

Порядок ввода-вывода

Обычная запись отправляет `TEvWritePersistentBuffers` во все три основных PB. После трёх подтверждений клиент получает ответ, а фоновая операция Sync копирует данные в DDisk. Запись не использует DDisk `TEvWrite`; записи и чтения PB продолжаются параллельно с Sync. Незавершённая перезапись не скрывает предыдущую подтверждённую версию, а переставленные подтверждения не возвращают чтение к более старой версии.

Операции Sync пересекающихся записей выполняются в порядке приёма записей: новая версия ждёт завершения всех получателей предыдущей. В обычную когорту сброса и удаления входят ровно текущие ещё не принятые готовые записи. Пакеты Sync группируют их по получателю и vChunk: каждый запрос содержит один vChunk, а общий предел на получателя сохраняется для суммы пакетов. Более позднее завершение, в том числе с LSN меньше, чем у уже принятых записей, ждёт новую когорту или очистку при отсутствии активности. До завершения Sync чтение использует подтверждённую версию PB, после — DDisk. Незавершённое чтение PB удерживает эту версию от Erase, но не блокирует следующую допустимую Sync. Незавершённое чтение DDisk удерживает свой диапазон от пересекающейся Sync, но не блокирует запись PB. Пересекающиеся записи PB удаляются от старых к новым; независимые диапазоны могут обрабатываться параллельно.

Каждый актор DBG объединяет запросы очистки от готовых записей и завершений PB/Sync в один таймер с задержкой в одну секунду от первого запроса. При срабатывании он принимает готовые записи на сброс и удаление из локальных vChunk без активности, даже если порог не достигнут. Отсутствие активности означает отсутствие незавершённых записей PB и ожидающих ответа сегментов Sync для получателей; чтения и удаления этому не мешают. Оба прохода приёма используют одно состояние активности до запуска любой из очередей. Порядок операций и удержание данных чтениями сохраняются. Занятые vChunk и удерживаемые чтениями записи сами по себе не вызывают повторных проходов таймера: следующий проход запрашивают завершения PB/Sync, а последний читатель пробуждает только свой уже принятый слот. Запись, ставшая готовой незадолго до срабатывания общего таймера, может попасть в этот проход; непрерывная секунда без активности не требуется.

Нагрузка хранит состояние запросов до получения соответствующих ответов. Неуспешные записи не становятся видимыми, а подтверждённые копии PB удаляются после завершения оставшихся ответов записи. Отсутствующие или неоднозначные ответы не разрешают очистку или завершение ожидания операций жизненного цикла. Это моделирует обычный порядок операций раздела и штатное завершение, но не полное восстановление NBS после сбоя, маркеры или барьеры восстановления, ремонт данных или использование резервных узлов.

Результаты и освобождение ресурсов

Результаты содержат число запросов записи и чтения в секунду, перцентили p50/p95/p99 времени выполнения в микросекундах, заданный `max_in_flight`, общее число успешных запросов (`txs`), пропускную способность (`rps`) и число ошибок в секунду (`errors`). Результаты для нескольких таблеток также содержат массив `tablets`. Координатор объединяет гистограммы задержек, а не усредняет перцентили отдельных таблеток.

Момент завершения записи — получение подтверждения PB, ещё до фонового копирования в DDisk и удаления из PB. Результаты чтения объединяют обращения как к PB, так и к DDisk. Последний HTML-отчёт содержит счётчики, суммарные объёмы в байтах, длительность измерений и перцентили задержки. Счётчики таблетки и отдельных DBG показывают активность записи, сброса, удаления и чтения через сервис счётчиков `load_actor`. Показатель жизненного цикла `DbgsAllocated` обновляется при смене фазы: после выделения он содержит число выделенных DBG, после удаления — ноль.

Ограничение длительности, достижение целевого числа записей или Stop прекращает отправку новых запросов и даёт до 30 секунд на получение оставшихся клиентских ответов. Это ожидание не форсирует фоновый сброс или удаление. Очистка по таймеру с задержкой в одну секунду позволяет небольшим остаткам завершиться после остановки нагрузки, в том числе удалению при отключённой репликации, но завершение клиентской нагрузки не ждёт этих фоновых операций. Уже принятые записи остаются принятыми и не учитываются в следующей когорте.

Следующая настройка закрывает приём запросов и завершает принятые записи через Sync, ждёт принятые чтения и удаляет копии PB до установки параметров. Прокси запускает генераторы только после подтверждения настройки всеми акторами отдельных DBG. Перенастройка сохраняет соединения с узлами хранения. Ожидание операций жизненного цикла и poison аннулируют обычные таймеры очистки через внутреннее поколение, независимое от пользовательских идентификаторов конфигурации. Устаревшее событие не влияет на новую конфигурацию и не сбрасывает признак её ожидающего таймера. Принудительная очистка при операциях жизненного цикла и очистка при отсутствии активности не увеличивают счётчики блокировки обычным порогом. Удаление и poison рабочего актора также завершают принятые операции, затем ждут подтверждения отключения от узлов. Это штатное ожидание рабочего актора не имеет предельной длительности: неразрешённый результат может удерживать эти операции бесконечно; тайм-аут клиента не доказывает, что очистка закончилась.

После завершения нагрузки удалите каждую таблетку по индексу владельца, с которым она была создана:

```
curl --fail-with-body 'http://<node>:8765/actors/load' \
  -H 'Content-Type: application/x-protobuf-text' \
  --data-urlencode 'mode=tablet_delete' \
  --data-urlencode 'owner_idx=1'
```

Вспомогательный актор запрашивает у таблетки завершение операций и освобождение DBG, а затем удаление из Hive. Таблетка отправляет запрос освобождения в BSC только после завершения операций и подтверждения отключения всех акторов отдельных DBG. Такое удаление нагружающей таблетки сначала сбрасывает подтверждённые записи, затем очищает копии PB; это более сильная гарантия, чем разрушительное удаление раздела в рабочем NBS. Операция освобождает выделенные ресурсы, но не обеспечивает гарантированного стирания данных. Если выделение или удаление завершается ошибкой, перед повторной попыткой изучите журналы BSC и таблетки: текущая реализация не полностью восстанавливает прерванное выделение ресурсов и может очистить локальное состояние после ошибки освобождения в BSC.

Исходный код и описание реализации

- Текущая конфигурация и протокол акторов.
- Руководство по реализации нагружающего актора, включая алгоритм нагрузки, ограничения жизненного цикла и тестовые цели.
- Группы прямого блочного доступа.
- Обзор тестирования с помощью нагружающих акторов.

VDiskLoad

Подает нагрузку write-only на VDisk. Имитирует Distributed Storage Proxy. Результатом теста является производительность записи на VDisk в операциях в секунду.

Параметры актора

Ниже описаны основные параметры актора. Полный список параметров смотрите в файле load_test.proto Git-репозитория YDB.

Параметр	Описание
`VDiskId`	Параметры VDisk, от имени которого подается нагрузка. • `GroupID` — идентификатор группы. • `GroupGeneration` — поколение группы. • `Ring` — идентификатор кольца в группе. • `Domain` — идентификатор фэйл-домена в кольце. • `VDisk` — индекс VDisk в фэйл-домене.
`GroupInfo`	Описание группы, в которую входит нагружаемый VDisk (в корректном поколении).
`TabletId`	Идентификатор таблетки, от имени которой подается нагрузка. Должен быть уникальным для каждого нагружающего актора.
`Channel`	Номер канала внутри таблетки, который будет указан в командах записи блобов и сборки мусора.
`DurationSeconds`	Полная длительность теста в секундах, по достижению которой нагрузка автоматически прекращается.
`WriteIntervals`	Описание параметров вероятностного распределения временных интервалов между записями.
`WriteSizes`	Размер записываемых данных. Для каждого запроса выбирается случайным образом из интервала `Min` - `Max`. Вы можете задать несколько диапазонов `WriteSizes`, и тогда выбор значения из конкретного диапазона будет определяться его `Weight`.
`InFlightPutsMax`	Максимальное количество одновременно выполняемых запросов на запись блоба в VDisk (TEvVPut-запросы); если не указан, то число запросов не ограничивается.
`InFlightPutBytesMax`	Максимальное количество байт в одновременно выполняемых запросах на запись блоба в VDisk (TEvVPut-запросы).
`PutHandleClass`	Класс записи данных в дисковую подсистему. В случае `TabletLog` запись выполняется с максимальным приоритетом.
`BarrierAdvanceIntervals`	Описание параметров вероятностного распределения интервалов между передвижением барьера сборки мусора и шага записи.
`StepDistance`	Расстояние между текущим записываемым шагом `Gen:Step` блоба и собираемым. Чем больше эта величина, тем больше данных хранится. Запись происходит с `Step = X`, удаление — всех блобов со `Step = X - StepDistance`. При этом периодически (с периодом `BarrierAdvanceIntervals`) `Step` увеличивается на единицу.

Параметры вероятностного распределения

Интервал, записанный в виде repeated-поля `TIntervalInfo`, вычисляется по следующему алгоритму:

- случайным образом выбирается элемент из массива `TIntervalInfo` с вероятностью, пропорциональной весу;
- для элемента типа `TIntervalUniform` равновероятно выбирается значение в интервале `Min-Max` (если указано `MinMs/MaxMs`, то значение в миллисекундах; если `MinUs/MaxUs` — в микросекундах);
- для элемента типа `TIntervalPoisson` выбирается интервал по формуле `Min(log(-x / Frequency), MaxIntervalMs)`, где `x` — случайное значение в интервале `[0, 1]`, что обеспечивает соответствие интервалов распределению Пуассона с частотой `Frequency`, но с интервалом не больше `MaxIntervalMs`.

Аналогичный механизм применяется и для вероятностного распределения размеров записываемых данных, за исключением того, что в том механизме есть только равновероятное распределение размера `[Min, Max]`.

Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

PDiskWriteLoad

Тестирует производительность записи на PDisk. Нагрузка подается от имени VDisk. Актор создает на указанном PDisk чанки и записывает в них случайные данные. После снятия нагрузки записанные актором данные удаляются.

Вы можете подать нагрузку двух видов:

- *Постоянная* — актор следит, чтобы одновременно было запущено указанное число запросов. Чтобы подать постоянную нагрузку, задайте нулевую паузу между запросами (например, `IntervalMsMin: 0`, `IntervalMsMax: 0`) и отличный от нуля `InFlightWrites`.
- *Интервальная* — актор запускает запросы через заданные промежутки времени. Чтобы подать интервальную нагрузку, задайте ненулевую паузу между запросами (например, `IntervalMsMin: 10`, `IntervalMsMax: 100`). Максимальное количество одновременно выполняемых запросов задается параметром `InFlightWrites`. Если его значение равно `0`, то ограничения нет.

Параметры актора

Ниже описаны основные параметры актора. Полный список параметров смотрите в файле load_test.proto Git-репозитория YDB.

Параметр	Описание
`PDiskId`	Идентификатор нагружаемого PDisk на узле.
`PDiskGuid`	Глобально-уникальный идентификатор нагружаемого PDisk.
`VDiskId`	Нагрузка подается от имени VDisk со следующими реквизитами: • `GroupID` — идентификатор группы. • `GroupGeneration` — поколение группы. • `Ring` — идентификатор кольца в группе. • `Domain` — идентификатор фэйл-домена в кольце. • `VDisk` — индекс VDisk в фэйл-домене.
`Chunks`	Параметры чанка. `Slots` — количество слотов в чанке, определяет размер записи. Вы можете указать несколько `Chunks`, и тогда выбор конкретного чанка для записи будет определяться его `Weight`.
`DurationSeconds`	Продолжительность нагрузки в секундах.
`IntervalMsMin`, `IntervalMsMax`	Минимальный и максимальный промежутки времени между запросами при интервальной нагрузке в миллисекундах. Значение промежутка выбирается случайно из указанного диапазона.
`InFlightWrites`	Количество одновременно обрабатываемых запросов на запись.
`LogMode`	Режим записи лога. В режиме `LOG_SEQUENTIAL` сначала происходит запись в чанк, а после ее подтверждения запись — в лог.
`Sequential`	Тип записи. • `True` — последовательная. • `False` — случайная.
`IsWardenlessTest`	Если PDiskReadLoad запускается на кластере, укажите `False`. Иначе (например, при запуске в юнит-тестах) укажите `True`.

Примеры

Следующий актор пишет данные блоками по `32` МБ, в течение `120` секунд, одновременно выполняются `64` запроса (постоянная нагрузка):

```
PDiskWriteLoad: {
    PDiskId: 1000
    PDiskGuid: 2258451612736857634
    VDiskId: {
        GroupID: 11234
        GroupGeneration: 5
        Ring: 1
        Domain: 1
        VDisk: 3
    }
    Chunks: { Slots: 4096 Weight: 1 }
    Chunks: { Slots: 4096 Weight: 1 }
    Chunks: { Slots: 4096 Weight: 1 }
    Chunks: { Slots: 4096 Weight: 1 }
    Chunks: { Slots: 4096 Weight: 1 }
    Chunks: { Slots: 4096 Weight: 1 }
    Chunks: { Slots: 4096 Weight: 1 }
    Chunks: { Slots: 4096 Weight: 1 }
    DurationSeconds: 120
    IntervalMsMin: 0
    IntervalMsMax: 0
    InFlightWrites: 64
    LogMode: LOG_SEQUENTIAL
    Sequential: false
    IsWardenlessTest: false
}
```

При просмотре результата тестирования наибольший интерес представляют следующее значение:

- `Average speed since start` — средняя скорость записи с момента запуска в МБ/с, например `615.484013`.

Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

PDiskReadLoad

Тестирует производительность чтения с PDisk. Нагрузка подается от имени VDisk. Актор создает на указанном PDisk чанки, записывает в них случайные данные и выполняет чтение из них с указанными параметрами. После снятия нагрузки записанные актором данные удаляются.

Вы можете подать нагрузку двух видов:

- *Постоянная* — актор следит, чтобы одновременно было запущено указанное число запросов. Чтобы подать постоянную нагрузку, задайте нулевую паузу между запросами (например, `IntervalMsMin: 0`, `IntervalMsMax: 0`) и отличный от нуля `InFlightReads`.
- *Интервальная* — актор запускает запросы через заданные промежутки времени. Чтобы подать интервальную нагрузку, задайте ненулевую паузу между запросами (например, `IntervalMsMin: 10`, `IntervalMsMax: 100`). Максимальное количество одновременно выполняемых запросов задается параметром `InFlightReads`. Если его значение равно `0`, то ограничения нет.

Параметры актора

Ниже описаны основные параметры актора. Полный список параметров смотрите в файле load_test.proto Git-репозитория YDB.

Параметр	Описание
`PDiskId`	Идентификатор нагружаемого PDisk на узле.
`PDiskGuid`	Глобально-уникальный идентификатор нагружаемого PDisk.
`VDiskId`	Нагрузка подается от имени VDisk со следующими реквизитами: • `GroupID` — идентификатор группы. • `GroupGeneration` — поколение группы. • `Ring` — идентификатор кольца в группе. • `Domain` — идентификатор фэйл-домена в кольце. • `VDisk` — индекс VDisk в фэйл-домене.
`Chunks`	Параметры чанка. `Slots` — количество слотов в чанке, определяет размер записи. Вы можете указать несколько `Chunks`, и тогда выбор конкретного чанка для чтения будет определяться его `Weight`.
`DurationSeconds`	Продолжительность нагрузки в секундах.
`IntervalMsMin`, `IntervalMsMax`	Минимальный и максимальный промежутки времени между запросами при интервальной нагрузке в миллисекундах. Значение промежутка выбирается случайно из указанного диапазона.
`InFlightReads`	Количество одновременно обрабатываемых запросов на чтение.
`Sequential`	Тип чтения. • `True` — последовательное. • `False` — случайное.
`IsWardenlessTest`	Если PDiskReadLoad запускается на кластере, укажите `False`. Иначе (например, при запуске в юнит-тестах) укажите `True`.

Примеры

Следующий актор читает данные блоками по `32` МБ, в течение `120` секунд, одновременно выполняются `64` запроса (постоянная нагрузка):

```
PDiskReadLoad: {
    PDiskId: 1000
    PDiskGuid: 2258451612736857634
    VDiskId: {
        GroupID: 11234
        GroupGeneration: 5
        Ring: 1
        Domain: 1
        VDisk: 3
    }
    Chunks: { Slots: 4096 Weight: 1 }
    Chunks: { Slots: 4096 Weight: 1 }
    Chunks: { Slots: 4096 Weight: 1 }
    Chunks: { Slots: 4096 Weight: 1 }
    Chunks: { Slots: 4096 Weight: 1 }
    Chunks: { Slots: 4096 Weight: 1 }
    Chunks: { Slots: 4096 Weight: 1 }
    Chunks: { Slots: 4096 Weight: 1 }
    DurationSeconds: 120
    IntervalMsMin: 0
    IntervalMsMax: 0
    InFlightReads: 64
    Sequential: false
    IsWardenlessTest: false
}
```

При просмотре результата тестирования наибольший интерес представляет следующее значение:

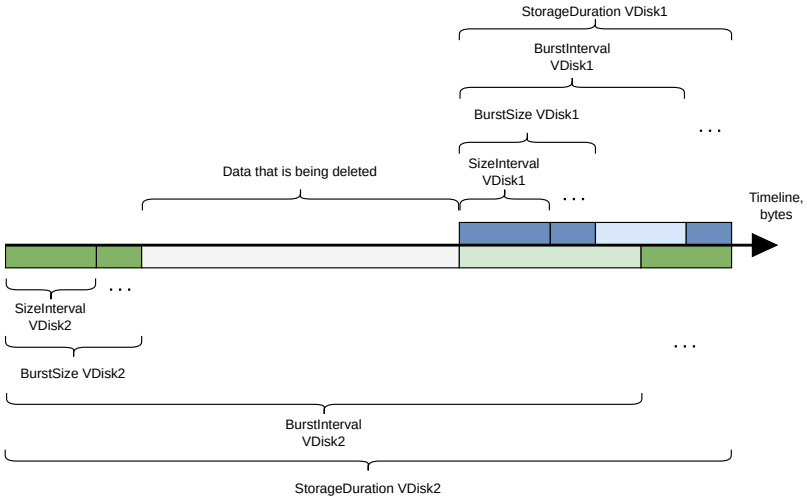
- `Average speed since start` — средняя скорость чтения с момента запуска в МБ/с, например `1257.148154`.

Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

PDiskLogLoad

Все VDisk, размещенные на некотором PDisk, пишут данные о своей работе в общий лог PDisk. VDisk постепенно удаляют свои устаревшие данные в начале лога для освобождения места на диске. В некоторых случаях между окончанием записи одного VDisk и началом записи другого может образоваться участок, содержащий ненужные устаревшие данные. Такие данные автоматически удаляются. Корректность этой операции тестирует актор PDiskLogLoad.



Примечание

Это узкоспециализированный актор для тестирования конкретной функциональности. Не является нагружающим, его назначение — проверка корректности работы.

Параметры актора

Ниже описаны основные параметры актора. Полный список параметров смотрите в файле load_test.proto Git-репозитория YDB.

Параметр	Описание
`PDiskId`	Идентификатор нагружаемого PDisk на узле.
`PDiskGuid`	Глобально-уникальный идентификатор нагружаемого PDisk.
`VDiskId`	Параметры VDisk, от имени которого подается нагрузка. • `GroupID` — идентификатор группы. • `GroupGeneration` — поколение группы. • `Ring` — идентификатор кольца в группе. • `Domain` — идентификатор фэйл-домена в кольце. • `VDisk` — индекс VDisk в фэйл-домене.
`MaxInFlight`	Количество одновременно обрабатываемых запросов.
`SizeIntervalMin`	Минимальный размер записи в лог в байтах.
`SizeIntervalMax`	Максимальный размер записи в лог в байтах.
`BurstInterval`	Интервал между сеансами записи в лог в байтах.
`BurstSize`	Общее количество данных, которое будет записано в одном сеансе, в байтах.
`StorageDuration`	Виртуальное время в байтах. Показывает, как долго VDisk должен хранить свои данные в логе.
`IsWardenlessTest`	Если PDiskReadLoad запускается на кластере, укажите `false`. Иначе (например, при запуске в юнит-тестах) укажите `true`.

Примеры

Следующий актор имитирует работу двух VDisk. Первый VDisk через каждые `65536` байт записывает в лог сообщение размером `65536` байт, удаляет данные свыше `1048576` байт. Второй через каждые `2147483647` байт записывает `1024` байт сообщениями по `128` байт, удаляет данные свыше `2147483647` байт.

```
PDiskLogLoad: {
    Tag: 1
    PDiskId: 1
    PDiskGuid: 12345
    DurationSeconds: 60
    Workers: {
        VDiskId: {GroupID: 1 GroupGeneration: 5 Ring: 1 Domain: 1 VDisk: 1}
        MaxInFlight: 1
        SizeIntervalMin: 65536
        SizeIntervalMax: 65536
        BurstInterval: 65536
```

```
        BurstSize: 65536
        StorageDuration: 1048576
    }
    Workers: {
        VDiskId: {GroupID: 2 GroupGeneration: 5 Ring: 1 Domain: 1 VDisk: 1}
        MaxInFlight: 1
        SizeIntervalMin: 128
        SizeIntervalMax: 128
        BurstInterval: 2147483647
        BurstSize: 1024
        StorageDuration: 2147483647
    }
    IsWardenlessTest: false
}
```

Тест завершился успешно, если в течение проверки все узлы кластера работали без перезагрузок и тестируемый PDisk имел состояние `Normal`. Убедиться в этом можно с помощью YDB UI кластера.



Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

MemoryLoad

Аллоцирует блоки памяти указанного размера через заданные промежутки времени. После снятия нагрузки аллоцированная память освобождается. С помощью этого актора вы можете протестировать работу логики, например срабатывание некоторого триггера при достижении лимита по RSS.

Примечание

Это узкоспециализированный актор для тестирования конкретной функциональности. Не является нагружающим, его назначение — проверка корректности работы.

Параметры актора

Параметр	Описание
`DurationSeconds`	Продолжительность нагрузки в секундах.
`BlockSize`	Размер аллоцируемого блока в байтах.
`IntervalUs`	Интервал времени между аллоцированиями блоков в микросекундах.

Примеры

Следующий актор аллоцирует блоки по `1048576` байт каждые `9000000` микросекунд в течение `3600` секунд и за время работы займет 32 ГБ:

```
MemoryLoad: {
    DurationSeconds: 3600
    BlockSize: 1048576
    IntervalUs: 9000000
}
```

Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

Stop

С помощью этой команды можно остановить всю или только указанную нагрузку.

Параметры актора

Параметр	Описание
`Tag`	Тег нагружающего актора, который нужно остановить. Тег можно посмотреть в YDB UI кластера.
`RemoveAllTags`	При значении параметра `True` будут остановлены все нагружающие акторы.

Примеры

Следующая команда остановит нагрузку с тегом `123`:

```
Stop: {
    Tag: 123
}
```

Команда для остановки всей нагрузки:

```
Stop: {
    RemoveAllTags: true
}
```



Хотите присоединиться к команде разработки YDB?

Ознакомьтесь с разделами о команде YDB и открытых вакансиях, а также о возможностях для студентов.

Хаос-тестирование

Хаос-тестирование — это методология проверки устойчивости YDB путём намеренного внесения сбоев в работающий кластер. Цель — убедиться, что YDB корректно переживает реальные отказы: потери узлов, сетевые разделения, проблемы с дисками и другие нештатные ситуации. За внесение этого хаоса отвечает инструмент **Nemesis**.

Что проверяется

В рамках хаос-тестирования YDB проверяется поведение кластера при следующих видах отказов:

Сетевые сбои

- Изоляция узлов от сети (блокировка входящего и исходящего трафика)
- Сдвиг системного времени на узлах (time skew)

Отказы узлов

- Принудительное завершение процессов узлов кластера
- Остановка и повторный запуск узлов
- Приостановка (suspend) процессов узлов

Отказы таблеток

Таблетки — это основные вычислительные единицы YDB. Проверяется устойчивость при принудительном завершении различных типов таблеток:

- **Coordinator** — координатор распределённых транзакций
- **Hive** — менеджер размещения таблеток
- **BSController** — контроллер распределённого хранилища
- **SchemeShard** — менеджер схемы данных
- **DataShard** — таблетки хранения данных
- **Mediator** — медиатор транзакций
- **PersQueue** — таблетки очередей сообщений
- И другие системные таблетки

Также проверяется перебалансировка таблеток между узлами через Hive.

Дисковые сбои

- Безопасное выведение диска из строя на узле
- Очистка дисков на узлах

Сценарии для мультидатацентровых кластеров

- Остановка всех узлов в одном датацентре
- Сетевая изоляция датацентра

Сценарии для кластеров с bridge-режимом

- Остановка всех узлов в одном pile
- Сетевая изоляция pile

Интеграция с нагрузочным тестированием

Хаос-тестирование обычно запускается совместно с нагрузочными тестами из директории ydb/tests/stress. Такое сочетание позволяет тестировать кластер в условиях нагрузки при одновременном возникновении различных сбоев.

Как работает проверка

Во время внесения сбоев система по запросу проверяет два свойства:

- **Живучесть (liveness)** — кластер остаётся доступным и продолжает обрабатывать запросы
- **Безопасность (safety)** — в логах и метриках кластера не появляются признаки нарушения корректности данных или внутренних инвариантов системы

Сбои вносятся по расписанию автоматически, а результаты проверок агрегируются и доступны для анализа.

Инструмент Nemesis

Для проведения хаос-тестирования YDB используется инструмент **Nemesis** — приложение для внесения сбоев, расположенное в репозитории YDB на GitHub. Он разворачивается непосредственно на узлах тестируемого кластера и управляет внесением сбоев по заданному расписанию.

> **Важно**
>
> Nemesis работает только с кластерами YDB, которые были развёрнуты с помощью утилиты `ydbd_slice`.

Установка

Разверните Nemesis на кластере:

```
# Однофайловая конфигурация (cluster.yaml содержит и хосты, и шаблон базы данных)
nemesis install --yaml-config-location /path/to/cluster.yaml

# Двухфайловая конфигурация (отдельные cluster.yaml и databases.yaml)
nemesis install \
    --yaml-config-location /path/to/cluster.yaml \
    --database-config-location /path/to/databases.yaml
```

Первый хост в конфигурации кластера становится оркестратором, остальные — агентами. Сервисы разворачиваются как systemd-юниты и запускаются автоматически.

Основной способ наблюдать за работой Nemesis и анализировать результаты — это веб-интерфейс, работающий на оркестраторе. Откройте в браузере URL, который был напечатан в конце выполнения `nemesis install`; по умолчанию он доступен по адресу:

```
http://<orchestrator_host>:31434/static/index.html
```

В UI отображаются:

- **Активные сбои** — текущие внесённые сбои и логи их работы, сгруппированные по категориям (сеть, узлы, таблетки, диски, датацентры, pile)
- **Расписания** — типы сбоев, зарегистрированные для автоматического внесения, их интервалы и время следующего запуска
- **Ручное управление** — кнопки для внесения конкретных сбоев по требованию
- **История выполнения** — прошедшие сбои с временными метками, целевыми хостами
- **Liveness-проверки** — кластерные проверки здоровья, выполняемые оркестратором
- **Safety-проверки** — детекторы нарушений за последние 24 часа по локальным логам

Те же данные доступны через HTTP API оркестратора, если нужно собирать их программно.

Интерпретация результатов

Результат запуска Nemesis оценивается по двум свойствам: **живучесть (liveness)** и **безопасность (safety)**. Критерии успеха и отказа:

- **Успех (Pass)** — в течение всего запуска все liveness-проверки сообщают о здоровом кластере, и все safety-warden'ы возвращают пустые списки нарушений
- **Отказ (Fail)** — хотя бы одна liveness-проверка сообщила о недоступности кластера (кластер не восстановился после сбоя за ожидаемое время) **либо** хотя бы один safety-warden вернул непустой список нарушений (например, ошибки или assertion'ы в логах кластера, рассогласования SchemeShard, нарушения инвариантов DataShard)

В UI успешные проверки отображаются с пустым списком нарушений, а отказавшие — с сообщениями о нарушениях, сформированными соответствующим warden'ом.

Чтобы разобраться с проблемой, откройте отказавшую проверку в UI и прочитайте список сообщений о нарушениях.

Описание конкретных проверок и шаблонов, которые ищет каждый warden, см. в README Nemesis.

Остановка сервисов

Остановите все сервисы Nemesis на кластере:

```
nemesis stop --yaml-config-location /path/to/cluster.yaml
```

Расширение Nemesis

Если нужно добавить новый тип хаоса (сбоя) или новую проверку safety/liveness, см. README. Там описано, как реализовать и зарегистрировать новые раннеры сбоев, планировщики и классы warden'ов.

Справка по YDB

В этом разделе документации содержится справочная информация по различным аспектам YDB, включая:

- YQL - Обзор
- Использование YDB UI
- Интеграции YDB
- YDB CLI
- Справка по YDB SDK
- Языки и API
- Kafka API
- Amazon SQS API
- Параметры конфигурации кластера
- Справка по наблюдаемости YDB
- Обзор утилиты YDB DSTool
- Обзор утилиты ydbops
- Справка по работе YDB в Docker

YQL - Обзор

YQL (YDB Query Language) — универсальный декларативный язык запросов к YDB, диалект SQL. YQL создавался для работы с большими распределенными базами данных, и поэтому обладает рядом отличий от стандарта SQL.

Инструменты работы с YDB поддерживают интерфейсы отправки YQL-запросов и получения результатов их исполнения:

- YDB CLI
- YDB SDK

В данном разделе документации находится справочник по YQL, который включает разделы:

- Типы данных с описанием типов данных, применяемых в YQL
- Синтаксис с полным перечнем команд YQL
- Встроенные функции с описанием доступных встроенных функций

Также вы можете пройти серию уроков, знакомящих вас с основными командами YQL, в разделе Туториал YQL.

YQL - Обзор

YQL (YDB Query Language) — универсальный декларативный язык запросов к YDB, диалект SQL. YQL создавался для работы с большими распределенными базами данных, и поэтому обладает рядом отличий от стандарта SQL.

Инструменты работы с YDB поддерживают интерфейсы отправки YQL-запросов и получения результатов их исполнения:

- YDB CLI
- YDB SDK

В данном разделе документации находится справочник по YQL, который включает разделы:

- Типы данных с описанием типов данных, применяемых в YQL
- Синтаксис с полным перечнем команд YQL
- Встроенные функции с описанием доступных встроенных функций

Также вы можете пройти серию уроков, знакомящих вас с основными командами YQL, в разделе Туториал YQL.

Типы данных YQL

В данном разделе размещены статьи по типам данных YQL:

- Простые/примитивные типы
- Серийные типы
- Опциональные типы
- Контейнеры
- Специальные типы
- Преобразования типов
- Текстовое представление типов данных
- Представление данных в формате JSON
- Представление данных в формате YSON

Примитивные типы данных

Термины «простые», «примитивные» и «элементарные» типы данных используются как синонимы.

Числовые типы

Тип	Описание	Примечания
`Bool`	Логическое значение.	
`Int8`	Целое число со знаком. Допустимые значения: от -2^7 до 2^7-1.	
`Int16`	Целое число со знаком. Допустимые значения: от -2^{15} до $2^{15}-1$.	
`Int32`	Целое число со знаком. Допустимые значения: от -2^{31} до $2^{31}-1$.	
`Int64`	Целое число со знаком. Допустимые значения: от -2^{63} до $2^{63}-1$.	
`Uint8`	Беззнаковое целое число. Допустимые значения: от 0 до 2^8-1.	
`Uint16`	Беззнаковое целое число. Допустимые значения: от 0 до $2^{16}-1$.	
`Uint32`	Беззнаковое целое число. Допустимые значения: от 0 до $2^{32}-1$.	
`Uint64`	Беззнаковое целое число. Допустимые значения: от 0 до $2^{64}-1$.	
`Float`	Вещественное число с переменной точностью размером 4 байта.	Не может использоваться в первичном ключе и в колонках, формирующих ключ вторичного индекса
`Double`	Вещественное число с переменной точностью размером 8 байт.	Не может использоваться в первичном ключе и в колонках, формирующих ключ вторичного индекса
`Decimal(precision, scale)`	Вещественное число с фиксированной точностью размером 16 байт. Точность (precision) — максимальное общее число хранимых десятичных разрядов, принимает значения от 1 до 35. Масштаб (scale) — максимальное число хранимых десятичных разрядов справа от десятичной запятой, принимает значения от 0 до значения precision.	
`DyNumber`	Бинарное представление вещественного числа точностью до 38 знаков. Допустимые значения: положительные от 1×10^{-130} до $1\times10^{126}-1$, отрицательные от $-1\times10^{126}-1$ до -1×10^{-130} и 0. Совместим с типом `Number` AWS DynamoDB. Не рекомендуется для использования в ydb-native приложениях.	

Примеры

```
SELECT
  Bool("true"),
  Uint8("0"),
  Int32("-1"),
  Uint32("2"),
  Int64("-3"),
  Uint64("4"),
  Float("-5"),
  Double("6"),
  Decimal("1.23", 5, 2), -- до 5 десятичных знаков, из которых 2 после запятой
  DyNumber("1"),
  String("foo"),
  Utf8("привет"),
  Yson("<a=1>[3;%false]"),
  Json(@@{"a":1,"b":null}@@),
  Date("2017-11-27"),
  Datetime("2017-11-27T13:24:00Z"),
  Timestamp("2017-11-27T13:24:00.123456Z"),
  Interval("P1W2DT2H3M4.567890S"),
  TzDate("2017-11-27,Europe/Moscow"),
  TzDatetime("2017-11-27T13:24:00,America/Los_Angeles"),
  TzTimestamp("2017-11-27T13:24:00.123456,GMT"),
  Uuid("f9d5cc3f-f1dc-4d9c-b97e-766e57ca4ccb");
```

Строковые типы

Тип	Описание	Примечания

`String`	Строка, может содержать произвольные бинарные данные	
`Bytes` (псевдоним типа `String`)	Строка, может содержать произвольные бинарные данные	
`Utf8`	Текст в кодировке UTF-8	
`Text` (псевдоним типа `Utf8`)	Текст в кодировке UTF-8	
`Json`	JSON в текстовом представлении	Не поддерживает возможность сравнения, не может использоваться в первичном ключе и в колонках, формирующих ключ вторичного индекса
`JsonDocument`	JSON в бинарном индексированном представлении	Не поддерживает возможность сравнения, не может использоваться в первичном ключе и в колонках, формирующих ключ вторичного индекса
`Yson`	YSON в текстовом или бинарном представлении	Не поддерживает возможность сравнения, не может использоваться в первичном ключе и в колонках, формирующих ключ вторичного индекса
`Uuid`	Универсальный идентификатор UUID	

Ограничения на размер

Максимальный размер значения в ячейке неключевого столбца с любым строковым типом данных — 8 МБ.

В отличие от типа данных `Json`, который хранит исходное текстовое представление, переданное пользователем, `JsonDocument` использует бинарное индексированное представление. Важное отличие с точки зрения семантики состоит в том, что `JsonDocument` не сохраняет форматирование, порядок ключей в объектах и их дубликаты.

За счет индексированного представления `JsonDocument` позволяет обходить документную модель с использованием `JsonPath` без необходимости парсинга всего содержимого. Это позволяет эффективно выполнять операции из JSON API, уменьшая задержки и стоимость пользовательских запросов. Выполнение запросов над `JsonDocument` может быть до нескольких раз эффективнее в зависимости от типа нагрузки.

Из-за добавленной избыточности `JsonDocument` менее эффективен в хранении. Дополнительные накладные расходы на хранение зависят от конкретного содержимого и в среднем составляют 20–30% от исходного объема. Сохранение данных в формате `JsonDocument` требует дополнительной конвертации из текстового представления, что делает его запись менее эффективной. Тем не менее, для большинства read-intensive сценариев, подразумевающих обработку данных из JSON, этот тип данных является предпочтительным и рекомендуется к использованию.

Важно

Для хранения чисел (JSON Number) в `JsonDocument`, а также для арифметических операций над ними в JSON API используется тип Double. Возможна потеря точности при использовании нестандартных представлений чисел в исходном JSON-документе.

Дата и время

Тип	Описание	Возможные значения	Размер (байты)	Примечания
`Date`	Момент времени, соответствующий полуночи[1] по UTC, точность до дней	с 00:00 01.01.1970 до 00:00 01.01.2106	2	—
`Date32`	Момент времени, соответствующий полуночи[1] по UTC, точность до дней	с 00:00 01.01.144169 до н.э. по 00:00 01.01.148107 н.э.	4	—
`Datetime`	Момент времени по UTC, точность до секунд	с 00:00 01.01.1970 до 00:00 01.01.2106	4	—
`Datetime64`	Момент времени по UTC, точность до секунд	с 00:00 01.01.144169 до н.э. по 00:00 01.01.148107 н.э.	8	—
`Timestamp`	Момент времени по UTC с точностью до микросекунд	с 00:00 01.01.1970 до 00:00 01.01.2106	8	—
`Timestamp64`	Момент времени по UTC с точностью до микросекунд	с 00:00 01.01.144169 до н.э. по 00:00 01.01.148107 н.э.	8	—
`Interval`	Интервал времени, точность до микросекунд	от -136 лет до +136 лет	8	
`Interval64`	Интервал времени, точность до микросекунд	от -292277 лет до +292277 лет	8	
`TzDate`	Момент времени по UTC, соответствующий полуночи в заданной таймзоне	с 00:00 01.01.1970 до 00:00 01.01.2106		Не поддерживается в столбцах таблиц

`TzDate32`	Момент времени по UTC, соответствующий полуночи в заданной таймзоне	с 00:00 01.01.144169 до н.э. по 00:00 01.01.148107 н.э.	4 и метка таймзоны	—
`TzDateTime`	Момент времени по UTC с меткой временной зоны и точностью до секунд	с 00:00 01.01.1970 до 00:00 01.01.2106		Не поддерживается в столбцах таблиц
`TzDateTime64`	Момент времени по UTC с меткой временной зоны и точностью до секунд	с 00:00 01.01.144169 до н.э. по 00:00 01.01.148107 н.э.	8 и метка таймзоны	
`TzTimestamp`	Момент времени по UTC с меткой временной зоны и точностью до микросекунд	с 00:00 01.01.1970 до 00:00 01.01.2106		Не поддерживается в столбцах таблиц
`TzTimestamp64`	Момент времени по UTC с меткой временной зоны и точностью до микросекунд	с 00:00 01.01.144169 до н.э. по 00:00 01.01.148107 н.э.	8 и метка таймзоны	—

[1] Под полночью понимается момент времени, в котором все компоненты *времени* равны нулю.

Особенности поддержки типа Interval

Тип `Interval` использует синтаксис, основанный на стандарте ISO 8601, с рядом особенностей:

- поддерживаются только интервалы, которые можно представить в виде целого числа микросекунд (не поддерживаются интервалы в годах `Y` и месяцах `M` из-за их переменной продолжительности);
- не поддерживаются варианты с началом/концом интервала, а также повторами;
- поддерживается использование отрицательного знака для обозначения сдвига в прошлое;
- поддерживается совмещение литерала недели `W` с другими литералами;
- микросекунды могут быть указаны в виде дробной части секунд.

Примеры использования

```
SELECT
    Interval("P1W2DT2H3M4.567890S"), -- интервал 1 неделя 2 дня 2 часа 3 минуты 4.567890 секунды
    Interval("P1W"),                 -- интервал 1 неделя (7 дней)
    Interval("-P1D");                -- интервал в прошлое 1 сутки (24 часа)
```

Особенности поддержки типов с меткой временной зоны

Метка временной зоны у типов `TzDate`, `TzDatetime`, `TzTimestamp` это атрибут, который используется:

- При преобразовании (CAST, DateTime::Parse, DateTime::Format) в строку и из строки.
- В DateTime::Split - появляется компонент таймзоны в `Resource<TM>`.

Само значение позиции во времени у этих типов хранится в UTC, и метка таймзоны никак не участвует в прочих расчётах. Например:

```
SELECT -- эти выражения всегда true для любых таймзон: таймзона не влияет на точку во времени.
    AddTimezone(CurrentUtcDate(), "Europe/Moscow") ==
        AddTimezone(CurrentUtcDate(), "America/New_York"),
    AddTimezone(CurrentUtcDatetime(), "Europe/Moscow") ==
        AddTimezone(CurrentUtcDatetime(), "America/New_York");
```

Важно понимать, что при преобразованиях между `TzDate` и `TzDatetime` или `TzTimestamp` дате соответствует не полночь по локальному времени таймзоны, а полночь по UTC для даты в UTC.

Приведение простых типов данных

Явное приведение

Явное приведение при помощи CAST:

Приведение к числовым типам

Тип	Bool	Int8	Int16	Int32	Int64	Uint8	Uint16	Uint32	Uint64	Float	Double	Decimal
Bool	—	Да[1]	Да[1]	Да[1]	Да[1]	Да[1]	Да[1]	Да[1]	Да[1]	Да[1]	Да[1]	Нет
Int8	Да[2]	—	Да	Да	Да	Да[3]	Да[3]	Да[3]	Да[3]	Да	Да	Да
Int16	Да[2]	Да[4]	—	Да	Да	Да[3,4]	Да[3]	Да[3]	Да[3]	Да	Да	Да
Int32	Да[2]	Да[4]	Да[4]	—	Да	Да[3,4]	Да[3,4]	Да[3]	Да[3]	Да	Да	Да
Int64	Да[2]	Да[4]	Да[4]	Да[4]	—	Да[3,4]	Да[3,4]	Да[3,4]	Да[3]	Да	Да	Да
Uint8	Да[2]	Да[4]	Да	Да	Да	—	Да	Да	Да	Да	Да	Да
Uint16	Да[2]	Да[4]	Да[4]	Да	Да	Да[4]	—	Да	Да	Да	Да	Да
Uint32	Да[2]	Да[4]	Да[4]	Да[4]	Да	Да[4]	Да[4]	—	Да	Да	Да	Да
Uint64	Да[2]	Да[4]	Да[4]	Да[4]	Да[4]	Да[4]	Да[4]	Да[4]	—	Да	Да	Да
Float	Да[2]	Да[4]	Да[4]	Да[4]	Да[4]	Да[3,4]	Да[3,4]	Да[3,4]	Да[3,4]	—	Да	Нет

Double	Да[2]	Да[4]	Да[4]	Да[4]	Да[4]	Да[3,4]	Да[3,4]	Да[3,4]	Да[3,4]	Да	—	Нет
Decimal	Нет	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да	—
String	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
Bytes	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
Utf8	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
Text	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
Json	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет
Yson	Да[5]	Да[5]	Да[5]	Да[5]	Да[5]	Да[5]	Да[5]	Да[5]	Да[5]	Да[5]	Да[5]	Нет
Uuid	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет
Date	Нет	Да[4]	Да[4]	Да	Да	Да[4]	Да	Да	Да	Да	Да	Нет
Datetime	Нет	Да[4]	Да[4]	Да[4]	Да	Да[4]	Да[4]	Да	Да	Да	Да	Нет
Timestamp	Нет	Да[4]	Да[4]	Да[4]	Да[4]	Да[4]	Да[4]	Да[4]	Да	Да	Да	Нет
Interval	Нет	Да[4]	Да[4]	Да[4]	Да	Да[3,4]	Да[3,4]	Да[3,4]	Да[3]	Да	Да	Нет
Date32	Нет	Да[4]	Да[4]	Да	Да	Да[4]	Да	Да	Да	Да	Да	Нет
Datetime64	Нет	Да[4]	Да[4]	Да[4]	Да	Да[4]	Да[4]	Да	Да	Да	Да	Нет
Timestamp64	Нет	Да[4]	Да[4]	Да[4]	Да[4]	Да[4]	Да[4]	Да[4]	Да	Да	Да	Нет
Interval64	Нет	Да[4]	Да[4]	Да[4]	Да	Да[3,4]	Да[3,4]	Да[3,4]	Да[3]	Да	Да	Нет

[1] `True` преобразуется в `1`, `False` преобразуется в `0`.
[2] Любое значение кроме `0` преобразуется в `True`, `0` преобразуется в `False`.
[3] Возможно только в случае неотрицательного значения.
[4] Возможно только в случае попадания в диапазон допустимых значений.
[5] При помощи встроенной функции Yson::ConvertTo.

Приведение к типам данных даты и времени

Тип	Date	Datetime	Timestamp	Interval	Date32	Datetime64	Timestamp64	Interval64
Bool	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет
Int8	Да	Да	Да	Да	Да	Да	Да	Да
Int16	Да	Да	Да	Да	Да	Да	Да	Да
Int32	Да	Да	Да	Да	Да	Да	Да	Да
Int64	Да	Да	Да	Да	Да	Да	Да	Да
Uint8	Да	Да	Да	Да	Да	Да	Да	Да
Uint16	Да	Да	Да	Да	Да	Да	Да	Да
Uint32	Да	Да	Да	Да	Да	Да	Да	Да
Uint64	Да	Да	Да	Да	Да	Да	Да	Да
Float	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет
Double	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет
Decimal	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет
String	Да	Да	Да	Да	Да	Да	Да	Да
Bytes	Да	Да	Да	Да	Да	Да	Да	Да
Utf8	Да	Да	Да	Да	Да	Да	Да	Да
Text	Да	Да	Да	Да	Да	Да	Да	Да
Json	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет
Yson	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет
Uuid	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет
Date	—	Да	Да	Нет	Да	Да	Да	Нет
Datetime	Да	—	Да	Нет	Да	Да	Да	Нет
Timestamp	Да	Да	—	Нет	Да	Да	Да	Нет
Interval	Нет	Нет	Нет	—	Нет	Нет	Нет	Да
Date32	Да	Да	Да	Нет	—	Да	Да	Нет
Datetime64	Да	Да	Да	Нет	Да	—	Да	Нет

Timestamp64	Да	Да	Да	Нет	Да	Да	—	Нет
Interval64	Нет	Нет	Нет	Да	Нет	Нет	Нет	—

Приведение к другим типам данных

Тип	String	Bytes	Utf8	Text	Json	Yson	Uuid
Bool	Да	Да	Нет	Нет	Нет	Нет	Нет
Int8	Да	Да	Нет	Нет	Нет	Нет	Нет
Int16	Да	Да	Нет	Нет	Нет	Нет	Нет
Int32	Да	Да	Нет	Нет	Нет	Нет	Нет
Int64	Да	Да	Нет	Нет	Нет	Нет	Нет
Uint8	Да	Да	Нет	Нет	Нет	Нет	Нет
Uint16	Да	Да	Нет	Нет	Нет	Нет	Нет
Uint32	Да	Да	Нет	Нет	Нет	Нет	Нет
Uint64	Да	Да	Нет	Нет	Нет	Нет	Нет
Float	Да	Да	Нет	Нет	Нет	Нет	Нет
Double	Да	Да	Нет	Нет	Нет	Нет	Нет
Decimal	Да	Да	Нет	Нет	Нет	Нет	Нет
String	—	—	Да	Да	Да	Да	Да
Bytes	—	—	Да	Да	Да	Да	Да
Utf8	Да	Да	—	—	Нет	Нет	Нет
Text	Да	Да	—	—	Нет	Нет	Нет
Json	Да	Да	Да	Да	—	Нет	Нет
Yson	Да[1]	Да[1]	Нет	Нет	Нет	Нет	Нет
Uuid	Да	Да	Да	Да	Нет	Нет	—
Date	Да	Да	Да	Да	Нет	Нет	Нет
Datetime	Да	Да	Да	Да	Нет	Нет	Нет
Timestamp	Да	Да	Да	Да	Нет	Нет	Нет
Interval	Да	Да	Да	Да	Нет	Нет	Нет
Date32	Да	Да	Да	Да	Нет	Нет	Нет
Datetime64	Да	Да	Да	Да	Нет	Нет	Нет
Timestamp64	Да	Да	Да	Да	Нет	Нет	Нет
Interval64	Да	Да	Да	Да	Нет	Нет	Нет

[1] С помощью встроенной функции Yson::ConvertTo.

Примеры

```
SELECT
    CAST("12345" AS Double),               -- 12345.0
    CAST(1.2345 AS Uint8),                 -- 1
    CAST(12345 AS String),                 -- "12345"
    CAST("1.2345" AS Decimal(5, 2)),       -- 1.23
    CAST("xyz" AS Uint64) IS NULL,         -- true, так как не удалось
    CAST(-1 AS Uint16) IS NULL,            -- true, отрицательное в беззнаковое
    CAST([-1, 0, 1] AS List<Uint8?>),            -- [null, 0, 1]
        --Тип элемента опциональный: неудачный элемент в null.
    CAST(["3.14", "bad", "42"] AS List<Float>),   -- [3.14, 42]
        --Тип элемента не опциональный: неудачный элемент удалён.
    CAST(255 AS Uint8),                    -- 255
    CAST(256 AS Uint8) IS NULL;            -- true, выходит за диапазон
```

Неявное приведение

Неявное приведение типов, которое возникает в базовых операциях (`+`, `-`, `*`, `/`, `%`) между разными типами данных. В ячейках таблицы указан тип результата операции, если она возможна:

Числовые типы

При несовпадении числовых типов сначала выполняется BITCAST обоих аргументов к типу результата, а потом уже операция.

Тип	Int8	Int16	Int32	Int64	Uint8	Uint16	Uint32	Uint64	Float	Double

Int8	—	`Int16`	`Int32`	`Int64`	`Int8`	`Uint16`	`Uint32`	`Uint64`	`Float`	`Double`
Int16	`Int16`	—	`Int32`	`Int64`	`Int16`	`Int16`	`Uint32`	`Uint64`	`Float`	`Double`
Int32	`Int32`	`Int32`	—	`Int64`	`Int32`	`Int32`	`Int32`	`Uint64`	`Float`	`Double`
Int64	`Int64`	`Int64`	`Int64`	—	`Int64`	`Int64`	`Int64`	`Int64`	`Float`	`Double`
Uint8	`Int8`	`Int16`	`Int32`	`Int64`	—	`Uint16`	`Uint32`	`Uint64`	`Float`	`Double`
Uint16	`Uint16`	`Int16`	`Int32`	`Int64`	`Uint16`	—	`Uint32`	`Uint64`	`Float`	`Double`
Uint32	`Uint32`	`Uint32`	`Int32`	`Int64`	`Uint32`	`Uint32`	—	`Uint64`	`Float`	`Double`
Uint64	`Uint64`	`Uint64`	`Uint64`	`Int64`	`Uint64`	`Uint64`	`Uint64`	—	`Float`	`Double`
Float	`Float`	`Float`	`Float`	`Float`	`Float`	`Float`	`Float`	`Float`	—	`Double`
Double	`Double`	`Double`	`Double`	`Double`	`Double`	`Double`	`Double`	`Double`	`Double`	—

Типы даты и времени

Тип	Date	Datetime	Timestamp	Interval	TzDate	TzDatetime	TzTimestamp	Date32	Datetime64	Timestamp64
Date	—	`DateTime`	`Timestamp`	—	`TzDate`	`TzDatetime`	`TzTimestamp`	`Date32`	`DateTime64`	`Timestamp64`
Datetime	—	—	`Timestamp`	—	—	`TzDatetime`	`TzTimestamp`	—	`Datetime64`	`Timestamp64`
Timestamp	—	—	—	—	—	—	`TzTimestamp`	—	—	`Timestamp64`
Interval	—	—	—	—	—	—	—	—	—	—
TzDate	—	—	—	—	—	`TzDatetime`	`TzTimestamp`	—	—	—
TzDatetime	—	—	—	—	—	—	`TzTimestamp`	—	—	—
TzTimestamp	—	—	—	—	—	—	—	—	—	—
Date32	—	—	—	—	—	—	—	—	`DateTime64`	`Timestamp64`
Datetime64	—	—	—	—	—	—	—	—	—	`Timestamp64`
Timestamp64	—	—	—	—	—	—	—	—	—	—
Interval64	—	—	—	—	—	—	—	—	—	—
TzDate32	—	—	—	—	—	—	—	—	—	—
TzDatetime64	—	—	—	—	—	—	—	—	—	—
TzTimestamp64	—	—	—	—	—	—	—	—	—	—

Серийные типы данных

Серийные типы данных представляют собой целые числа, но с дополнительным механизмом генерации значений. Эти типы данных используются для создания автоинкрементных колонок, а именно для каждой новой строки, добавляемой в таблицу, будет автоматически генерироваться уникальное значение для такой колонки (подобно типу SERIAL в PostgreSQL или свойству AUTO_INCREMENT в MySQL).

> **Примечание**
>
> Не рекомендуется использовать серийные типы данных в качестве первичного ключа: монотонно возрастающие значения приводят к неравномерному распределению данных и перегрузке отдельных партиций. Подробнее — Выбор первичного ключа для максимальной производительности.

Пример использования

```
CREATE TABLE users (
    user_id Serial,
    name Utf8,
    email Utf8,
    PRIMARY KEY (user_id)
);
```

```
UPSERT INTO users (name, email) VALUES ('Alice', 'alice@example.com');
INSERT INTO users (name, email) VALUES ('Bob', 'bob@example.com');
REPLACE INTO users (name, email) VALUES ('John', 'john@example.com');
```

```
SELECT * FROM users;
```

email	name	user_id
`alice@example.com`	Alice	1
`bob@example.com`	Bob	2
`john@example.com`	John	3

Можно самостоятельно указать значение `Serial` колонки при вставке, в этом случае вставка будет выполняться, как с обычной целочисленной колонкой, и `Sequence` затрагиваться при таком запросе никак не будет:

```
UPSERT INTO users (user_id, name, email) VALUES (4, 'Peter', 'peter@example.com');
```

Описание

Только колонки, участвующие в первичном ключе таблиц, могут иметь тип `Serial`.

При определении такого типа для колонки создаётся отдельный схемный объект `Sequence`, привязанный к этой колонке и являющийся генератором последовательности, из которого извлекаются значения. Этот объект является приватным и скрыт от пользователя. `Sequence` будет уничтожен вместе с таблицей.

Значения последовательности начинаются с единицы, выдаются с шагом, равным единице, и ограничены в зависимости от используемого типа.

Тип	Максимальное значение	Тип значения
`SmallSerial`		`Int16`
`Serial2`		`Int16`
`Serial`		`Int32`
`Serial4`		`Int32`
`Serial8`		`Int64`
`BigSerial`		`Int64`

При переполнении `Sequence` на вставке будет возвращаться ошибка:

```
Error: Failed to get next val for sequence: /dev/test/users/_serial_column_user_id, status: SCHEME_ERROR
    <main>: Error: sequence [OwnerId: <some>, LocalPathId: <some>] doesn't have any more values available, co
de: 200503
```

Отметим, что следующее значение выдаётся генератором до непосредственной вставки в таблицу и уже будет считаться использованным, даже если строка, содержащая это значение, не была успешно вставлена, например, при откате транзакции. Поэтому множество значений такой колонки может содержать пропуски и состоять из нескольких промежутков.

Для таблиц с автоинкрементными колонками поддержаны операции copy, dump, restore и import/export.

Типы данных, допускающие значение NULL

Любые типизированные данные в YQL, включая столбцы таблиц, бывают как гарантированно имеющие значение, так и потенциально пустые (что обозначается как `NULL`). Типы данных, которые могут содержать значения `NULL`, называются *опциональными* или, в терминах SQL, — *nullable*.

Опциональные типы данных в текстовом виде обозначаются вопросительным знаком в конце (например, `String?`) или как `Optional<...>`.
Наиболее часто на опциональных типах данных выполняются следующие операции:

- IS NULL - проверка на пустое значение
- COALESCE - оставить заполненные значения без изменений, а `NULL` заменить на указанное следом значение по умолчанию
- UNWRAP - извлечь значение оригинального типа из опционального, `T?` преобразуется в `T`
- JUST — добавить опциональность к текущему типу, `T` преобразуется в `T?`
- NOTHING — создать пустое значение с указанным типом.

`Optional` (nullable) является не свойством типа данных или колонки, а одним из видов контейнеров, которые могут быть произвольным образом вложены друг в друга. Так, например, столбец с типом `Optional<Optional<Boolean>>` может принимать 4 значения - `NULL` всего контейнера, `NULL` внутреннего контейнера, `TRUE` и `FALSE`. Описанный тип отличается от `List<List<Boolean>>` тем, что роль пустого списка в нем играет `NULL` и отсутствует возможность положить больше одного содержательного элемента. Также значения типа `Optional<Optional<T>>` возвращаются в качестве результата поиска по ключу в словаре `Dict(k,v)` со значениями типа `Optional<T>`. Такой тип данных результата позволяет отличать лежащий в словаре `NULL` от ситуации отсутствия ключа.

Примечание

По умолчанию, при указании примитивного типа `T` как типа колонки таблицы YDB в базе данных создается соответсвующий контейнерный тип `Optional<T>`. Колонку с типом `T` можно создать, используя ключевое слово `NOT NULL`.
В явном виде контейнерные типы (включая контейнеры `Optional<T>` и производные от них более сложные типы) в настоящее время нельзя использовать в качестве типов данных колонок при создании таблиц YDB.

Запросы YQL могут возвращать значения контейнерных типов, а также принимать их в качестве входных параметров.

Пример

```
$dict = {"a":1, "b":null};
$found = $dict["b"];
select if($found is not null, unwrap($found), -1);
```

Результат:

```
# column0
null
```

Логические и арифметические операции с NULL

Литерал `NULL` имеет отдельный сингулярный тип `Null` и может быть неявно сконвертирован к любому опциональному типу (в том числе и вложенному `Optional<Optional<...Optional<T>...>>`). В ANSI SQL `NULL` имеет семантику "неизвестное значение" – поэтому логические и арифметические операции с `NULL` или с незаполненными `Optional` имеют некоторые особенности.

Примеры

```
SELECT
    True OR NULL,        -- Just(True) (работает как True OR <неизвестное значение типа Bool>)
    False AND NULL,      -- Just(False)
    True AND NULL,       -- NULL   (точнее Nothing<Bool?> – <неизвестное значение типа Bool>)
    NULL OR NOT NULL,    -- NULL   (все NULL-ы "разные")
    1 + NULL,            -- NULL   (Nothing<Int32?>) - результат сложения 1 с
                         --         неизвестным значением типа Int)
    1 == NULL,           -- NULL   (результат сравнения 1 с неизвестным значением типа Int)
    (1, NULL) == (1, 2), -- NULL   (сравнение композитных элементов производится покомпонентно
                         --         через `AND`)
    (2, NULL) == (1, 3), -- Just(False) (выражение эквивалентно 2 == 1 AND NULL == 3)
```

Типы данных, не допускающие значение NULL

Примитивные типы YQL не могут хранить в себе значение `NULL`: для хранения `NULL` предназначен описанный выше контейнер `Optional`. В терминах SQL примитивные типы YQL являются *non-nullable* типами.

В YQL отсутствует неявное преобразование типов из `Optional<T>` в `T`, поэтому выполнимость `NOT NULL` ограничения на колонку таблицы обеспечивается на этапе компиляции запроса YDB.

Создать non-nullable колонку в таблице YDB можно с помощью операции CREATE TABLE, пользуясь ключевым словом `NOT NULL`.

Пример

```
CREATE TABLE t (
    Key Uint64 NOT NULL,
```

```
    Value String NOT NULL,
    PRIMARY KEY (Key))
```

После этого операции записи в таблицу `t` будут выполняться, только если среди значений на вставку в колонки `key`, `value` будут отсутствовать значения NULL.

Пример взаимодействия NOT NULL ограничения с функциями YQL

Многие из функций YQL имеют опциональные типы в качестве возвращаемого значения. Так как YQL является строго типизированным языком, запрос вида

```
CREATE TABLE t (
    c Utf8 NOT NULL,
    PRIMARY KEY (c)
);

INSERT INTO t(c)
SELECT CAST('q' AS Utf8);
```

не может быть исполнен. Причиной этому является несоответствие типов колонки `c`, имеющей тип `Utf8`, и результата функции `CAST`, имеющей тип `Optional<Utf8>`. Для корректной работы запроса в таких сценариях требуется использовать функцию COALESCE, аргументом которой можно указать fallback-значение для вставки в таблицу для случая, когда функция (в примере `CAST`) вернет пустой `Optional`. Если же в случае пустого `Optional` нужно не выполнять вставку, а вернуть ошибку, то для этого подойдет функция UNWRAP, позволяющая распаковать содержимое опционального типа.

Контейнеры

YQL поддерживает контейнерные типы для определения сложных структур данных различной организации.
Значения контейнерных типов могут быть переданы в запросы YQL в качестве входных параметров, либо возвращены из запросов YQL в качестве выходных колонок набора результатов.
Контейнерные типы не могут быть использованы в качестве типов данных колонок таблиц.

Тип	Объявление, пример	Описание
Список	`List<Type>`, `List<Int32>`	Последовательность переменной длины, состоящая из элементов одного типа.
Словарь	`Dict<KeyType, ValueType>`, `Dict<String,Int32>`	Набор пар ключ—значение с фиксированным типом ключей и значений.
Множество	`Set<KeyType>`, `Set<String>`	Набор элементов с фиксированным типом, является частным случаем словаря с типом значения `Void`.
Кортеж	`Tuple<Type1, ..., TypeN>`, `Tuple<Int32,Double>`	Набор безымянных элементов фиксированной длины с указанными типами всех элементов.
Структура	`Struct<Name1:Type1, ..., NameN:TypeN>`, `Struct<Name:String,Age:Int32>`	Набор именованных полей с указанными типами значений, фиксированный на момент начала запроса (то есть обязательно не зависящий от данных).
Поток	`Stream<Type>`, `Stream<Int32>`	Однопроходной итератор по значениям одного типа. Не является сериализуемым.
Вариант над кортежем	`Variant<Type1, Type2>`, `Variant<Int32,String>`	Кортеж, про который известно, что заполнен ровно один элемент.
Вариант над структурой	`Variant<Name1:Type1, Name2:Type2>`, `Variant<value:Int32,error:String>`	Структура, про которую известно, что заполнен ровно один элемент.
Перечисление	`Enum<Name1, Name2>`, `Enum<value,error>`	Контейнер, в котором выбран ровно один элемент перечисления, который определяется только своим именем.

При необходимости контейнеры можно вкладывать друг в друга в произвольных комбинациях, например `List<Tuple<Int32,Int32>>`.

Опциональные значения в некоторых контекстах также могут рассматриваться как один из видов контейнеров (`Optional<Type>`), который ведёт себя как список длины 0 или 1.

Для создания литералов контейнеров списка, словаря, множества, кортежа, структуры можно использовать операторную запись.
Для создания литерала варианта над кортежем или структурой используется функция Variant.
Для создания литерала перечисления используется функция Enum.

Для обращения к элементам контейнера используется точка или квадратные скобки, в зависимости от его типа.

Специальные типы данных

Тип	Описание
`Callable`	Вызываемое значение, которое можно исполнить, передав аргументы в круглых скобках в SQL-синтаксисе YQL.
`Resource`	Непрозрачный указатель на ресурс, который можно передавать между пользовательскими функциями (UDF, user defined function). Тип возвращаемого и принимаемого ресурса объявляется внутри функции строковой меткой. При передаче ресурса YQL проверяет совпадение меток, чтобы предотвратить передачу ресурсов между несовместимыми функциями. В случае несовпадения меток происходит ошибка типизации.
`Tagged`	Возможность дать прикладное имя какому-либо другому типу.
`Generic`	Тип данных у типа данных.
`Unit`	Тип данных у невычисляемых сущностей (источники и приемники данных, атомы и т. п.).
`Null`	Сингулярный тип данных с единственным возможным значением null. Является типом литерала `NULL` и может преобразовываться к любому `Optional` типу.
`Void`	Сингулярный тип данных с единственным возможным значением `null`.
`EmptyList`	сингулярный тип данных с единственным возможным значением []; является типом литерала `[]` и может преобразовываться к любому `List` типу.
`EmptyDict`	сингулярный тип данных с единственным возможным значением {}; является типом литерала `{}` и может преобразовываться к любому `Dict` или `Set` типу.

Правила преобразования типов через оператор CAST

Правила преобразования примитивных типов данных

- В процессе преобразования примитивных типов данных часть исходной информации может быть отброшена, если она не содержится в целевом типе. Например:
  - Дробная часть `Float` / `Double` при преобразовании в целочисленные типы;
  - Время `Datetime` / `Timestamp` при преобразовании в `Date`.
  - Таймзона при преобразовании из типов с таймзоной к типу даты/времени без таймзоны.
- Если для определённого сочетания исходного и целевого типа преобразование не может быть выполнено для всех возможных значений исходного типа, то при неудачном преобразовании `CAST` вернёт `NULL`. В таких случаях к типу возвращаемого значения добавляется один уровень `Optional`, если его не было. Например, конструкции: `CAST("3.14" AS Float?)` и `CAST("3.14" AS Float)` полностью эквиваленты и возвращают `Float?`.
- Если же преобразование возможно для всех значений исходного типа, то добавление '?' работает как `Just` сверху: `CAST(3.14 AS Utf8?)` то же, что и `Just(CAST(3.14 AS Utf8))`

Все сочетания примитивных типов, для которых возможен `CAST` описаны тут.

Правила преобразований для контейнеров

Правила для Optional

- Если для целевого типа задан больший уровень `Optional` чем для исходного то это эквивалентно добавлению `Just` поверх `CAST` с меньшим уровнем `Optional`.
- Если же больший уровень `Optional` у исходного типа, то `NULL` на любом уровне больше целевого приводит к `NULL` в результате.
- При равных уровнях `Optional` значение `NULL` остаётся на том же уровне.

```
SELECT
    CAST(1 AS Int32?),                  -- тоже что и Just(1)
    CAST(Just(2/1) AS Float??),         -- [2]
    CAST(Just(3/0) AS Float??) IS NULL; -- false: результат Just(NULL)
```

Правила для List/Dict

- Список формируется путём выполнения `CAST` для каждого элемента исходного списка в тип элемента целевого типа.
- Если тип элемента целевого типа не опциональный, а `CAST` элемента может быть неуспешным, то такие преобразования отбрасываются. В этом случае список в результате может быть меньшей длины или вовсе пустой, если успешных преобразований не было.
- Для словарей преобразование выполняется полностью аналогично спискам, выполняя `CAST` для ключей и значений.

```
SELECT
    CAST([-1, 0, 1] AS List<Uint8?>),             -- [null, 0, 1]
    CAST(["3.14", "bad", "42"] AS List<Float>),   -- [3.14, 42]

    CAST({-1:3.14, 7:1.6} AS Dict<Uint8, Utf8>),  -- {7: "1.6"}
    CAST({-1:3.14, 7:1.6} AS Dict<Uint8?, Utf8>); -- {7: "1.6", null:"3.14"}
```

Правила для Struct/Tuple

- Структура или кортеж формируется путём выполнения `CAST` для каждого элемента исходного типа в элемент с тем же именем или индексом целевого типа.
- Если какое-то поле отсутствует в целевом типе, оно просто отбрасывается.
- Если какое-то поле отсутствует в типе исходного значения, то оно может быть добавлено только если является опциональным, и получает значение `NULL`.
- Если какое-то поле не является опциональным в целевом типе, но его преобразование может быть неуспешным, то `CAST` добавляет опциональность на уровень структуры или кортежа и может вернуть `NULL` для всего результата.

```
SELECT
    CAST((-1, 0, 1) AS Tuple<Uint16?, Uint16?, Utf8>), -- (null, 0, "1")
    CAST((-2, 0) AS Tuple<Uint16, Utf8>),              -- null
    CAST((3, 4) AS Tuple<Uint16, String>),             -- (3, "4") тип Tuple<Uint16, String>?
    CAST(("4",) AS Tuple<Uint16, String?>),            -- (4, null)
    CAST((5, 6, null) AS Tuple<Uint8?>);               -- (5,)  элементы удалены.

SELECT -- Одно поле удалено и одно добавлено: ("three":null, "two": "42")
    CAST(<|one:"8912", two:42|> AS Struct<two:Utf8, three:Date?>);
```

Правила для Variant

- Для варианта с определёнными именем или индексом выполняется преобразование в вариант с тем же именем или индексом.
- Если преобразование для варианта может быть неуспешным и тип этого варианта не опциональный, то `CAST` добавляет опциональность на верхний уровень и может вернуть `NULL`.
- Если какой-то вариант отсутствует в целевом типе, то `CAST` добавляет опциональность на верхний уровень и для такого значения возвращает `NULL`.

Вложенные контейнеры

- Все вышеперечисленные правила работают рекурсивно для вложенных контейнеров.

Работа с текстовым представлением типов данных

Введение

Так как YQL является строго типизированным языком, во многих аспектах работы с ним тип данных имеет большое значение. Для удобства работы с типами существует конвенция описания типов данных YQL в текстовом виде. Она используется в разных местах документации, а также существует библиотека, предоставляющая функции для построения типа данных по текстовому описанию (например, при ручном описании сигнатуры вызываемого значения) или для сериализации типа данных в строку для отладки.

Функции для работы с типами данных описаны в статье, а ниже описан сам формат текстового представления типов данных.

Общие правила

- Примитивные типы данных представляются в текстовом виде просто своим именем.
- Сложный тип данных представляет собой композицию из других типов данных. Если представить эту композицию в виде дерева, то на листьях окажутся примитивные типы данных, а в остальных узлах — контейнеры. Специальные типы данных можно рассматривать как исключение, они могут находиться и там и там.
- Текстовое представление повторяет структуру этого дерева в порядке от корня к листьям: в каждом узле дерева указывается имя текущего типа данных, а переход на уровень глубже обозначается различными видами скобок.
- Допустимо использование пробелов и переносов строк, если это облегчает чтение.
- Если идентификатор состоит не только из английских букв и цифр, то его нужно записывать в одиночных кавычках и использовать C-escaping.

Контейнеры

- Для указания типов элементов контейнера используются угловые скобки.

  Пример: `List<Int32>`.
- Если контейнер предполагает несколько разнородных элементов, то они перечисляются внутри угловых скобок через запятую.

  Пример: `Tuple<Int32, String>`.
- Если контейнер предполагает именованные элементы, то вместо типов данных через запятую используются пары из имени и типа через двоеточие.

  Пример: `Struct<a:Int32, b:String>`.
- Нижележащий тип контейнера `Variant` выбирается в зависимости от наличия имён в аргументах.

  Пример: `Variant<Int32, String>` — вариант над кортежем, `Variant<a:Int32, b:String>` — вариант над структурой.

Типы, допускающие NULL

- В терминах YQL это называется `Optional`, в терминах классического SQL — nullable.
- Формально этот тип является контейнером, т.е. запись `Optional<...>` допустима, но обычно вместо неё используется shortcut в виде суффикса из знака вопроса.

  Пример: `String?`.

Вызываемые значения

- Базовая форма вызываемых значений выглядит следующим образом: `(arg1, arg2, ...) -> result`.

  Пример описания сигнатуры функции, принимающей две строки и возвращающей число: `(String, String) -> Int64`.
- Вызываемые значения могут возвращать вызываемые значения, в этом случае они образуют цепочку необходимой длины.

  Пример: `(String, String) -> (String, String) -> Int64`.
- Опциональные аргументы должны иметь на верхнем уровне тип `Optional` и обрамляются в квадратные скобки.

  Пример: `(String, [String?, Double?]) -> Int64`.
- У аргументов вызываемых значений могут быть указаны флаги.

  На текущий момент возможен только один флаг — `AutoMap`, который означает, что если в этот аргумент передали NULL, то результат нужно сделать тоже NULL, а саму функцию не запускать.

  Пример: `(String{Flags: AutoMap}) -> Int64`.
- Если нужен `Optional<Callable<...>>`, то нужно использовать именно такую форму, т.к. знак вопроса в конце относится к результату вызываемого значения.

Ресурсы

- В отличие от контейнеров, ресурс параметризуется не типом элемента (ресурс является указателем в памяти и YQL ничего не знает о его содержимом), а строковой меткой, которая может использоваться для защиты от передачи ресурсов между несовместимыми функциями.

  Пример: `Resource<Foo>`.

Передача параметров и данных в формате JSON

YQL позволяет задавать параметры используя DECLARE выражение. Сами параметры передаются с помощью restricted JSON формата.
Ниже дано описание формата передачи различных типов данных.
Этот формат также используется для ответов из API.

Числовые типы

Bool

Логическое значение.

- Тип в JSON — `bool`.
- Пример значения JSON — `true`.

Int, Uint, Float, Double, Decimal

- Тип в JSON — `string`.
- Числа передаются в строковом представлении.
- Пример значения JSON — `"123456"`, `"-42"`, `"0.12345679"`, `"-320.789"`.

Строковые типы

String

Бинарные строки.

- при наличии невалидных utf-8 символов, например, `\xFF`, строка кодируется в base64 и оборачивается в массив с одним элементом
- если нет специальных символов, то передается как есть
- Тип в JSON - `string` или `array`
- Пример значения JSON - `"AB"`, `["qw6="]` (для строки `\xAB\xAC`)

Utf8

Строковые типы в utf-8. Такие строки представляются в JSON строками с escaping'ом JSON-символов: `\\`, `\"`, `\n`, `\r`, `\t`, `\f`.

- Тип в JSON — `string`.
- Пример значения JSON — `"Escaped characters: \\ \" \f \b \t \r\nNon-escaped characters: / ' < > & []() "`.

Uuid

Универсальный идентификатор UUID.

- бинарный формат UUID кодируется с помощью base64 и оборачивается в лист
- Тип в JSON - `array`
- Пример значения JSON - `["AIQOVZvi1EGnFkRmVUQAAA=="]` для `550e8400-e29b-41d4-a716-446655440000`

Yson

- Тип в JSON - `object`.
- YSON более богатый язык, чем JSON, поэтому есть необходимость предоставлять дополнительные атрибуты, поэтому задаются поля `$value`, `$type`, `$attributes`.
- Строки, числа и логические значения заменяются на объект с двумя ключами: `$value` и `$type`. Типы могут быть: `boolean`, `int64`, `uint64`, `double`, `string`.
- Каждый байт бинарной строки переводится в юникодный символ с соответствующим номером и кодируется в UTF-8 (такой же механизм использует YT при преобразовании Yson в Json).
- Yson атрибуты добавляются в виде отдельного объекта с ключом `$attributes` и значениями атрибутов.
- если какое имя начинается с `$`, то он экранируется с помощью удваивания `$$`.
- Пример YSON:

```
{ "$a" = 2;
  b = {
    c = <attr1=val1;attr2=5>12.5;
    d = [ "el"; # ]
  }
}
```

- Пример значения JSON:

```
{
  "$$a" : { "$value": "2", "$type": "int64" },
  "b" : {
    "c": {
          "$value" : "12.5",
          "$type" : "double",
          "$attributes" :
            {
              "attr1": { "$value": "b", "$type": "string" },
              "attr2": { "$value": "5", "$type": "int64" }
            }
        },
    "d": [ { "$value": "el", "$type": "string" }, null ]
```

```
  }
}
```

Json

- Тип в JSON - `object`.
- Пример значения JSON - `{ "a" : 12.5, "c" : 25 }`.

Дата и время

Date

Дата, внутреннее представление - Uint16, количество дней с unix epoch.

- Тип в JSON — `string`.
- Пример значения JSON — `"19509"` для даты `2023-06-01`.

Datetime

Дата и время, внутреннее представление - Uint32, количество секунд с unix epoch.

- Тип в JSON — `string`.
- Пример значения JSON — `"1686966302"` для даты `"2023-06-17T01:45:02Z"`.

Timestamp

Дата и время, внутреннее представление - Uint64, количество микросекунд с unix epoch.

- Тип в JSON — `string`.
- Пример значения JSON — `"1685577600000000"` для `2023-06-01T00:00:00.000000Z`.

Interval

Временной интервал, внутреннее представление - Int64, точность до микросекунд.

- Тип в JSON — `string`.
- Пример значения JSON — `"12345678910"`.

TzDate, TzDateTime, TzTimestamp

Временные типы с меткой временной зоны.

- Тип в JSON - `string`.
- Значение представляется как строковое представление времени и временная зона через запятую.
- Пример значения JSON - `"2023-06-29,Europe/Moscow"`, `""2023-06-29T17:14:11,Europe/Moscow""`, `""2023-06-29T17:15:36.645735,Europe/Moscow""` для TzDate, TzDateTime и TzTimestamp соответственно.

Контейнеры

В контейнерах все элементы кодируются согласно их типу.

List

Список. Упорядоченный набор значений заданного типа, значения передаются как строки.

- Тип в JSON — `array`.
- Пример значения JSON для `List<Int32>` — `["1","10","100"]`.

Struct

Структура. Неупорядоченный набор значений с заданными именами и типом.

- Тип в JSON — `object` или `array`.
- Может передоваться как объект или как массив, где порядок элементов должен совпадать с соответствующими полями структуры.
- Пример значения JSON `{"a": "-100", "b": "foo"}`, `{"a": "-100", "b": "foo", "c": null}` или `["-100", "foo", null]` для `Struct<a:Int32, b:String, c:Optional<String>>`.

Tuple

Кортеж. Упорядоченный набор значений заданных типов.

- Тип в JSON — `array`.
- Пример значения JSON для типа Tuple<Int32, String, Float?> — `[-1,"Some string",null]`.

Dict

Словарь. Неупорядоченный набор пар ключ-значение.

- Тип в JSON — `object` или `array`.
- для ключей типа `String` и `Utf8` можно передавать объект, для остальных типов надо передавать массив массивов, состоящих из двух элементов (ключа и значения).
- Пример значения JSON — `[["1","123"],["2","456"]]` для `Dict<Int32, Interval>`, `{ "foo": "123", "bar": "456" }` для `Dict<String, Int32>`.

Enum

Перечисление.

- Тип в JSON - `string`.
- Пример значения JSON - `"b"` для `Enum<a,b>`.

Variant

Кортеж или структура, про которые известно, что заполнен ровно один элемент.

- Тип в JSON - `array`.
- Первый способ передачи - массив из поля структуры, обернутый в массив, и значения. Способ подходит только для варианта над структурами.
- Второй способ передачи - массив из индекса поля структуры/кортежа и само значение.
- Пример значения JSON - `[["foo"], false]`, `[["bar"], "6"]` или `["0", false]`, `["1", "6"]` для `Variant<foo: Int32, bar: Bool>`.

Специальные типы

Optional

Означает, что значение может быть `null`.

- Тип в JSON — `array` или `null`.
- Значения обертываются в массив, `null` значение представляется пустым массивом или `null`.
- Пример значения JSON - `[["1"], ["2"], ["3"], []]` или `[["1"], ["2"], ["3"], null]` для `List<Optional<Int32>>`.

Void

Сингулярный тип данных с единственным возможным значением `null`.

- Тип в JSON - `string`.
- Значение JSON - `"Void"`.

Список статей по синтаксису YQL

- Лексическая структура
- Выражения
- SELECT
- VALUES
- CREATE TABLE
- DROP TABLE
- TRUNCATE TABLE
- INSERT
- SHOW CREATE
- ANALYZE
- KILL SESSION
- ALTER TABLE
- UPDATE
- DELETE
- REPLACE
- UPSERT
- BATCH UPDATE
- BATCH DELETE
- GROUP BY
- JOIN
- WINDOW
- FLATTEN
- ACTION
- DISCARD
- INTO RESULT
- PRAGMA
- DECLARE
- CREATE TOPIC
- ALTER TOPIC
- DROP TOPIC
- CREATE ASYNC REPLICATION
- ALTER ASYNC REPLICATION
- DROP ASYNC REPLICATION
- CREATE BACKUP COLLECTION
- BACKUP
- RESTORE
- DROP BACKUP COLLECTION
- CREATE TRANSFER
- ALTER TRANSFER
- DROP TRANSFER
- COMMIT
- CREATE VIEW
- ALTER VIEW
- DROP VIEW
- CREATE EXTERNAL DATA SOURCE
- CREATE EXTERNAL TABLE
- DROP EXTERNAL DATA SOURCE
- DROP EXTERNAL TABLE
- CREATE OBJECT (TYPE SECRET)
- CREATE OBJECT (TYPE SECRET_ACCESS)
- DROP OBJECT (TYPE SECRET)
- DROP OBJECT (TYPE SECRET_ACCESS)
- UPSERT OBJECT (TYPE SECRET)
- CREATE RESOURCE POOL
- ALTER RESOURCE POOL
- DROP RESOURCE POOL
- CREATE RESOURCE POOL CLASSIFIER
- ALTER RESOURCE POOL CLASSIFIER
- DROP RESOURCE POOL CLASSIFIER
- CREATE USER
- ALTER USER
- DROP USER
- CREATE GROUP
- ALTER GROUP
- DROP GROUP
- GRANT
- REVOKE
- Неподдерживаемые конструкции

Лексическая структура

Запрос на языке YQL представляет собой валидный UTF-8 текст, который состоит из *команд* (statement) разделенных символом точка с запятой (`;`).
Последняя точка с запятой при этом может отсутствовать.
Каждая команда, в свою очередь, состоит из последовательности *токенов* допустимых для данной команды.
Токеном может быть *ключевое слово*, *идентификатор*, *литерал* и другие.
Токены разделяются пробельными символами (пробел, табуляция, перевод строки) либо *комментариями*. Комментарий не является частью команды и синтаксически эквивалентен пробельному символу.

Режимы совместимости синтаксиса

Поддерживаются два режима совместимости синтаксиса:

- Расширенный C++ (по умолчанию)
- ANSI SQL

Режим ANSI SQL включается с помощью специального комментария `--!ansi_lexer`, который должен стоять в начале запроса.

Особенности интерпретации лексических элементов в разных режимах совместимости описаны ниже.

Комментарии

Поддерживаются следующие виды комментариев:

- Однострочные: начинается с последовательности символов `--` (два минуса *подряд*) и продолжается до конца строки
- Многострочные: начинается с последовательности символов `/*` и заканчивается на последовательности символов `*/`

```
SELECT 1; -- A single-line comment
/*
   Some multi-line comment
*/
```

В режиме совместимости синтаксиса C++ (по умолчанию) многострочный комментарий заканчивается на *ближайшей* последовательности символов `*/` .
В режиме совместимости синтаксиса ANSI SQL учитывается вложенность многострочных комментариев:

```
--!ansi_lexer
SELECT * FROM T; /* комментарий /* вложенный комментарий, без ansi_lexer будет ошибка  */ */
```

Ключевые слова и идентификаторы

Ключевые слова – это токены, имеющее фиксированное значение в языке YQL. Примеры ключевых слов – `SELECT`, `INSERT`, `FROM`, `ACTION` и т.д. Ключевые слова регистронезависимы, то есть `SELECT` и `SeLEcT` эквивалентны.
Список ключевых слов не фиксирован – по мере развития языка он будет расширяться. Ключевое слово не может содержать цифры и начинаться или заканчиваться символом подчеркивания.

Идентификаторы – это токены, которые идентифицируют имена таблиц, колонок и других объектов в YQL. Идентификаторы в YQL всегда регистрозависимы.
Идентификатор может быть записан в теле программы без специального оформления, если он:

- Не является ключевым словом
- Начинается с латинской буквы или подчеркивания
- Последующими символами могут быть латинская буква, подчеркивание или цифра

```
SELECT my_column FROM my_table; -- my_column and my_table are identifiers
```

Для записи в теле запроса произвольного идентификатора он заключается в обратные кавычки (бэктики):

```
SELECT `column with space` from T;
SELECT * FROM `my_dir/my_table`
```

Идентификатор в обратных кавычках никогда не интерпретируется как ключевое слово:

```
SELECT `select` FROM T; -- select - имя колонки в таблице T
```

При использовании обратных кавычек применим стандартный C-эскейпинг:

```
SELECT 1 as `column with\n newline, \x0a newline and \` backtick `;
```

В режиме совместимости синтаксиса ANSI SQL произвольные идентификаторы также могут быть выделены заключением их в двойные кавычки. Для включения двойной кавычки в идентификатор в кавычках она должна быть удвоена:

```
--!ansi_lexer
SELECT 1 as "column with "" double quoute"; -- имя колонки будет: column with " double quoute
```

Параметризация запроса

Параметры запроса — это именованные выражения, значение для которых передаётся отдельно от текста запроса (см. DECLARE). Такие параметры можно подставлять только в те части запроса, которые не требуются для его компиляции. Например, параметр можно использовать в фильтре:

```
DECLARE $my_param AS String;
SELECT Data FROM my_table WHERE Data LIKE $my_param
```

Или в данных:

```
DECLARE $my_param AS String;
SELECT Data || $my_param FROM my_table
```

При этом параметры нельзя использовать в частях, необходимых для компиляции запроса, например:

- В именах таблиц;
- В любых параметрах блока WITH;
- В любых выражениях внутри EvaluateExpr.

Для таких частей можно использовать только строковые константы внутри запроса, например:

```
$table = "my_table";
SELECT * FROM $table
```

SQL хинты

SQL хинты – это специальные настройки, которые позволяют пользователю влиять на план выполнения запроса (например, включать/выключать определенные оптимизации, форсировать стратегию JOIN-а и т.п.).
В отличие от PRAGMA, SQL хинты обладают локальным действием – они привязаны к определенной точке YQL запроса (обычно следуют после ключевого слова)
и влияют только на соответствующий statement или даже его часть.
SQL хинты представляют собой набор настроек "имя-список значений" и задаются внутри комментариев специального вида – первым символом комментария с SQL хинтами должен быть `+`:

```
--+ Name1(Value1 Value2 Value3) Name2(Value4) ...
```

Имя SQL хинта должно состоять из алфавтно-цифровых ASCII символов и начинаться с буквы. Регистр букв в имени хинта игнорируется.
После имени хинта в скобках задается произвольное количество значений, разделенных пробелами. В качестве значения может выступать произвольный набор символов.
Если в наборе символов значения имеется пробел или скобка, то необходимо использовать одинарные кавычки:

```
--+ foo('value with space and paren)')
```

```
--+ foo('value1' value2)
-- эквивалетно
--+ foo(value1 value2)
```

Одинарную кавычку внутри значения необходимо эскейпить путем дублирования:

```
--+ foo('value with single quote '' inside')
```

Неизвестные имена SQL хинтов (либо синтаксически некорректные хинты) никогда не вызывают ошибок – они просто игнорируется:

```
--+ foo(value1) bar(value2  baz(value3)
-- из-за пропущенной закрывающей скобки в bar эквивалетно
--+ foo(value1)
```

Такое поведение связано с нежеланием ломать написанные ранее валидные YQL запросы с комментариями, которые похожи на хинты.
При этом синтаксически корректные SQL хинты в неожиданном для YQL месте вызывают предупреждение:

```
-- в данный момент хинты после SELECT не поддерживаются
SELECT /*+ foo(123) */ 1; -- предупреждение 'Hint foo will not be used'
```

Хочется заметить, что SQL хинты – это именно подсказки оптимизатору, поэтому:

- хинты никогда не влияют на результат запроса
- по мере развития оптимизаторов в YQL вполне возможна ситуация, в которой хинт становится неактуальным и начнет игнорироваться (например, полностью поменялся алгоритм, который настраивался данным хинтом, либо оптимизатор настолько улучшился, что гарантированно выбирает оптимальное решение, поэтому какие-то ручные настройки будут скорее вредить)

Строковые литералы

Строковый литерал (константа) записывается как последовательность символов, заключенных в одинарные кавычки. Внутри строкового литерала можно использовать правила эскейпинга в стиле C:

```
SELECT 'string with\n newline, \x0a newline and \' backtick ';
```

В режиме совместимости синтаксиса C++ (по-умолчанию) разрешается использовать вместо одинарных кавычек двойные:

```
SELECT "string with\n newline, \x0a newline and \" backtick ";
```

В режиме совместимости синтаксиса ASNI SQL двойные кавычки используются для идентификаторов, а единственный вид эскепинга который действует для строковых литералов – это дублирование символа одиночной кавычки:

```
--!ansi_lexer
SELECT 'string with '' quote'; -- результат: string with ' quote
```

На основании строковых литералов могут быть получены литералы простых типов.

Многострочные строковые литералы

Многострочный строковой литерал записывается в виде произвольного набора символов между двойными собачками `@@`:

```
$text = @@some
multiline
text@@;
SELECT LENGTH($text);
```

Если необходимо вставить в текст двойную собачку, ее необходимо удвоить:

```
$text = @@some
multiline with double at: @@@@
text@@;
SELECT $text;
```

Типизированные строковые литералы

- Для строкового литерала, включая многострочный, по умолчанию используется тип `String` (см. также PRAGMA UnicodeLiterals).
- С помощью следующих суффиксов можно явно управлять типом литерала:
  - `s` — `String`;
  - `u` — `Utf8`;
  - `y` — `Yson`;
  - `j` — `Json`.

Пример

```
SELECT "foo"u, '[1;2]'y, @@{"a":null}@@j;
```

Числовые литералы

- Целочисленные литералы по умолчанию имеют тип `Int32`, если попадают в его диапазон, и в противном случае автоматически расширяются до `Int64`.
- С помощью следующих суффиксов можно явно управлять типом литерала:
  - `l` — `Int64`;
  - `s` — `Int16`;
  - `t` — `Int8`.
- Добавление суффикса `u` превращает тип в соответствующий беззнаковый:
  - `ul` — `Uint64`;
  - `u` — `Uint32`;
  - `us` — `Uint16`;
  - `ut` — `Uint8`.
- Также для целочисленных литералов доступна запись в шестнадцатеричной, восьмеричной и двоичной форме с помощью префиксов `0x`, `0o` и `0b`, соответственно. Их можно произвольным образом комбинировать с описанными выше суффиксами.
- Литералы с плавающей точкой по умолчанию имеют тип `Double`, но с помощью суффикса `f` его можно сузить до `Float`.

```
SELECT
  123l AS `Int64`,
  0b01u AS `Uint32`,
  0xfful AS `Uint64`,
  0o7ut AS `Uint8`,
  456s AS `Int16`,
  1.2345f AS `Float`;
```

Литералы PostgreSQL

Строковые и числовые литералы Pg типов можно создавать с помощью специальных суффиксов (аналогично простым строковым и числовым литералам).

Целочисленные литералы

Суффикс	Тип	Комментарий
`p`	`PgInt4`	32-битное знаковое целое (в PostgreSQL нет беззнаковых типов)
`ps`	`PgInt2`	16-битное знаковое целое
`pi`	`PgInt4`	

`pb`	`PgInt8`	64-битное знаковое цело
`pn`	`PgNumeric`	знаковое целое произвольной точности (до 131072 цифр)

Литералы с плавающей точкой

Суффикс	Тип	Комментарий
`p`	`PgFloat8`	число с плавающей точкой (64 бит double)
`pf4`	`PgFloat4`	число с плавающей точкой (32 бит float)
`pf8`	`PgFloat8`	
`pn`	`PgNumeric`	число с плавающей точкой произвольной точности (до 131072 цифр перед запятой, до 16383 цифр после запятой)

Строковые литералы

Суффикс	Тип	Комментарий
`p`	`PgText`	текстовая строка
`pt`	`PgText`	
`pv`	`PgVarchar`	текстовая строка
`pb`	`PgBytea`	бинарная строка

Внимание

Значения строковых/числовых литералов (т.е. то что идет перед суффиксом) должны быть валидной строкой/числом с точки зрения YQL.
В частности, должны соблюдаться правила эскейпинга YQL, а не PostgreSQL.

Пример:

```
SELECT
    1234p,             -- pgint4
    0x123pb,         -- pgint8
    "тест"pt,         -- pgtext
    123e-1000pn; -- pgnumeric
;
```

Литерал массива

Для построения литерала массива используется функция `PgArray`:

```
SELECT
    PgArray(1p, NULL ,2p) -- {1,NULL,2}, тип _int4
;
```

Конструктор литералов произвольного типа

Литералы всех типов (в том числе и Pg типов) могут создаваться с помощью конструктора литералов со следующей сигнатурой:
`Имя_типа(<строковая константа>)`.

Напрмер:

```
DECLARE $foo AS String;
SELECT
    PgInt4("1234"), -- то же что и 1234p
    PgInt4(1234),   -- в качестве аргумента можно использовать литеральные константы
    PgInt4($foo),   -- и declare параметры
    PgBool(true),
    PgInt8(1234),
    PgDate("1932-01-07"),
;
```

Также поддерживается встроенная функция `PgConst` со следующей сигнатурой: `PgConst(<строковое значение>, <тип>)`.
Такой способ более удобен для кодогенерации.

Например:

```
SELECT
    PgConst("1234", PgInt4), -- то же что и 1234p
    PgConst("true", PgBool)
;
```

Для некоторых типов в функции `PgConst` можно указать дополнительные модификаторы. Возможные модификаторы для типа `pginterval` перечислены в документации PostgreSQL.

```
SELECT
    PgConst(90, pginterval, "day"), -- 90 days
    PgConst(13.45, pgnumeric, 10, 1); -- 13.5
;
```

Выражения

Конкатенация строк

Выполняется через бинарный оператор `||`.

Как и у других бинарных операторов, если в данных с одной из сторон оказался `NULL`, то и результат будет `NULL`.

Не следует путать этот оператор с логическим «или», в SQL оно обозначается ключевым словом `OR`. Также не стоит пытаться делать конкатенацию через `+`.

Примеры

```
SELECT "fo" || "o";
```

Проверка строки на соответствие шаблону

`REGEXP` и `RLIKE` являются алиасами и представляют собой короткий способ записи для вызова Re2::Grep. `MATCH` — аналогично для Re2::Match.

`LIKE` работает по следующим принципам:

- В шаблонах используется два спецсимвола:
  - `%` — ноль и более любых символов;
  - `_` — ровно один любой символ.

  Все остальные символы выступают как литералы, то есть обозначают сами себя.
- В отличие от `REGEXP`, с шаблоном `LIKE` строка должна совпасть полностью. Например, для поиска подстроки в середине нужно добавить `%` в начале и в конце шаблона.
- `ILIKE` является не чувствительной к регистру версией `LIKE`.
- Если `LIKE` применяется к ключевой колонке сортированной таблицы и шаблон начинается не со спецсимвола, то фильтрация по префиксу опускается прямо до кластера, что в некоторых случаях позволяет не сканировать всю таблицу целиком. Для `ILIKE` данная оптимизация отключена.
- Чтобы заэкранировать спецсимволы, необходимо после шаблона указать символ для экранирования с помощью ключевого слова `ESCAPE '?'`. Вместо `?` можно использовать любой символ, кроме `%`, `_` и `\`. Например, если в качестве экранирующего символа используется знак вопроса, то выражения `?%`, `?_` и `??` в шаблоне совпадут со своим вторым символом: процент, подчеркивание и знак вопроса, соответственно. По умолчанию экранирующий символ не определен.

Наиболее популярный способ использования ключевых слов `LIKE` и `REGEXP` — фильтрация таблицы в выражениях с `WHERE`. Однако ограничения на использование шаблонов именно в этом контексте нет, и их можно использовать в большинстве контекстов при работе со строками, наравне, например, с конкатенацией с помощью `||`.

LIKE / ILIKE с полнотекстовым индексом

Для полнотекстовых индексов с N-граммами (параметры `use_filter_ngram` / `use_filter_edge_ngram`) поддерживается `LIKE` / `ILIKE` по индексируемой текстовой колонке.

```
SELECT id, title
FROM articles VIEW ft_idx
WHERE body ILIKE "%обуч%ние%"
LIMIT 20;
```

Примеры

```
SELECT * FROM my_table
WHERE string_column REGEXP '\\d+';
-- второй слеш нужен, так как все
-- стандартные строковые литералы в SQL
-- могут принимать C-escaped строки
```

```
SELECT
    string_column LIKE '___!_!_!_!!!!!!' ESCAPE '!'
    -- ищет строку из ровно 9 символов:
    --   3 произвольных,
    --   затем 3 подчеркивания
    --   и 3 восклицательных знака
FROM my_table;
```

```
SELECT * FROM my_table
WHERE key LIKE 'foo%bar';
-- при наличии сортировки таблицы по key просканирует только ключи,
-- начинающиеся на foo, и затем среди них
-- оставит только заканчивающиеся на bar
```

Операторы

Арифметические операторы

Операторы `+`, `-`, `*`, `/`, `%` определены для примитивных типов данных, являющихся разновидностями чисел.

Для типа данных Decimal используется банковское округление (до ближайшего четного).

Примеры

```
SELECT 2 + 2;
```

```
SELECT 0.0 / 0.0;
```

Операторы сравнения

Операторы `=`, `==`, `!=`, `<>`, `>`, `<`, `>=`, `<=` определены для:

- Примитивных типов данных за исключением Yson и Json.
- Кортежей и структур с одинаковым набором полей. Для структур не определен порядок, но можно проверять на (не-)равенство, а кортежи сравниваются поэлементно слева направо.

Примеры

```
SELECT 2 > 1;
```

Логические операторы

С помощью операторов `AND`, `OR`, `XOR` осуществляются логические операции над булевыми значениями (`Bool`).

Примеры

```
SELECT 3 > 0 AND false;
```

Битовые операторы

Битовые операции над числами:

- `&`, `|`, `^` — AND, OR и XOR соответственно. Не следует путать битовые операции с аналогичными ключевыми словами. Ключевые слова `AND`, `OR` и `XOR` используются *только для булевых значений*, но не для чисел;
- `~` — отрицание;
- `<<`, `>>` — сдвиги влево-вправо;
- `|<<`, `>>|` — кольцевые сдвиги влево-вправо.

Примеры

```
SELECT
    key << 10 AS key,
    ~value AS value
FROM my_table;
```

Приоритет и ассоциативность операторов

Приоритет оператора определяет порядок вычисления выражения содержащего разные операторы.
Например, выражение `1 + 2 * 3` вычисляется как `1 + (2 * 3)`,
поскольку приоритет оператора умножения больше приоритета оператора сложения.

Ассоциативность определяет порядок вычисления в выражениях содержащих операторы одного типа.
Например, выражение `1 + 2 + 3` вычисляется как `(1 + 2) + 3`, поскольку оператор сложения является лево-ассоциативным.
С другой стороны, выражение `a ?? b ?? c` вычисляется как `a ?? (b ?? c)` из-за право-ассоциативности оператора `??`

В таблице ниже указаны приоритет и ассоциативность операторов языка YQL.
Операторы в таблице перечислены в порядке убывания приоритета.

Приоритет	Оператор	Описание	Ассоциативность
1	`a[], a.foo, a()`	Обращение к элементу контейнера, вызов функции	Левая
2	`+a, -a, ~a, NOT a`	Унарные операторы: плюс, минус, битовое и логическое отрицание	Правая
3	`a||b`	Конкатенация строк	Левая
4	`a*b, a/b, a%b`	Умножение,деление, остаток от деления	Левая
5	`a+b, a-b`	Сложение / вычитание	Левая
6	`a ?? b`	Операторная форма записи NVL/COALESCE	Правая
7	`a<<b, a>>b, a|<<b, a>>|b, a|b, a^b, a&b`	Сдвиговые и логические битовые операторы	Левая
8	`a<b, a<=b, a>=b, a>b`	Сравнение	Левая
9	`a IN b`	Вхождение элемента в множество	Левая
9	`a==b, a=b, a!=b, a<>b, a is (not) distinct from b`	Сравнение на (не)равенство	Левая
10	`a XOR b`	Логическое XOR	Левая

11	`a AND b`	Логическое AND	Левая
12	`a OR b`	Логическое OR	Левая

IS [NOT] NULL

Проверка на пустое значение (`NULL`). Так как `NULL` является особым значением, которое ничему не равно, то обычные операторы сравнения для этой задачи не подходят.

Примеры

```
SELECT key FROM my_table
WHERE value IS NOT NULL;
```

IS [NOT] DISTINCT FROM

Сравнение двух значений. В отличие от обычных операторов сравнения, нуллы считаются равными друг другу.

Сравнение осуществляется по следующим правилам:

1. операторы `IS DISTINCT FROM` / `IS NOT DISTINCT FROM` определены для тех и только для тех аргументов, для которых определены операторы `!=` и `=`;
2. результат `IS NOT DISTINCT FROM` равен логическому отрицанию результата `IS DISTINCT FROM` для данных аргументов;
3. если результат оператора `==` не равен нуллу для некоторых аргументов, то он совпадает с результатом оператора `IS NOT DISTINCT FROM` для тех же аргументов;
4. если оба аргумента являются незаполненными `Optional`ми или `NULL`ами, то значение `IS NOT DISTINCT FROM` равно `True`
5. результат `IS NOT DISTINCT FROM` от незаполненного `Optional` или `NULL` и заполненного `Optional` или не-`Optional` значения равен `False`.

Для значений композитных типов эти правила применяются рекурсивно.

BETWEEN

Проверка на вхождение значения в диапазон. Синтаксис: `expr [NOT] BETWEEN [ASYMMETRIC | SYMMETRIC] expr AND expr`.

- `BETWEEN` и `BETWEEN ASYMMETRIC` эквивалентны, `x BETWEEN a AND b` эквивалентно `a <= x AND x <= b`.
- `BETWEEN SYMMETRIC` автоматически переставляет аргументы местами так чтобы диапазон получился непустым, `x BETWEEN SYMMETRIC a AND b` эквивалентно `(x BETWEEN a AND b) OR (x BETWEEN b AND a)`.
- `NOT` инвертирует результат проверки.

Примеры

```
SELECT * FROM my_table
WHERE key BETWEEN 10 AND 20;
```

```
SELECT * FROM my_table
WHERE key NOT BETWEEN SYMMETRIC 20 AND 10;
```

IN

Проверка вхождения одного значения в набор значений. Логически эквивалентно цепочке сравнений на равенство через `OR`, но реализовано более эффективно.

> **Важно**
>
> В отличие от аналогичного ключевого слова в Python, в YQL `IN` **НЕ** является поиском подстроки в строке. Для поиска подстроки можно использовать функцию String::Contains или описанные выше LIKE / REGEXP.

Сразу после `IN` можно указать хинт `COMPACT`.
Если `COMPACT` не указан, то `IN` с подзапросом по возможности выполняется как соответствующий `JOIN` (`LEFT SEMI` для `IN` и `LEFT ONLY` для `NOT IN`).
Наличие `COMPACT` форсирует in-memory стратегию выполнения: из содержимого правой части `IN` в памяти сразу строится хеш-таблица, по которой затем фильтруется левая часть.

Хинтом `COMPACT` следует пользоваться с осторожностью. Поскольку хеш-таблица строится в памяти, то запрос может завершиться с ошибкой, если правая часть `IN` содержит много больших и/или различных элементов.

Примеры

```
SELECT column IN (1, 2, 3)
FROM my_table;
```

```
SELECT * FROM my_table
WHERE string_column IN ("a", "b", "c");
```

```
$foo = AsList(1, 2, 3);
SELECT 1 IN $foo;
```

```
$values = (SELECT column + 1 FROM table);
SELECT * FROM my_table WHERE
    -- фильтрация по in-memory хеш-таблице на основе table
    column1 IN /*+ COMPACT() */ $values AND
    -- с последующим LEFT ONLY JOIN с other_table
    column2 NOT IN (SELECT other_column FROM other_table);
```

AS

Может использоваться в следующих сценариях:

- Присвоение короткого имени (алиаса) столбцам или таблицам в рамках запроса.
- Указание именованных аргументов при вызове функций.
- При явном приведении типов данных для указания целевого типа, см. CAST.

Примеры

```
SELECT key AS k FROM my_table;
```

```
SELECT t.key FROM my_table AS t;
```

```
SELECT
    MyFunction(key, 123 AS my_optional_arg)
FROM my_table;
```

CAST

Пробует привести значение к указанному типу. Попытка может оказаться неуспешной и вернуть `NULL`. Для чисел может потерять точность или старшие биты.

Для параметрического типа данных Decimal дополнительно указывается два аргумента:

- общее число десятичных знаков (до 35, включительно);
- число десятичных знаков после запятой (из общего числа, то есть строго не больше предыдущего аргумента).

Примеры

```
SELECT
    CAST("12345" AS Double),               -- 12345.0
    CAST(1.2345 AS Uint8),                 -- 1
    CAST(12345 AS String),                 -- "12345"
    CAST("1.2345" AS Decimal(5, 2)),       -- 1.23
    CAST("xyz" AS Uint64) IS NULL,         -- true, так как не удалось
    CAST(-1 AS Uint16) IS NULL,            -- true, отрицательное в беззнаковое
    CAST([-1, 0, 1] AS List<Uint8?>),             -- [null, 0, 1]
        --Тип элемента опциональный: неудачный элемент в null.
    CAST(["3.14", "bad", "42"] AS List<Float>),   -- [3.14, 42]
        --Тип элемента не опциональный: неудачный элемент удалён.
    CAST(255 AS Uint8),                    -- 255
    CAST(256 AS Uint8) IS NULL;            -- true, выходит за диапазон
```

BITCAST

Выполняет побитное преобразование целочисленного значения к указанному целочисленному типу. Преобразование всегда успешно, но может потерять точность или старшие биты.

Примеры

```
SELECT
    BITCAST(100000ul AS Uint32),     -- 100000
    BITCAST(100000ul AS Int16),      -- -31072
    BITCAST(100000ul AS Uint16),     -- 34464
    BITCAST(-1 AS Int16),            -- -1
    BITCAST(-1 AS Uint16);           -- 65535
```

CASE

Условные выражения и ветвление. Аналог `if`, `switch` и тернарных операторов в императивных языках программирования.
Если результатом выражения `WHEN` оказывается `true`, значением выражения `CASE` становится результат,
следующий за условием, а остальная часть выражения `CASE` не вычисляется. Если же условие не выполняется,
за ним таким же образом проверяются все последующие предложения `WHEN`. Если не выполняется
ни одно из условий `WHEN`, значением `CASE` становится результат, записанный в предложении `ELSE`.
Ветка `ELSE` является обязательной в выражении `CASE`. Выражения в `WHEN` проверяются последовательно, сверху вниз.

Так как синтаксис достаточно громоздкий, зачастую удобнее пользоваться встроенной функцией IF.

Примеры

```
SELECT
  CASE
    WHEN value > 0
    THEN "positive"
    ELSE "negative"
```

```
  END
FROM my_table;
```

```
SELECT
  CASE value
    WHEN 0 THEN "zero"
    WHEN 1 THEN "one"
    ELSE "not zero or one"
  END
FROM my_table;
```

Именованные выражения

Сложные запросы могут выглядеть громоздко и содержать много уровней вложенности и/или повторяющихся частей. В YQL имеется возможность использовать именованные выражения – способ назначить имя произвольному выражению или подзапросу. На это именованное выражение можно ссылаться в других выражениях или подзапросах. При этом фактически происходит подстановка исходного выражения/подзапроса по месту использования.

Именованное выражение определяется следующим образом:

```
<named-expr> = <expression> | <subquery>;
```

Здесь `<named-expr>` состоит из символа $ и произвольного непустого идентификатора (например `$foo`).

Если выражение в правой части представляет собой кортеж, то его можно автоматически распаковать, указав в левой части несколько именованных выражений через запятую:

```
<named-expr1>, <named-expr2>, <named-expr3> ... = <expression-returning-tuple>;
```

В этом случае число выражений должно совпадать с размером кортежа.

У каждого именованного выражения есть область видимости. Она начинается сразу после определения именованного выражения и заканчивается в конце ближайшего охватывающего scope имен (например в конце запроса либо в конце тела лямбда-функции, ACTION).
Повторное определение именованного выражения с тем же именем приводит к сокрытию предыдущего выражения из текущей области видимости.

Если именованное выражение ни разу не использовалось, то генерируется предупреждение. Для того, чтобы избавиться от такого предупреждения, достаточно использовать символ подчеркивания в качестве первого символа идентификатора (например `$_foo`).
Именованное выражение `$_` называется анонимным именованным выражением и обрабатывается специальным образом: оно работает, как если бы `$_` автоматически заменялось на `$_<some_uniq_name>`.
Анонимными именованными выражениями удобно пользоваться в тех случаях, когда мы не интересуемся его значением. Например, для извлечения второго элемента из кортежа из трех элементов можно написать:

```
$_, $second, $_ = AsTuple(1, 2, 3);
select $second;
```

Попытка сослаться на анонимное именованное выражение приводит к ошибке:

```
$_ = 1;
select $_; --- ошибка: Unable to reference anonymous name $_
```

Анонимные имена аргументов поддерживаются также для лямбда-функций, ACTION.

Примечание

Если в результате подстановки именованных выражений в графе выполнения запроса получились полностью одинаковые подграфы, они объединяются, чтобы такой подграф выполнялся только один раз.

Примеры

```
$multiplier = 712;
SELECT
  a * $multiplier, -- $multiplier is 712
  b * $multiplier,
  (a + b) * $multiplier
FROM abc_table;
```

```
$intermediate = (
  SELECT
    value * value AS square,
    value
  FROM my_table
);
SELECT a.square * b.value
FROM $intermediate AS a
INNER JOIN $intermediate AS b
ON a.value == b.square;
```

```
$a, $_, $c = AsTuple(1, 5u, "test"); -- распаковка кортежа
```

```
SELECT $a, $c;
```

```
$x, $y = AsTuple($y, $x); -- swap значений выражений
```

Табличные выражения

Табличное выражения – это выражение, которое возвращает таблицу. Табличными выражениями в YQL являются:

- подзапросы: `(SELECT key, subkey FROM T)`
- именованные подзапросы: `$foo = SELECT * FROM T;` (использование именованного подзапроса `$foo` является табличным выражением)

Семантика табличного выражения зависит от контекста в котором оно используется. В YQL табличные выражения могут применяться в следующих контекстах:

- табличный контекст - после FROM.

  Здесь табличные выражения работают как ожидается – например `$input = SELECT a, b, c FROM T; SELECT * FROM $input` вернет таблицу с тремя колонками.

  Табличный контекст также возникает после UNION ALL, JOIN;
- векторный контекст - после IN. В этом контексте табличное выражение обязано содержать ровно одну колонку (имя этой колонки никак не влияет на результат выражения).

  Табличное выражение в векторном контексте типизируется как список (тип элемента списка при этом совпадает с типом колонки). Пример: `SELECT * FROM T WHERE key IN (SELECT k FROM T1)`;
- скалярный контекст возникает *во всех остальных случаях*. Как и в векторном контексте, табличное выражение должно содержать ровно одну колонку, но значением табличного выражения будет скаляр – произвольно выбранное значение этой колонки (если получилось ноль строк, то результатом будет `NULL`). Пример: `$count = SELECT COUNT(*) FROM T; SELECT * FROM T ORDER BY key LIMIT $count / 2`.

Порядок строк в табличном контексте, порядок элементов в векторном контексте и правило выбора значения в скалярном контексте (в случае если значений несколько) не определены. На этот порядок также нельзя повлиять с помощью `ORDER BY`: `ORDER BY` без `LIMIT` в табличных выражениях будет игнорироваться с выдачей предупреждения, а использование `ORDER BY` с `LIMIT` определяет множество элементов, но не порядок внутри этого множества.

Lambda функции

Позволяют комбинировать несколько выражений в одно вызываемое значение.

В круглых скобках перечисляются аргументы, далее после стрелки указывается тело lambda. Тело lambda состоит либо из выражения в круглых скобках, либо из фигурных скобок вокруг необязательной цепочки выражений с присвоением именованных выражений и результата вызова после ключевого слова `RETURN` в последнем выражении.

Область видимости для тела lambda — сначала локальные именованные выражения, затем аргументы, затем именованные выражения, определенные выше lambda на верхнем уровне запроса.

В теле lambda можно использовать только чистые выражения — в том числе другие lambda, возможно, переданные через аргументы. Но нельзя использовать SELECT, INSERT INTO и прочие выражения верхнего уровня.

Один или более последних параметров lambda могут быть помечены вопросиком как необязательные — если они не были указаны при вызове lambda, то им будет присвоено значение `NULL`.

Примеры

```
$f = ($y) -> {
    $prefix = "x";
    RETURN $prefix || $y;
};

$g = ($y) -> ("x" || $y);

$h = ($x, $y?) -> ($x + ($y ?? 0));

SELECT $f("y"), $g("z"), $h(1), $h(2, 3); -- "xy", "xz", 1, 5
```

```
-- если результат лямбды вычисляется единственным выражением, то можно использовать более компактный вариант
синтаксиса:
$f = ($x, $_) -> ($x || "suffix"); -- второй аргумент не используется
SELECT $f("prefix_", "whatever");
```

Обращение к контейнерам

Для адресации к значениям внутри контейнеров:

- `Struct<>`, `Tuple<>` и `Variant<>` — используется **точка**. Набор ключей (для кортежа и соответствующего варианта — индексов) известен в момент компиляции запроса. Валидность ключа **проверяется** до начала выполнения запроса.
- `List<>` и `Dict<>` — используются **квадратные скобки**. Набор ключей (для списка — индексов) известен только во время выполнения запроса. Валидность ключа **не проверяется** до начала выполнения запроса. Если значение не найдено — будет возвращено пустое значение (NULL)

Описание и список доступных контейнеров.

При использовании этого синтаксиса для обращения к контейнерам в столбцах таблиц обязательно нужно указывать полное имя столбца, включая имя или алиас таблицы через точку (см. первый пример ниже).

Примеры

```
SELECT
  t.struct.member,
  t.tuple.7,
  t.dict["key"],
  t.list[7]
FROM my_table AS t;
```

```
SELECT
  Sample::ReturnsStruct().member;
```

Неподдерживаемые конструкции из классического SQL

Коррелированные EXISTS и NOT EXISTS

`EXISTS` и `NOT EXISTS` можно использовать с некоррелированными подзапросами. Коррелированные подзапросы не поддерживаются. Для отбора строк по наличию или отсутствию совпадающих строк используйте `LEFT SEMI JOIN` или `LEFT ONLY JOIN`. Подробнее см. в разделе Коррелированные подзапросы, EXISTS и NOT EXISTS.

INTERSECT и EXCEPT

YQL не поддерживает `INTERSECT` и `EXCEPT`.

NATURAL JOIN

Доступный альтернативный вариант — явно перечислить совпадающие с обеих сторон колонки.

NOW() / CURRENT_TIME()

Доступный альтернативный вариант — воспользоваться функциями CurrentUtcDate, CurrentUtcDatetime и CurrentUtcTimestamp.

Коррелированные подзапросы, EXISTS и NOT EXISTS

Коррелированный подзапрос обращается к колонке или псевдониму таблицы из внешнего запроса. Каждый подзапрос в YQL имеет собственную область видимости и не может обращаться к колонкам или псевдонимам таблиц из внешнего запроса. Поэтому YQL не поддерживает коррелированные подзапросы. Поддержку коррелированных подзапросов планируется добавить в следующих релизах.

Матрица поддержки

SQL-шаблон	Поддержка в YQL	Что использовать
Некоррелированный подзапрос во `FROM` или `IN`	Поддерживается	Используйте подзапрос непосредственно
Некоррелированный `EXISTS (SELECT ...)`	Поддерживается; возвращает `true`, если подзапрос содержит хотя бы одну строку	Используйте непосредственно, если условие не зависит от строки внешнего запроса
Некоррелированный `NOT EXISTS (SELECT ...)`	Поддерживается; возвращает `true`, если подзапрос пуст	Используйте непосредственно, если условие не зависит от строки внешнего запроса
`EXISTS` с обращением к внешней колонке	Не поддерживается	`LEFT SEMI JOIN` или `INNER JOIN` с уникальными ключами справа
`NOT EXISTS` с обращением к внешней колонке	Не поддерживается	`LEFT ONLY JOIN`
Коррелированный скалярный подзапрос или подзапрос с агрегацией	Не поддерживается	Предварительно вычислите результат и используйте `JOIN`

EXISTS

Некоррелированный `EXISTS` проверяет, содержит ли его подзапрос хотя бы одну строку. Его результат не зависит от строки внешнего запроса. Например, следующий запрос возвращает всех клиентов, если таблица `orders` не пуста, и не возвращает ни одного клиента, если она пуста:

```
SELECT client_id, name
FROM clients
WHERE EXISTS (
    SELECT 1
    FROM orders
);
```

Следующий распространённый SQL-запрос не поддерживается, поскольку внутренний запрос обращается к внешнему псевдониму `c`:

```
-- Не поддерживается в YQL.
SELECT c.client_id, c.name
FROM clients AS c
WHERE EXISTS (
    SELECT 1
    FROM orders AS o
    WHERE o.client_id = c.client_id
);
```

Чтобы вернуть клиента, только если существует хотя бы один подходящий заказ, используйте `LEFT SEMI JOIN`:

```
SELECT c.client_id, c.name
FROM clients AS c
LEFT SEMI JOIN orders AS o
ON o.client_id = c.client_id;
```

`LEFT SEMI JOIN` возвращает только колонки левой стороны. Несколько подходящих строк справа не дублируют строку слева, поэтому такое преобразование сохраняет семантику проверки существования.

В качестве альтернативы используйте `INNER JOIN`, предварительно удалив повторяющиеся ключи с правой стороны:

```
$order_clients = (
    SELECT client_id
    FROM orders
    GROUP BY client_id
);

SELECT c.client_id, c.name
FROM clients AS c
INNER JOIN $order_clients AS o
ON o.client_id = c.client_id;
```

Правая сторона содержит не более одной строки для каждого ключа, поэтому соединение не дублирует строки слева. Применять `DISTINCT` к колонкам левой стороны не требуется: он может объединить одинаковые строки внешней выборки.

Для одного неопционального ключа запрос можно также переписать через `IN`:

```
SELECT c.client_id, c.name
FROM clients AS c
```

```
WHERE c.client_id IN (
    SELECT o.client_id
    FROM orders AS o
);
```

NOT EXISTS

Некоррелированный `NOT EXISTS` возвращает результат, обратный `EXISTS`: `true`, если подзапрос пуст, и `false`, если подзапрос содержит хотя бы одну строку. Его результат также не зависит от строки внешнего запроса.

Чтобы вернуть клиента, только если подходящих заказов нет, используйте `LEFT ONLY JOIN`:

```
SELECT c.client_id, c.name
FROM clients AS c
LEFT ONLY JOIN orders AS o
ON o.client_id = c.client_id;
```

Коррелированные подзапросы с агрегацией

Предварительно вычислите агрегат и присоедините его к внешней таблице:

```
$last_order = (
    SELECT client_id, MAX(created_at) AS last_order_at
    FROM orders
    GROUP BY client_id
);

SELECT c.client_id, o.last_order_at
FROM clients AS c
LEFT JOIN $last_order AS o
ON o.client_id = c.client_id;
```

См. также

- JOIN
- Выражения: IN
- Известные ограничения

ACTION

DEFINE ACTION

Задает именованное действие, которое представляют собой параметризуемый блок из нескольких выражений верхнего уровня.

Синтаксис

1. `DEFINE ACTION` — объявление действия.
2. Имя действия, по которому объявляемое действие доступно далее для вызова.
3. В круглых скобках — список имен параметров.
4. Ключевое слово `AS`.
5. Список выражений верхнего уровня.
6. `END DEFINE` — маркер последнего выражения внутри действия.

Один или более последних параметров могут быть помечены знаком вопроса `?` как необязательные. Если они не будут указаны при вызове, то им будет присвоено значение `NULL`.

DO

Выполняет `ACTION` с указанными параметрами.

Синтаксис

1. `DO` — выполнение действия.
2. Именованное выражение, по которому объявлено действие.
3. В круглых скобках — список значений для использования в роли параметров.

`EMPTY_ACTION` — действие, которое ничего не выполняет.

Пример

```
DEFINE ACTION $hello_world($name, $suffix?) AS
    $name = $name ?? ($suffix ?? "world");
    SELECT "Hello, " || $name || "!";
END DEFINE;

DO EMPTY_ACTION();
DO $hello_world(NULL);
DO $hello_world("John");
DO $hello_world(NULL, "Earth");
```

BEGIN .. END DO

Выполнение действия без его объявления (анонимное действие).

Синтаксис

1. `BEGIN`;
2. Список выражений верхнего уровня;
3. `END DO`.

Анонимное действие не может содержать параметров.

Пример

```
DO BEGIN
    SELECT 1;
    SELECT 2  -- здесь и в предыдущем примере ';' перед END можно не ставить
END DO
```

ALTER ASYNC REPLICATION

Вызов `ALTER ASYNC REPLICATION` изменяет параметры и состояние экземпляра асинхронной репликации.

Синтаксис

```
ALTER ASYNC REPLICATION <name> SET (option = value [, ...])
```

где:

- `name` — имя экземпляра асинхронной репликации.
- `SET (option = value [, ...])` — параметры асинхронной репликации.

Параметры

- `STATE` — состояние асинхронной репликации. Возможные значения:
  - `PAUSED`: приостановить применение изменений. Потоки изменений в источнике продолжают накапливать записи согласно политике доступности, заданной параметром `AVAILABILITY_PERIOD`. Уже подключённые объекты-реплики остаются доступны только для чтения.
  - `STANDBY`: запустить или возобновить репликацию изменений. Для репликации, созданной с `INITIAL_SCAN = FALSE`, подключает заранее восстановленные целевые таблицы. Перед запуском завершите импорт всех таблиц, как описано в руководстве по первоначальной синхронизации.
  - `DONE` — завершение процесса асинхронной репликации. Для принудительного завершения укажите также `FAILOVER_MODE = 'FORCE'`.
- `FAILOVER_MODE` — режим завершения репликации. Применимо только совместно с `STATE = 'DONE'`. Для `PAUSED` и `STANDBY` этот параметр не требуется. Возможные значения:
  - `FORCE` — принудительное переключение состояния.
- Настройки для аутентификации в базе-источнике одним из способов:
  - С помощью токена:
    - `TOKEN_SECRET_PATH` — секрет, содержащий токен.
  - С помощью логина и пароля:
    - `USER` — имя пользователя.
    - `PASSWORD_SECRET_PATH` — секрет, содержащий пароль.
  - С помощью сервисного агента:
    - `SERVICE_ACCOUNT_ID` — идентификатор сервисного аккаунта.
    - `INITIAL_TOKEN_SECRET_PATH` — секрет, содержащий токен от сервисного аккаунта. Используется для первоначальной инициализации.

Примеры

Следующий запрос запускает репликацию после импорта первоначальных данных:

```
ALTER ASYNC REPLICATION my_replication SET (STATE = 'STANDBY');
```

Следующий запрос приостанавливает применение изменений:

```
ALTER ASYNC REPLICATION my_replication SET (STATE = 'PAUSED');
```

Для возобновления используйте `STATE = 'STANDBY'` . При конечном `AVAILABILITY_PERIOD` учитывайте возраст уже накопленных сообщений, время приостановки и последующей обработки: пауза не останавливает отсчёт периода доступности. При `'UNLIMITED'` неподтверждённые изменения сохраняются до обработки; контролируйте свободное место в источнике.

Следующий запрос принудительно завершит процесс асинхронной репликации:

```
ALTER ASYNC REPLICATION my_replication SET (STATE = "DONE", FAILOVER_MODE = "FORCE");
```

Следующий запрос изменяет секрет:

```
ALTER ASYNC REPLICATION my_replication SET (TOKEN_SECRET_PATH = "my_token");
```

См. также

- Первоначальная синхронизация с помощью резервного копирования и восстановления
- CREATE ASYNC REPLICATION
- DROP ASYNC REPLICATION

ALTER DATABASE

Изменяет настройки базы данных.

Синтаксис

```
ALTER DATABASE path action;
```

Параметры

- `path` — путь к базе данных;
- `action` — любое действие по изменению базы данных, из описанных ниже:
  - Изменение владельца базы данных.
  - Изменение настроек базы данных.

Изменение владельца базы данных

Изменяет владельца базы данных. Выполнить операцию может только администратор базы данных.

Синтаксис

```
ALTER DATABASE path OWNER TO user_name;
```

Параметры

- `path` — путь к базе данных;
- `user_name` — имя пользователя, который станет владельцем базы данных.

Примеры

Владельцем базы данных `/Root/test` становится пользователь `user1`:

```
ALTER DATABASE `/Root/test` OWNER TO user1;
```

Изменение настроек базы данных

Изменяет настройки базы данных. Выполнить операцию может только администратор базы данных.

Синтаксис

```
ALTER DATABASE path SET (key = value, ...)
```

Параметры

- `path` — путь к базе данных.
- `key` — имя изменяемой настройки:
  - `MAX_PATHS` — максимальное количество путей (объектов в схеме) в базе данных.
  - `MAX_SHARDS` — максимальное количество таблеток в базе данных.
  - `MAX_CHILDREN_IN_DIR` — максимальное количество объектов в директории.
  - `MAX_SHARDS_IN_PATH` — максимальное количество таблеток, ассоциированных с одним объектом схемы. Например, максимальное количество партиций одной таблицы.
- `value` — значение изменяемой настройки.

Примеры

Изменение ограничения на максимальное количество путей (объектов в схеме) в базе данных `/Root/test`:

```
ALTER DATABASE `/Root/test` SET (MAX_PATHS = 20000);
```

ALTER GROUP

Команда `ALTER GROUP` добавляет или удаляет пользователей в указанной группе. Для одного оператора вы можете указать несколько пользователей.

Синтаксис:

```
ALTER GROUP group_name ADD USER user_name [, ... ]
ALTER GROUP group_name DROP USER user_name [, ... ]
```

- `group_name` — имя группы. Может содержать только строчные буквы латинского алфавита (a–z), цифры (0–9). Имя не должно быть пустым.
- `user_name` — имя пользователя.

Примеры

Успешно выполненные команды

Добавление и удаление пользователей:

```
ALTER GROUP group1 ADD USER root;
```

```
ALTER GROUP group1 DROP USER root;
```

Обработка ошибок

Корректно обработанные ошибки:

Ситуация	Сообщение об ошибке
Обращение к несуществующей группе	Group not found

Встроенные группы

YDB может иметь набор групп и пользователей уже с момента начального развёртывания.

Подробнее см. Первичные настройки безопасности и Начальная настройка безопасности кластера.

Такие пользователи и группы ничем не отличаются от пользователей и групп, созданных позже.

См. также

- CREATE GROUP
- DROP GROUP

ALTER TABLE

При помощи команды `ALTER TABLE` можно изменить состав колонок и дополнительные параметры строковых и колоночных таблиц. В одной команде можно указать несколько действий. В общем случае команда `ALTER TABLE` выглядит так:

```
ALTER TABLE table_name action1, action2, ..., actionN;
```

`action` — это любое действие по изменению таблицы, из описанных ниже:

- Переименование таблицы.
- Работа с колонками строковой и колоночной таблиц.
- Добавление или удаление потока изменений.
- Работа с индексами.
- Работа с группами колонок строковой таблицы.
- Изменение дополнительных параметров таблиц.
- Запуск принудительного компакшена.
- Изменение ограничения NOT NULL.

Управление индексами

Добавление индекса

`ADD INDEX` — добавляет индекс с указанным именем и типом для заданного набора колонок в строковых таблицах.
Грамматика:

```
ALTER TABLE `<table_name>`
  ADD INDEX `<index_name>`
    [GLOBAL|LOCAL]
    [UNIQUE]
    [SYNC|ASYNC]
    [USING <index_type>]
    ON ( <index_columns> )
    [COVER ( <cover_columns> )]
    [WITH ( <parameter_name> = <parameter_value>[, ...])]
  [,   ...]
```

- `GLOBAL/LOCAL` — глобальный или локальный индекс, в зависимости от типа индекса (`<index_type>`) может быть доступен только один из них:
  - `GLOBAL` — индекс, реализованный в виде отдельной таблицы или набора таблиц. Синхронное обновление такого индекса требует распределённых транзакций.
  - `LOCAL` — локальный индекс в рамках шарда колоночной или строковой таблицы, не требует распределённых транзакций при обновлении, однако не обеспечивает прюнинг при поиске.
- `<index_name>` — уникальное имя индекса, по которому будет возможно обращение к данным.
- `SYNC/ASYNC` — признак синхронности индекса.
  - `SYNC` - синхронный индекс. Значение по умолчанию.
  - `ASYNC` - асинхронный индекс.
- `UNIQUE` — признак уникального вторичного индекса. Уникальный индекс должен быть глобальным синхронным (`GLOBAL UNIQUE SYNC`) и не должен содержать конструкцию `USING <index_type>` .
- `<index_type>` - тип индекса, в настоящее время поддерживаются:
  - `secondary` — вторичный индекс. Для вторичных индексов доступен только режим `GLOBAL` . Это тип индекса по умолчанию.
  - `vector_kmeans_tree` — векторный индекс. Подробнее описан в разделе Векторный индекс.
  - `fulltext_plain` — базовый полнотекстовый индекс. Подробнее описан в Полнотекстовый индекс.
  - `fulltext_relevance` — полнотекстовый индекс со статистикой BM25 для расчёта релевантности. Подробнее описан в Полнотекстовый индекс.
  - `json` — JSON-индекс для ускорения предикатов `JSON_EXISTS` и `JSON_VALUE` по колонке типа `Json` или `JsonDocument` . Подробнее описан в JSON-индекс.
  - `bloom_filter` — локальный Блум-индекс. Доступен только `LOCAL` . См. ALTER TABLE ADD INDEX.
  - `bloom_ngram_filter` — локальный N-граммный Блум-индекс. Доступен только `LOCAL` . См. ALTER TABLE ADD INDEX.
  - `min_max` — локальный min/max-индекс. Доступен только `LOCAL` . См. ALTER TABLE ADD INDEX.
- `<index_columns>` — список имён колонок создаваемой таблицы через запятую, по которому определяется состав и порядок включения колонок в ключ индекса. Обязательно должен быть указан. Ключ индекса будет состоять из этих колонок с добавлением колонок первичного ключа таблицы.
- `<cover_columns>` — список имён колонок создаваемой таблицы через запятую, которые будут сохранены в индексе дополнительно к колонкам ключа индекса, давая возможность получить дополнительные данные без обращения за ними в таблицу. По умолчанию пуст.
- `<parameter_name>` и `<parameter_value>` — параметры индекса, специфичные для конкретного `<index_type>` . Некоторые параметры индекса нельзя задать при его создании. См. Изменение параметров индекса.

Также добавить вторичный индекс можно с помощью команды table index YDB CLI.

Параметры для всех типов индексов:

- `parallel` - максимальное число параллельных обработчиков на основе партиций, задействованных в построении индекса (целое число между `1` и `MaxBuildIndexShardsInFlight` из `SchemeShardConfig`).
  - Если параметр не указан, сейчас используется значение по умолчанию `32` или `MaxBuildIndexShardsInFlight` , если это значение меньше. `MaxBuildIndexShardsInFlight` по умолчанию равен `1000` . В будущих версиях логика выбора параллелизма по умолчанию может быть изменена.
  - Вы можете установить меньший лимит, чтобы снизить влияние построения индекса на производительность базы данных.
  - Вы также можете установить больший лимит, чтобы ускорить построение индекса, если у вас достаточно аппаратных ресурсов.

Параметры, специфичные для векторных индексов:

- общие параметры для всех векторных индексов:
  - `vector_dimension` - размерность вектора эмбеддинга (значение от 1 до 65536);
  - `vector_type` - тип значений вектора (`float` , float16, bfloat16, `uint8` , `int8` или `bit`);
  - `distance` - функция расстояния (`cosine` , `manhattan` или `euclidean`), взаимоисключающий с `similarity` ;
  - `similarity` - функция схожести (`inner_product` или `cosine`), взаимоисключающий с `distance` ;
- специфичные параметры для `vector_kmeans_tree` (подробнее о типе индекса):
  - `clusters` - количество центроидов для алгоритма k-means (значение от 2 до 2048);
  - `levels` - количество уровней в дереве (значение от 1 до 16);
  - `overlap_clusters` - максимальное число ближайших листовых кластеров, в которые добавляется каждый вектор; оно не должно превышать `clusters` . Если параметр не указан, YDB может автоматически включить перекрытие для больших наборов данных. Если перекрытие не включено, каждый вектор попадает в один кластер;

- `adaptive_clusters` - для индексов с фильтрацией автоматически выбирать количество кластеров для каждого значения фильтруемой колонки в зависимости от количества векторов у этого значения (`true` или `false`, по умолчанию `false`; включается при автоматическом подборе параметров). Подробнее см. Адаптивное количество кластеров;
- общее количество узлов в дереве, рассчитываемое как `clusters` в степени `levels`, должно быть не более чем 1073741824;
- произведение `vector_dimension` на `clusters` должно быть не более чем 4194304.

> **Примечание**
>
> Для векторных индексов параметры `vector_type` и `vector_dimension` можно не указывать, если таблица не пуста — они определяются автоматически по содержимому строк. Параметры `levels` и `clusters` также определяются автоматически, и для них таблица может быть пустой, но делать это крайне не рекомендуется так как дефолтные значения в этом случае `levels` =1, `clusters` =2, гораздо лучше создавать индекс по таблице куда уже загруженны данные, чтобы значения могли правильно определиться.

Параметры, специфичные для полнотекстовых индексов:

- общие параметры для всех полнотекстовых индексов:
  - `analyzer` - готовый анализатор: `standard` (стандартный токенизатор, нижний регистр и удаление английских стоп-слов), `snowball` (стандартный токенизатор, нижний регистр, удаление стоп-слов и лемматизация Snowball; требуется `language`) или `keyword` (весь текст становится одним токеном с сохранением регистра)
  - `tokenizer` - тип токенизатора (`standard`, `whitespace`, `alphanumeric` или `keyword`)
  - `use_filter_lowercase` - фильтр приведения к нижнему регистру (`true` или `false`)
  - `use_filter_stopwords` - удаление частотных слов, например `the` и `and` (`true` или `false`). Поддерживаются английский и русский языки; если `language` не указан, используется английский
  - `use_filter_length` - фильтр по длине токена (`true` или `false`); при значении `true` токены короче `filter_length_min` или длиннее `filter_length_max` не индексируются
  - `filter_length_min` - минимальная длина токена (положительное целое); применяется только при `use_filter_length=true`
  - `filter_length_max` - максимальная длина токена (положительное целое); применяется только при `use_filter_length=true`
  - `use_filter_snowball` - фильтр лемматизации Snowball (`true` или `false`)
  - `language` - язык или список языков через запятую для лемматизатора Snowball. Пробелы вокруг запятых допустимы, например `"english, russian"`. Готовый анализатор `snowball` дополнительно удаляет стоп-слова, поэтому вместе с ним можно указать только `english` и `russian`
  - `use_filter_ngram` - фильтр N-грамм (`true` или `false`)
  - `use_filter_edge_ngram` - фильтр краевых N-грамм (`true` или `false`)
  - `filter_ngram_min_length` - минимальная длина N-граммы (положительное целое)
  - `filter_ngram_max_length` - максимальная длина N-граммы (положительное целое)

Параметры локальных блум-индексов

- `bloom_filter`
  - `false_positive_probability` — целевая вероятность ложноположительного срабатывания фильтра: доля фрагментов, которые фильтр не отсечёт, хотя искомого значения в них нет (диапазон `(0, 1)`). Меньшее значение уменьшает число лишних чтений, но увеличивает размер индекса.
    - Если параметр не указан, для строковых таблиц используется `0.0001`, для колоночных — `0.1`. Более строгий порог по умолчанию в строковых таблицах соответствует OLTP-сценарию точечных чтений; в колоночных — аналитическим сканам, где допустим больший компромисс между размером индекса и отсечением фрагментов.
  - Индексируемые колонки — типы YQL, для которых определено сравнение на равенство, кроме `Yson`, `Json` и `JsonDocument` (см. операторы сравнения). В колоночной таблице допускается одна колонка; в строковой — несколько. На строковых таблицах индексируемые колонки должны образовывать левый префикс первичного ключа.
- `bloom_ngram_filter` (колонки `String`, `Utf8`; только колоночные таблицы)
  - `ngram_size` — размер n-граммы, целое число от `3` до `8` (по умолчанию `3`).
  - `false_positive_probability` — целевая вероятность ложноположительного срабатывания (диапазон `(0, 1)`; по умолчанию `0.1`).
  - `case_sensitive` — учёт регистра при построении n-грамм: `true` или `false` (по умолчанию `true`).

Параметры локального min_max-индекса

У min_max-индекса нет специфичных параметров `WITH (...)`.

Ограничения

Операция `ADD INDEX` для создания глобальных вторичных (`GLOBAL`, `UNIQUE` и т.п.) и векторных индексов поддерживается только для строковых таблиц. Для колоночных таблиц через `ADD INDEX` поддерживаются только локальные индексы: блум-индекс и min_max-индекс.

Особенности локальных блум-индексов:

- Индекс всегда локальный (`LOCAL`); глобального варианта нет.
- В запросах не используется синтаксис `VIEW <index>` (в отличие, например, от полнотекстовых индексов).
- Фильтр применяется при чтении только к тем фрагментам данных, для которых при записи или слиянии порций уже сохранён блок индекса, для остальных фрагментов пропуск по этому индексу не выполняется.
- На строковых таблицах `bloom_filter` реализован как **префиксный фильтр Блума**: индексируемые колонки должны образовывать левый префикс первичного ключа. Фильтр строится по этому префиксу ключа и ускоряет точечные чтения и сканы по диапазону, ограничивающие префикс. Два индекса `bloom_filter` с одинаковой длиной префикса не допускаются.

Ограничения

- Секции `COVER (...)` и дополнительные колонки индекса не поддерживаются.
- Для колоночных таблиц индексируемая колонка должна быть одна. Для строковых таблиц допускается несколько индексируемых колонок.
- Для строковых таблиц тип индекса `bloom_ngram_filter` не поддерживается.
- На строковых таблицах индексируемые колонки `bloom_filter` должны образовывать левый префикс первичного ключа. Наборы колонок, не являющиеся префиксом, отклоняются.
- На строковых таблицах два индекса `bloom_filter` не могут иметь одинаковую длину префикса (один и тот же набор ведущих колонок первичного ключа).

Особенности локального min_max-индекса:

- Индекс всегда локальный (`LOCAL`); глобального варианта нет.
- В запросах не используется синтаксис `VIEW <index>` (в отличие, например, от полнотекстовых индексов).
- Фильтр применяется при чтении только к тем фрагментам данных, для которых при записи или слиянии уже вычислены и сохранены вместе с данными таблицы минимальное и максимальное значения индексируемой колонки; для остальных фрагментов пропуск по этому индексу не выполняется.

Ограничения

- Поддерживаются только для колоночных таблиц.
- В `ON (...)` должна быть указана ровно одна колонка.
- `COVER (...)` и дополнительные колонки данных не поддерживаются.
- Специфичные параметры `WITH (...)` не поддерживаются.
- `ALTER INDEX` для min_max-индекса не поддерживается.
- Колонки типов `Json` и `JsonDocument` не поддерживаются.

Примеры

Вторичный индекс:

```
ALTER TABLE `series`
  ADD INDEX `title_index`
  GLOBAL ON (`title`);
```

Векторный индекс:

```
ALTER TABLE `series`
  ADD INDEX emb_cosine_idx GLOBAL SYNC USING vector_kmeans_tree
  ON (embedding) COVER (title)
  WITH (
    distance="cosine", vector_type="float", vector_dimension=512
  );
```

Полнотекстовый индекс:

```
ALTER TABLE `series`
  ADD INDEX ft_idx GLOBAL USING fulltext_plain
  ON (title)
  WITH (tokenizer=standard, use_filter_lowercase=true);
```

JSON-индекс:

```
ALTER TABLE `series`
  ADD INDEX json_idx GLOBAL USING json
  ON (metadata);
```

Блум-индекс:

```
ALTER TABLE `/Root/Table`
  ADD INDEX idx_bloom LOCAL USING bloom_filter
  ON (resource_id)
  WITH (false_positive_probability = 0.01);
```

Блум n-граммный индекс:

```
ALTER TABLE `/Root/Table`
  ADD INDEX idx_ngram LOCAL USING bloom_ngram_filter
  ON (message)
  WITH (
    ngram_size = 3,
    false_positive_probability = 0.01,
    case_sensitive = true
  );
```

min_max-индекс:

```
ALTER TABLE `/Root/Table`
  ADD INDEX idx_created_at LOCAL USING min_max
  ON (created_at);
```

Изменение параметров индекса

Индексы имеют параметры, зависящие от типа, которые можно настраивать. Глобальные индексы, синхронные или асинхронные, реализованы в виде скрытых таблиц, и их параметры автоматического партиционирования и реплик можно регулировать так же, как и настройки обычных таблиц.

> **Примечание**
>
> В настоящее время задание настроек партиционирования вторичных индексов при создании индекса не поддерживается ни в операторе `ALTER TABLE ADD INDEX`, ни в операторе `CREATE TABLE INDEX`.

```
ALTER TABLE <table_name> ALTER INDEX <index_name> SET <setting_name> <value>;
ALTER TABLE <table_name> ALTER INDEX <index_name> SET (<setting_name_1> = <value_1>, ...);
```

- `<table_name>` - имя таблицы, индекс которой нужно изменить.
- `<index_name>` - имя индекса, который нужно изменить.
- `<setting_name>` - имя изменяемого параметра. Набор допустимых параметров зависит от типа индекса:
  - для глобальных вторичных индексов:
    - AUTO_PARTITIONING_BY_SIZE
    - AUTO_PARTITIONING_BY_LOAD
    - AUTO_PARTITIONING_PARTITION_SIZE_MB
    - AUTO_PARTITIONING_MIN_PARTITIONS_COUNT
    - AUTO_PARTITIONING_MAX_PARTITIONS_COUNT
    - READ_REPLICAS_SETTINGS
  - для локальных блум-индексов (см. Параметры локальных блум-индексов):
    - `FALSE_POSITIVE_PROBABILITY`
    - `NGRAM_SIZE` и `CASE_SENSITIVE` (только для `bloom_ngram_filter`)
  - min_max-индекс не поддерживает `ALTER INDEX`.

> **Примечание**
>
> Операция `RESET` для `ALTER INDEX` не поддерживается.

- `<value>` - новое значение параметра. Возможные значения включают:
  - `ENABLED` или `DISABLED` для параметров `AUTO_PARTITIONING_BY_SIZE` и `AUTO_PARTITIONING_BY_LOAD`
  - `"PER_AZ:<count>"` или `"ANY_AZ:<count>"` где `<count>` — число реплик для `READ_REPLICAS_SETTINGS`
  - для остальных параметров — целое число типа `Uint64`
  - для `FALSE_POSITIVE_PROBABILITY` — число с плавающей точкой в диапазоне `(0, 1)`; меньшее значение обычно уменьшает число ложноположительных срабатываний, но увеличивает размер индекса
  - для `NGRAM_SIZE` — целое число в диапазоне от `3` до `8` (обычно рекомендуется начинать с `3`)
  - для `CASE_SENSITIVE` — `true` или `false`

Пример

Код из следующего примера включает автоматическое партиционирование по нагрузке для индекса с именем `title_index` в таблице `series`, устанавливает минимальное количество партиций равным 5 и запускает по одной реплике в каждой зоне доступности (AZ) для каждой партиции:

```
ALTER TABLE `series` ALTER INDEX `title_index` SET (
    AUTO_PARTITIONING_BY_LOAD = ENABLED,
    AUTO_PARTITIONING_MIN_PARTITIONS_COUNT = 5,
    READ_REPLICAS_SETTINGS = "PER_AZ:1"
);
```

Для локальных блум-индексов можно также менять специфичные для них параметры, например:

```
ALTER TABLE `/Root/Table` ALTER INDEX idx_ngram SET (
    ngram_size = 4,
    false_positive_probability = 0.005,
    case_sensitive = false
);
```

Перестроение векторного индекса

`REBUILD INDEX` создаёт новую версию существующего векторного индекса по данным таблицы и атомарно заменяет старый индекс с сохранением имени. Используйте команду для пересчёта кластеров после изменения распределения данных или для изменения параметров кластеризации.

Операция поддерживает только индексы `vector_kmeans_tree`, включая индексы с фильтрацией и покрывающие индексы. Индекс должен существовать и находиться в состоянии `Ready`, то есть его предыдущее построение должно быть завершено. Проверить наличие и тип индекса можно командой scheme describe.

Чтобы перестроить индекс с текущими настройками, выполните:

```
ALTER TABLE `my_table` REBUILD INDEX `my_vector_index`;
```

Чтобы изменить параметры кластеризации при перестроении, добавьте `WITH`:

```
ALTER TABLE `my_table` REBUILD INDEX `my_vector_index`
WITH (clusters = 128, levels = 2);
```

В `WITH` можно указать следующие параметры:

Параметр	Описание
`clusters`	Количество кластеров. Целое число от `2` до `2048`.
`levels`	Количество уровней дерева. Целое число от `1` до `16`.
`overlap_clusters`	Количество ближайших кластеров последнего уровня, в которые добавляется каждый вектор. Если параметр задан, допустимо целое число от `2` до `2048`, но не больше значения `clusters`. Если не задан, сохраняется текущее значение; при отсутствии значения в существующем индексе используется `1`.
`overlap_ratio`	Неотрицательный порог отношения расстояний, ограничивающий добавление вектора в дополнительные кластеры; значение `0` отключает порог.
`adaptive_clusters`	Автоматический выбор количества кластеров для каждого значения фильтруемой колонки в индексе с фильтрацией. Принимает `true` или `false`.
`parallel`	Максимальное число обработчиков партиций, участвующих в перестроении. Ограничения и значение по умолчанию совпадают с `ADD INDEX`.

Параметры кластеризации, не указанные в `WITH`, сохраняют текущие значения. В частности, если не указать `clusters` и `levels`, новые значения для текущего размера таблицы не будут подобраны автоматически. При перестроении действуют те же ограничения параметров векторного индекса, что и при создании.

Индекс сохраняет набор индексируемых и покрываемых колонок. Функция расстояния или сходства, тип элементов вектора и размерность вектора наследуются от существующего индекса. Не указывайте `distance`, `similarity`, `vector_type` или `vector_dimension` в `WITH`: команда отклоняет эти параметры, даже если их значения совпадают с текущими. Для их изменения создайте отдельный индекс.

Существующий индекс остаётся доступным для запросов и продолжает получать обновления таблицы, пока строится новая версия. После успешного построения YDB атомарно переключается на новую версию. Запросы продолжают использовать прежнее имя индекса. Если перестроение завершится ошибкой или будет отменено до замены, старый индекс останется доступным. Перестроение может продолжиться после перезапуска таблетки управления схемой.

Замена индекса может сделать кешированные планы запросов неактуальными. Во время переключения запрос к индексу может получить временный статус `ABORTED`. Если это происходит при завершении операции перестроения, повторите запрос целиком с помощью стандартного механизма повторных попыток SDK.

> **Важно**
>
> Новая версия строится по снимку данных и наследует ограничение консистентности при построении векторных индексов. Обновления таблицы во время перестроения могут не попасть в новый индекс. Если требуется полная консистентность, приостановите запись из приложений и дождитесь успешного завершения перестроения. Это временное ограничение планируется устранить в одной из следующих версий YDB.

Удаление индекса

`DROP INDEX` — удаляет индекс с указанным именем. Приведенный ниже код удалит индекс с именем `title_index`.

```
ALTER TABLE `series` DROP INDEX `title_index`;
```

Также удалить индекс можно с помощью команды table index YDB CLI.

Переименование вторичного индекса

`RENAME INDEX` — переименовывает индекс с указанным именем. Если индекс с новым именем существует, будет возвращена ошибка.

Возможность атомарной замены индекса под нагрузкой поддерживается командой ydb table index rename YDB CLI и специализированными методами YDB SDK.

Это относится к глобальным вторичным индексам (скрытая индексная таблица и режим `--replace`). Локальные блум-индексы к такой атомарной замене под нагрузкой не применимы.

Пример переименования индекса:

```
ALTER TABLE `series` RENAME INDEX `title_index` TO `title_index_new`;
```

Изменение колонок

YDB поддерживает возможность добавлять колонки в строковые и колоночные таблицы, удалять неключевые колонки из таблиц, а также изменять свойства существующих колонок.

ADD COLUMN

Строит новую колонку с указанными именем, типом и опциями для указанной таблицы.

```
ALTER TABLE table_name ADD COLUMN column_name column_data_type [FAMILY <family_name>] [NULL | NOT NULL] [DEFA
ULT <default_value>] [COMPRESSION([algorithm=<algorithm_name>[, level=<value>]])] [ENCODING([OFF|DICT])];
```

При добавлении колонки с ограничением `NOT NULL` в строковую таблицу необходимо также указать `DEFAULT`, чтобы заполнить новую колонку в существующих строках. В колоночную таблицу добавить колонку с ограничением `NOT NULL` нельзя, поскольку значения `DEFAULT` для таких таблиц не поддерживаются.

Параметры запроса

table_name

Путь таблицы, для которой требуется добавить новую колонку.

column_name

Имя колонки, которая будет добавлена в указанную таблицу. При выборе имени для колонки учитывайте общие правила именования колонок.

column_data_type

Тип данных колонки. Полный список типов данных, которые поддерживает YDB доступен в разделе Типы данных YQL.

FAMILY <family_name> (настройка колонки)

> **Важно**
>
> Поддерживается только для строковых таблиц.

Указание принадлежности данной колонки к указанной группе колонок. Подробнее в разделе Группы колонок.

DEFAULT <default_value>

> **Важно**
>
> Опция `DEFAULT` поддерживается:
>
> - Только для строковых таблиц.
> - Только с литеральными значениями.

Позволяет задать значение по умолчанию для колонки. Если при вставке строки значение для данной колонки не указано, будет использовано указанное значение по умолчанию. Значение по умолчанию должно соответствовать типу данных колонки.

Конструкция `DEFAULT false NOT NULL` недопустима по причине неоднозначности интерпретации. В таком случае следует использовать перечисление через запятые или изменить порядок опций.

NULL

Опция колонки, которое разрешает запись `NULL` в качестве значений колонки. Это является поведением по умолчанию.

NOT NULL

Ограничение колонки, которое запрещает запись `NULL` в качестве значений колонки. По умолчанию это ограничение отсутствует.

COMPRESSION([algorithm=<algorithm_name>[, level=]])

> **Важно**
>
> Поддерживается только для колоночных таблиц.

Для колонок можно задать следующие параметры сжатия:

- `algorithm` — алгоритм сжатия данных. Допустимые значения: `off` (отключение сжатия), `lz4`, `zstd`.
- `level` — уровень сжатия, поддерживается только для алгоритма `zstd` (допустимы значения от 0 до 22).

Если `COMPRESSION()` указан без параметров, для колонки используется сжатие по умолчанию. Сейчас это `lz4`; в будущих версиях появится возможность настраивать сжатие по умолчанию на уровне кластера или таблицы.

ENCODING([OFF|DICT])

> **Важно**
>
> Поддерживается только для колоночных таблиц.

Позволяет задать способ кодирования данных колонки.

Доступные варианты:

- `ENCODING(DICT)` — включает словарное кодирование (dictionary encoding). Повторяющиеся значения заменяются небольшими целочисленными идентификаторами, а сами значения хранятся в словаре. Словарное кодирование эффективно для колонок с низкой кардинальностью (небольшим количеством уникальных значений). Позволяет сократить объем хранимых данных и ускорить выполнение некоторых операций. Поддерживается только для сравнимых типов данных, таких как `String`, `Timestamp`, `UInt64` и других. Для несравнимых типов, таких как `Json`, `JsonDocument` или `Yson`, использование `ENCODING(DICT)` приведет к ошибке.
- `ENCODING(OFF)` — отключает специальное кодирование. Данные будут храниться в стандартном формате без дополнительного кодирования.

Если `ENCODING()` задается без параметров, то для колонки будет использоваться кодирование по умолчанию. Сейчас это `OFF`; в будущих версиях появится возможность настраивать кодирование по умолчанию на уровне базы или таблицы.

Пример

Приведённый ниже код добавит к таблице `episodes` колонку `views` с типом данных `Uint64`.

```
ALTER TABLE episodes ADD COLUMN views Uint64;
```

Приведённый ниже код добавит к таблице `episodes` колонку `rate` с типом данных `Double` и значением по умолчанию `5.0`.

```
ALTER TABLE episodes ADD COLUMN rate Double NOT NULL DEFAULT 5.0;
ALTER TABLE episodes ADD COLUMN rate Double (DEFAULT 5.0, NOT NULL); -- альтернативный синтаксис
```

ALTER COLUMN

Изменяет свойства существующей колонки в указанной таблице. Изменение свойства происходит без пересоздания колонки. Некоторые свойства применяются только к свежим записанным данным или в процессе компакшена (детали можно найти в описании конкретного свойства)

```
ALTER TABLE table_name ALTER COLUMN column_name SET [FAMILY <family_name>]{% if feature_alter_column_not_null
== true %} [NOT NULL]{% endif %} [DEFAULT <default_value>] [COMPRESSION([algorithm=<algorithm_name>[, level=<
value>]])] [ENCODING([OFF|DICT])];
ALTER TABLE table_name ALTER COLUMN column_name DROP [FAMILY] [NOT NULL] [DEFAULT] [COMPRESSION] [ENCODING];
```

Параметры запроса

table_name

Путь к таблице, в которой требуется изменить колонку.

column_name

Имя колонки, которая будет изменена в указанной таблице.

SET

Установить свойство для указанной колонки.

DROP

Удалить свойство для указанной колонки.

FAMILY <family_name> (настройка колонки)

> **Важно**
>
> Поддерживается только для строковых таблиц.

Указание принадлежности данной колонки к указанной группе колонок. Подробнее в разделе Группы колонок.

DEFAULT <default_value>

> **Важно**
>
> Опция `DEFAULT` поддерживается:
>
> - Только для строковых таблиц.
> - Только с литеральными значениями.

Позволяет задать значение по умолчанию для колонки. Если при вставке строки значение для данной колонки не указано, будет использовано указанное значение по умолчанию. Значение по умолчанию должно соответствовать типу данных колонки.

NOT NULL

Доступны следующие операции:

- `DROP NOT NULL` — снимает ограничение `NOT NULL` с колонки, снова разрешая значения `NULL`.

Подробнее см. в разделе «Изменение ограничения `NOT NULL`».

COMPRESSION([algorithm=<algorithm_name>[, level=]])

> **Важно**
>
> Поддерживается только для колоночных таблиц.

Для колонок можно задать следующие параметры сжатия:

- `algorithm` — алгоритм сжатия данных. Допустимые значения: `off` (отключение сжатия), `lz4`, `zstd`.
- `level` — уровень сжатия, поддерживается только для алгоритма `zstd` (допустимы значения от 0 до 22).

Если `COMPRESSION()` указан без параметров, для колонки используется сжатие по умолчанию. Сейчас это `lz4`; в будущих версиях появится возможность настраивать сжатие по умолчанию на уровне кластера или таблицы.

ENCODING([OFF|DICT])

> **Важно**
>
> Поддерживается только для колоночных таблиц.

Позволяет задать способ кодирования данных колонки.

Доступные варианты:

- `ENCODING(DICT)` — включает словарное кодирование (dictionary encoding). Повторяющиеся значения заменяются небольшими целочисленными идентификаторами, а сами значения хранятся в словаре. Словарное кодирование эффективно для колонок с низкой кардинальностью (небольшим количеством уникальных значений). Позволяет сократить объем хранимых данных и ускорить выполнение некоторых операций. Поддерживается только для сравнимых типов данных, таких как `String`, `Timestamp`, `UInt64` и других. Для несравнимых типов, таких как `Json`, `JsonDocument` или `Yson`, использование `ENCODING(DICT)` приведет к ошибке.
- `ENCODING(OFF)` — отключает специальное кодирование. Данные будут храниться в стандартном формате без дополнительного кодирования.

Если `ENCODING()` задается без параметров, то для колонки будет использоваться кодирование по умолчанию. Сейчас это `OFF`; в будущих версиях появится возможность настраивать кодирование по умолчанию на уровне базы или таблицы.

В одном запросе `ALTER TABLE` можно указать несколько действий `ALTER COLUMN` через запятую.

Примеры

Приведённый ниже код задаёт значение по умолчанию для колонки `rate` в таблице `episodes`.

```
ALTER TABLE episodes ALTER COLUMN rate SET DEFAULT 5.0;
```

Приведённый ниже код одним запросом изменяет значения по умолчанию для нескольких колонок таблицы — у `col_1` и `col_2` устанавливает новые значения, у `col_3` сбрасывает значение по умолчанию.

```
ALTER TABLE default_columns
    ALTER COLUMN col_1 SET DEFAULT "new_a"u,
    ALTER COLUMN col_2 SET DEFAULT 99,
    ALTER COLUMN col_3 DROP DEFAULT;
```

Сброс настроек сжатия колонки

```
ALTER TABLE compressed_table ALTER COLUMN info SET COMPRESSION();
```

После выполнения запроса для колонки снова действует алгоритм сжатия по умолчанию (см. описание опции `COMPRESSION` выше).

Включение словарного кодирования на колонке

```
ALTER TABLE movies ALTER COLUMN genre SET ENCODING(DICT);
```

DROP COLUMN

Удаляет колонку из таблицы с указанным именем.

```
ALTER TABLE table_name DROP COLUMN column_name;
```

Параметры запроса

table_name

Путь к таблице, в которой требуется удалить колонку.

column_name

Имя колонки, которая будет удалена из указанной таблицы.

Пример

Приведённый ниже код удалит колонку `views` из таблицы `episodes`.

```
ALTER TABLE episodes DROP COLUMN views;
```

Создание и изменение групп колонок

Важно

Поддерживается только для строковых таблиц.

Механизм групп колонок позволяет увеличить производительность операций неполного чтения строк путем разделения хранения колонок строковой таблицы на несколько групп. Наиболее часто используемый сценарий — организация хранения редко используемых атрибутов в отдельной группе колонок.

Создание группы колонок

`ADD FAMILY` — создаёт новую группу колонок в строковой таблице. Приведенный ниже код создаст в таблице `series_with_families` группу колонок `family_small`.

```
ALTER TABLE series_with_families ADD FAMILY family_small (
    DATA = "ssd",
    COMPRESSION = "off"
);
```

Изменение групп колонок

При помощи команды `ALTER COLUMN` можно изменить группу колонок для указанной колонки. Приведённый ниже код для колонки `release_date` в таблице `series_with_families` сменит группу колонок на `family_small`.

```
ALTER TABLE series_with_families ALTER COLUMN release_date SET FAMILY family_small;
```

Две предыдущие команды можно объединить в один вызов `ALTER TABLE`. Приведённый ниже код создаст в таблице `series_with_families` группу колонок `family_small` и установит её для колонки `release_date`.

```
ALTER TABLE series_with_families
  ADD FAMILY family_small (
      DATA = "ssd",
      COMPRESSION = "off"
  ),
  ALTER COLUMN release_date SET FAMILY family_small;
```

При помощи команды `ALTER FAMILY` можно изменить параметры группы колонок.

Изменение типа хранилища

Приведённый ниже код для группы колонок `default` в таблице `series_with_families` сменит тип хранилища на `rot`:

```
ALTER TABLE series_with_families ALTER FAMILY default SET DATA "rot";
```

Примечание

Доступные типы устройств хранения зависят от конфигурации кластера YDB.

Изменение кодека сжатия

Приведённый ниже код для группы колонок `default` в таблице `series_with_families` сменит кодек сжатия на `lz4`:

```
ALTER TABLE series_with_families ALTER FAMILY default SET COMPRESSION "lz4";
```

Изменение режима кэширования

При переключении режима кэширования на `in_memory` для существующей таблицы через команду `ALTER TABLE`, все страницы, которые ещё не находятся в памяти, будут подгружены автоматически.

Если для таблицы ранее был активирован режим `in_memory`, а затем через `ALTER TABLE` установлен режим кэширования `regular`, все находящиеся в памяти страницы сохраняются, но впоследствии могут вытесняться из памяти согласно общей политике кэширования.

Приведённый ниже код для группы колонок `default` в таблице `series_with_families` сменит режим кэширования на `in_memory`:

```
ALTER TABLE series_with_families ALTER FAMILY default SET CACHE_MODE "in_memory";
```

Могут быть указаны все параметры группы колонок, описанные в команде `CREATE TABLE`

Изменение ограничения `NOT NULL`

`NOT NULL` — ограничение целостности данных на уровне колонки, запрещающее запись значений `NULL`. Оно гарантирует, что колонка всегда содержит действительное значение.

Сброс `NOT NULL`

`DROP NOT NULL` снимает ограничение `NOT NULL` с указанной колонки.

Например, следующий запрос снимет ограничение `NOT NULL` с колонки `column_name` в таблице `table_name`:

```
ALTER TABLE table_name ALTER COLUMN column_name DROP NOT NULL;
```

`DROP NOT NULL` можно применить к неключевым колонкам как строковых, так и колоночных таблиц.

См. также

- ALTER COLUMN

Изменение дополнительных параметров таблиц

Большинство параметров строковых и колоночных таблиц в YDB, приведенных на странице описания таблицы, можно изменить командой `ALTER`.

В общем случае команда для изменения любого параметра таблицы выглядит следующим образом:

```
ALTER TABLE table_name SET (key = value);
```

`key` — имя параметра, `value` — его новое значение.

Пример изменения параметра `TTL`, отвечающего за время жизни записей в таблицы:

```
ALTER TABLE series SET (TTL = Interval("PT0S") ON expire_at);
```

Сброс дополнительных параметров таблицы

Некоторые параметры таблиц в YDB, приведенные на странице описания таблицы, можно сбросить командой `ALTER`. Команда для сброса параметра таблиц выглядит следующим образом:

```
ALTER TABLE table_name RESET (key);
```

`key` — имя параметра.

Например, такая команда сбросит (удалит) настройки TTL для строковых или колоночных таблиц:

```
ALTER TABLE series RESET (TTL);
```

Добавление или удаление потока изменений

> **Важно**
>
> Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

`ADD CHANGEFEED <name> WITH (option = value[, ...])` — добавляет поток изменений (changefeed) с указанным именем и параметрами.

Параметры потока изменений

- `MODE` — режим работы. Указывает, что именно будет записано в поток при каждом изменении данных в таблице:
  - `KEYS_ONLY` — будут записаны только компоненты первичного ключа и признак изменения.
  - `UPDATES` — будут записаны значения изменившихся столбцов, получившиеся в результате изменения.
  - `NEW_IMAGE` — будут записаны значения всех столбцов, получившиеся в результате изменения.
  - `OLD_IMAGE` — будут записаны значения всех столбцов, предшествующие изменению.
  - `NEW_AND_OLD_IMAGES` - комбинация режимов `NEW_IMAGE` и `OLD_IMAGE`. Будут записаны значения всех столбцов *до* и *в результате* изменения.
- `FORMAT` — формат данных, в котором будут записаны данные:
  - `JSON` — записывать данные в формате JSON.
  - `DEBEZIUM_JSON` — записывать данные в JSON-формате, аналогичном Debezium формату.
- `VIRTUAL_TIMESTAMPS` — включение-выключение виртуальных меток времени.
- `USER_SIDS` — включение-выключение выгрузки идентификаторов безопасности SID. По умолчанию выключено.
- `TRACE_IDS` — включение-выключение идентификатора трассировки OpenTelemetry. По умолчанию выключено.
- `BARRIERS_INTERVAL` — периодичность выгрузки барьеров. Тип значения — `Interval`. По умолчанию выключено.
- `RETENTION_PERIOD` — время хранения записей. Тип значения — `Interval`, значение по умолчанию — 24 часа (`Interval('PT24H')`).
- `TOPIC_AUTO_PARTITIONING` — режим автопартиционирования топика:
  - `ENABLED` — для потока изменений будет создан автопартиционированный топик. Количество партиций в таком топике увеличивается автоматически по мере роста скорости обновления таблицы. Параметры автопартиционирования топика можно настроить.
  - `DISABLED` — для потока изменений будет создан топик без автопартиционирования. Это значение по умолчанию.
- `TOPIC_MIN_ACTIVE_PARTITIONS` — количество партиций топика. По умолчанию количество партиций топика равно количеству партиций таблицы. Для автопартиционированных топиков количество партиций увеличивается по мере роста скорости обновления таблицы. Если при создании ченджфида опция `TOPIC_AUTO_PARTITIONING` была отключена (`DISABLED`), то число партиций в топике, связанном с таким ченджфидом, впоследствии изменить нельзя.
- `INITIAL_SCAN` — включение-выключение первоначального сканирования таблицы. По умолчанию выключено.

Приведенный ниже код добавит поток изменений с именем `updates_feed`, в который будут выгружаться значения изменившихся столбцов таблицы в формате JSON:

```
ALTER TABLE `series` ADD CHANGEFEED `updates_feed` WITH (
    FORMAT = 'JSON',
    MODE = 'UPDATES'
);
```

Записи в таком потоке изменений будут храниться в течение 24 часов (значение по умолчанию). Код из следующего примера создаст поток изменений с хранением записей в течение 12 часов:

```
ALTER TABLE `series` ADD CHANGEFEED `updates_feed` WITH (
    FORMAT = 'JSON',
    MODE = 'UPDATES',
    RETENTION_PERIOD = Interval('PT12H')
);
```

Пример создания потока изменений с включенными виртуальными метками времени:

```
ALTER TABLE `series` ADD CHANGEFEED `updates_feed` WITH (
    FORMAT = 'JSON',
    MODE = 'UPDATES',
    VIRTUAL_TIMESTAMPS = TRUE
);
```

Пример создания потока изменений с виртуальными метками времени и барьерами раз в 10 секунд:

```
ALTER TABLE `series` ADD CHANGEFEED `updates_feed` WITH (
    FORMAT = 'JSON',
    MODE = 'UPDATES',
    VIRTUAL_TIMESTAMPS = TRUE,
    BARRIERS_INTERVAL = Interval('PT10S')
);
```

Пример создания потока изменений с первоначальным сканированием:

```
ALTER TABLE `series` ADD CHANGEFEED `updates_feed` WITH (
    FORMAT = 'JSON',
    MODE = 'UPDATES',
```

```
    INITIAL_SCAN = TRUE
);
```

Пример создания потока изменений с автопартиционированием:

```
ALTER TABLE `series` ADD CHANGEFEED `updates_feed` WITH (
    FORMAT = 'JSON',
    MODE = 'UPDATES',
    TOPIC_AUTO_PARTITIONING = 'ENABLED',
    TOPIC_MIN_ACTIVE_PARTITIONS = 2
);
```

`DROP CHANGEFEED` — удаляет поток изменений с указанным именем. Приведенный ниже код удалит changefeed с именем `updates_feed`:

```
ALTER TABLE `series` DROP CHANGEFEED `updates_feed`;
```

Переименование таблицы

```
ALTER TABLE old_table_name RENAME TO new_table_name;
```

Примечание

При выборе имени для таблицы учитывайте общие правила именования схемных объектов.

Если таблица с новым именем существует, будет возвращена ошибка. Возможность транзакционной подмены таблицы под нагрузкой поддерживается специализированными методами в CLI и SDK.

Важно

Если в YQL запросе содержится несколько команд `ALTER TABLE ... RENAME TO ...`, то каждая будет выполнена в режиме автокоммита в отдельной транзакции. С точки зрения внешнего процесса, таблицы будут переименованы последовательно одна за другой. Чтобы переименовать несколько таблиц в одной транзакции, используйте специализированные методы, доступные в CLI и SDK.

Переименование может использоваться для перемещения таблицы из одной директории внутри БД в другую, например:

```
ALTER TABLE `table1` RENAME TO `/backup/table1`;
```

Запуск принудительного компакшена таблицы

```
ALTER TABLE table_name COMPACT [WITH (key = value [, ...])];
```

> **Важно**
>
> Поддерживается только для строковых таблиц.

YDB автоматически производит компакшен таблиц и обычно не требует ручного вмешательства.
Однако, при помощи команды `ALTER TABLE ... COMPACT` можно явно запустить компакшен данных заданной таблицы, не дожидаясь автоматического уплотнения. Команда полезна, например, в следующих сценариях:

- было выполнено массовое удаление данных и хочется быстрее освободить занимаемое место;
- были изменены настройки сжатия или групп колонок, и хочется быстрее применить их к уже существующим данным.

Команда создаёт фоновую операцию и синхронно ждёт её завершения. Ожидание можно прервать, тогда операция продолжит выполняться в фоне. Управлять фоновыми операциями компакшена можно через CLI и YDB UI.

Параметры

В блоке `WITH` можно указать следующие параметры:

- `CASCADE` — компактить ли вместе с таблицей её вторичные индексы. Допустимые значения: `TRUE` и `FALSE`. Значение по умолчанию — `FALSE` (компактится только сама таблица).
- `PARALLEL` — максимальное число партиций, которые могут одновременно компактиться в рамках этой операции. Допустимые значения — положительные целые числа. Значение по умолчанию — 1. Общий лимит на одновременно работающие ручные компакшены для всей базы задаётся отдельно в конфигурации кластера YDB.

Поведение

- За ходом запущенных операций компакшена можно следить с помощью CLI-команды `ydb operation list compaction`. С помощью соответствующих команд можно получить статус конкретной операции, отменить операцию или удалить запись завершённой операции.
- Также просматривать операции и управлять ими можно с помощью YDB UI. Для этого нужно перейти на страницу базы данных, открыть вкладку `Operations` и выбрать на ней тип операции `Compaction`.
- Если для той же таблицы (либо для одного из её вторичных индексов при `CASCADE = TRUE`) уже выполняется операция компакшена, запуск завершится с ошибкой. Нужно либо дождаться завершения предыдущей операции, либо отменить её.
- Для запуска команды требуются те же права, что и для остальных действий `ALTER TABLE`. Параллельные изменения схемы таблицы при этом не блокируются.
- Принудительный компакшен также материализует заимствованные данные. В результате общий объём данных в базе может увеличиться. Механизм заимствования используется, например, для реализации CoW копирования таблиц, поэтому при выполнении компакшена скопированных таблиц их объём в пределе может увеличиться на размер исходной таблицы.

Примеры

Запустить компакшен только основной таблицы:

```
ALTER TABLE series COMPACT;
```

Запустить компакшен таблицы вместе со всеми её вторичными индексами:

```
ALTER TABLE series COMPACT WITH (CASCADE = TRUE);
```

Запустить компакшен только impl-таблицы конкретного вторичного индекса:

```
ALTER TABLE `series/idx_release/indexImplTable` COMPACT;
```

Запустить компакшен таблицы, установив число одновременно обрабатываемых партиций:

```
ALTER TABLE series COMPACT WITH (PARALLEL = 5);
```

ALTER VIEW

`ALTER VIEW` изменяет определение представления.

> **Важно**
>
> Это команда не поддерживается в текущей версии YDB.

Для переопределения представления можно его удалить и воссоздать с другим определением:

```
DROP VIEW redefined_view;
CREATE VIEW redefined_view ...;
```

Обратите внимание, что эти две инструкции выполняются отдельно, в отличие от одной инструкции `ALTER VIEW`. При изменении представления таким образом его может быть видно в удалённом состоянии на короткий момент.

См. также

- CREATE VIEW
- DROP VIEW

ALTER RESOURCE POOL

`ALTER RESOURCE POOL` изменяет определение пула ресурсов.

Синтаксис

Изменение параметров

Синтаксис для изменения любого параметра пула ресурсов выглядит следующим образом:

```
ALTER RESOURCE POOL <name> SET (<key> = <value>);
```

`<key>` — имя параметра, `<value>` — его новое значение.

Например, такая команда включит ограничение на число параллельных запросов, равное 100:

```
ALTER RESOURCE POOL olap SET (CONCURRENT_QUERY_LIMIT = "100");
```

Сброс параметров

Команда для сброса параметра пула ресурсов выглядит следующим образом:

```
ALTER RESOURCE POOL <name> RESET (<key>);
```

`<key>` — имя параметра.

Например, такая команда сбросит настройки `TOTAL_CPU_LIMIT_PERCENT_PER_NODE` для пула ресурсов:

```
ALTER RESOURCE POOL olap RESET (TOTAL_CPU_LIMIT_PERCENT_PER_NODE);
```

Разрешения

Требуется разрешение `ALTER SCHEMA` на пул ресурсов в директории `.metadata/workload_manager/pools`, пример выдачи такого разрешения:

```
GRANT 'ALTER SCHEMA' ON `.metadata/workload_manager/pools/olap_pool` TO `user1@domain`;
```

Параметры

- `CONCURRENT_QUERY_LIMIT` (Int32) — опциональное поле, задающее количество параллельно выполняющихся запросов в пуле ресурсов. Если значение `-1`, то ограничений нет. Значение по умолчанию: `-1`. Допустимые значения: .
- `QUEUE_SIZE` (Int32) — опциональное поле, определяющее размер очереди ожидания. Всего в системе может находиться не более чем запросов одновременно. Если значение `-1`, ограничений нет. Значение по умолчанию: `-1`. Допустимые значения: .
- `DATABASE_LOAD_CPU_THRESHOLD` (Int32) — опциональное поле, задающее порог загрузки CPU всей базы данных, после которого запросы не отправляются на выполнение и остаются в очереди. Если значение `-1`, ограничений нет. Значение по умолчанию: `-1`. Допустимые значения: .
- `TOTAL_CPU_LIMIT_PERCENT_PER_NODE` (Double) — опциональное поле, задающее процент доступного CPU, который могут использовать все запросы на узле в данном пуле ресурсов. Если значение `-1`, ограничений нет. Значение по умолчанию: `-1`. Допустимые значения: .

См. также

- Workload Manager — управление потреблением ресурсов
- CREATE RESOURCE POOL
- DROP RESOURCE POOL

ALTER RESOURCE POOL CLASSIFIER

`ALTER RESOURCE POOL CLASSIFIER` изменяет определение классификатора пула ресурсов.

Синтаксис

Изменение параметров

Синтаксис для изменения любого параметра классификатора пула ресурсов выглядит следующим образом:

```
ALTER RESOURCE POOL CLASSIFIER <name> SET (<key> = <value>);
```

`<key>` — имя параметра, `<value>` — его новое значение.

Например, такая команда изменит пользователя, для которого применяется правило:

```
ALTER RESOURCE POOL CLASSIFIER olap_classifier SET (MEMBER_NAME = "user2@domain");
```

Сброс параметров

Команда для сброса параметра классификатора пула ресурсов выглядит следующим образом:

```
ALTER RESOURCE POOL CLASSIFIER <name> RESET (<key>);
```

`<key>` — имя параметра.

Например, такая команда сбросит настройку `MEMBER_NAME`:

```
ALTER RESOURCE POOL CLASSIFIER olap_classifier RESET (MEMBER_NAME);
```

Разрешения

Требуется разрешение `ALL` на базу данных, пример выдачи такого разрешения:

```
GRANT 'ALL' ON `/my_db` TO `user1@domain`;
```

Параметры

- `RANK` (Int64) — опциональное поле, задающее порядок выбора классификатора пула ресурсов. Если значение не указано, берётся максимальный существующий `RANK` и к нему прибавляется 1000. Допустимые значения: уникальное число в диапазоне .
- `RESOURCE_POOL` (String) — обязательное поле, задающее имя пула ресурсов, в который будут отправлены запросы, удовлетворяющие критериям классификатора.
- `MEMBER_NAME` (String) — опциональное поле, определяющее, какой пользователь или группа пользователей будут отправлены в указанный пул ресурсов. Формат значения описан в CREATE RESOURCE POOL CLASSIFIER.

См. также

- Workload Manager — управление потреблением ресурсов
- CREATE RESOURCE POOL CLASSIFIER
- DROP RESOURCE POOL CLASSIFIER

ALTER STREAMING QUERY

`ALTER STREAMING QUERY` изменяет настройки потоковых запросов, а также управляет их состоянием: запуском и остановкой.

Синтаксис

```
ALTER STREAMING QUERY [IF EXISTS] <query_name> SET (
    <key1> = <value1>,
    <key2> = <value2>,
    ...
)
```

Параметры

- `IF EXISTS` — не выводить ошибку, если потокового запроса не существует.
- `query_name` — имя потокового запроса, подлежащего изменению.
- `SET (<key> = <value>)` — список настроек потокового запроса, которые нужно обновить, опционально.

Изменение параметров запроса

Синтаксис:

```
ALTER STREAMING QUERY [IF EXISTS] <query_name> SET (<key> = <value>)
```

Доступные параметры:

- `RUN = (TRUE|FALSE)` — запустить или остановить запрос.
- `RESOURCE_POOL = <resource_pool_name>` — имя пула ресурсов, в котором будет выполняться запрос.

При выполнении `SET (RUN = TRUE)` смещения чтения из топика и состояния агрегационных функций восстанавливаются из чекпоинта. При отсутствии чекпоинта чтение начинается с самых свежих данных.

Примеры изменения параметров запроса см. ниже.

Разрешения

Для работы с потоковыми запросами требуется разрешение `ALTER SCHEMA`, пример выдачи такого разрешения для запроса `my_streaming_query`:

```
GRANT ALTER SCHEMA ON my_streaming_query TO `user@domain`
```

Примеры

Изменение параметров

Остановка запроса `my_streaming_query`:

```
ALTER STREAMING QUERY my_streaming_query SET (
    RUN = FALSE
)
```

Запуск запроса `my_streaming_query` в пуле ресурсов `my_resource_pool`:

```
ALTER STREAMING QUERY my_streaming_query SET (
    RUN = TRUE,
    RESOURCE_POOL = my_resource_pool
)
```

Статус запроса

Текущий статус запроса доступен в колонке `Status` системной таблицы .sys/streaming_queries:

```
SELECT
    Path,
    Status,
    Text,
    Run
FROM
    `.sys/streaming_queries`
```

Возможные значения статуса:

1. `CREATING` — запрос создаётся после выполнения команды `CREATE STREAMING QUERY`.
2. `CREATED` — запрос создан, но не запущен (например, при указании `RUN = FALSE`).
3. `STARTING` — запрос запускается.
4. `RUNNING` — запрос выполняется.
5. `SUSPENDED` — запрос приостановлен из-за внутренних ошибок. Система автоматически повторит запуск.
6. `STOPPING` — запрос останавливается по команде `ALTER STREAMING QUERY ... SET (RUN = FALSE)`.
7. `STOPPED` — запрос остановлен.

Гарантируется, что на момент успешного завершения DDL для создания или изменения потокового запроса, статус будет `CREATED`, `STARTING`, `RUNNING`, `STOPPED` или `SUSPENDED` в зависимости от настройки `RUN = (TRUE|FALSE)` и успешности запуска запроса.

Примеры обработки данных в других форматах приведены в статье Форматы данных при чтении/записи из топиков. Подробнее о возможностях и ограничениях потоковых запросов см. в документации.

См. также

- Потоковые запросы
- CREATE STREAMING QUERY
- DROP STREAMING QUERY

ALTER TOPIC

С помощью оператора `ALTER TOPIC` можно изменить настройки топика, а также добавить, изменить или удалить читателя.

Общий вид команды:

```
ALTER TOPIC topic_path action1, action2, ..., actionN;
```

- `action` — действие по изменению. Возможные действия описаны ниже.

Работа с топиком

Задать параметры топика

`SET (option = value[, ...])` — действие задает параметры топика.

Общий вид команды:

```
ALTER TOPIC topic_path SET (option = value[, ...]);
```

- `option` и `value` — параметр топика и его значение.

Параметры топика

- `metering_mode` — способ метеринга ресурсов (`RESERVED_CAPACITY` - по выделенным ресурсам или `REQUEST_UNITS` - по фактическому использованию). Актуально для топиков в serverless базах данных. Тип значения - `String`.
- `min_active_partitions` — минимальное количество активных партиций топика. Автопартиционирование не будет уменьшать количество активных партиций ниже этого значения. Тип — `integer`, значение по умолчанию — `1`.
- `max_active_partitions` — максимальное количество активных партиций топика. Автопартиционирование не будет увеличивать количество активных партиций выше этого значения. Тип — `integer`, по умолчанию равно `min_active_partitions`.
- `retention_period` — время хранения данных в топике. Тип значения — `Interval`.
- `retention_storage_mb` — ограничение на максимальное место на диске, занимаемое данными топика. При превышении этого значения старые данные будут удаляться, как по retention. При включенном автоматическом партиционировании потребляемое место может превышать установленное значение. Тип значения — `integer`, значение по умолчанию — `0` (не ограничено).
- `partition_write_burst_bytes` — размер запаса квоты на запись в партицию на случай всплесков записи. При выставлении в `0` фактическое значение write_burst принимается равным значению квоты (что позволяет всплески записи длительностью до 1 секунды). Тип значения — `integer`, значение по умолчанию: `0`.
- `partition_write_speed_bytes_per_second` — максимальная разрешенная скорость записи в 1 партицию. Если поток записи в партицию превысит это значение, запись будет квотироваться. Тип значения — `integer`, значение по умолчанию — `2097152` (2 МБ).
- `auto_partitioning_strategy` — режим автопартиционирования.
  Допустимые значения: `paused`, `scale_up`, значение по умолчанию — `disabled`.
- `auto_partitioning_up_utilization_percent` — определяет порог загрузки партиции в процентах от максимальной скорости записи, при достижении которого будет инициировано автоматическое **увеличение** числа партиций. Тип значения — `integer`, значение по умолчанию — `80`.
- `auto_partitioning_stabilization_window` — определяет временной интервал, в течение которого уровень нагрузки должен оставаться выше установленного порога (`auto_partitioning_up_utilization_percent`), прежде чем будет выполнено автоматическое увеличение количества партиций. Тип значения — `Interval`, значение по умолчанию — `5m`.
- `supported_codecs` — список кодеков, поддерживаемых топиком. Тип значения — `String`.

Следующая команда изменит время хранения данных в топике и квоту на скорость записи в 1 партицию:

```
ALTER TOPIC `my_topic` SET (
    retention_period = Interval('PT36H'),
    partition_write_speed_bytes_per_second = 3000000
);
```

Включение и приостановка автопартиционирования

Следующая команда включает автопартиционирование:

```
ALTER TOPIC `my_topic` SET (
    min_active_partitions = 1,
    max_active_partitions = 5,
    auto_partitioning_strategy = 'scale_up'
);
```

Следующая команда ставит автопартиционирование на паузу:

```
ALTER TOPIC `my_topic` SET (
    auto_partitioning_strategy = 'paused'
);
```

Следующая команда снимает автопартиционирование с паузы:

```
ALTER TOPIC `my_topic` SET (
    auto_partitioning_strategy = 'scale_up'
```

```
);
```

Работа с читателем

Добавить читателя

`ADD CONSUMER` — действие добавляет читателей для топика.

Общий вид команды:

```
ALTER TOPIC topic_path ADD CONSUMER consumer_name [WITH (option = value[, ...])];
```

- `option` и `value` — параметр читателя и его значение.

Параметры читателя:

- `type` — тип читателя. Возможные значения: `STREAMING` и `SHARED`. Значение по умолчанию `STREAMING`.
- `important` — признак важного читателя. Данные из топика не будут удалены, пока их не обработают все важные читатели. Тип значения — `boolean`, значение по умолчанию — `false`.
- `availability_period` — определяет время доступности сообщений для читателя. Опция позволяет продлить время хранения сообщений в топике сверх retention_period, вплоть до `availability_period`, если читатель не подтверждает их обработку. Тип значения — `Interval`. Несовместим с параметром `important`. Значение по умолчанию отсутствует.
- `read_from` — определяет момент записи сообщений, начиная с которого читатель будет получать данные. Сообщения, записанные ранее этого момента, читатель не получит. Тип значения — `Datetime`, `Timestamp` или `integer` (Unix timestamp в виде числа). Значение по умолчанию — `0` (чтение с самого раннего доступного в топике момента времени).
- `supported_codecs` — список кодеков, поддерживаемых читателем.

Параметры читателя, доступные только для разделяемого (общего) читателя:

- `keep_messages_order` — сохраняет порядок чтения сообщений. Если значение равно `true`, гарантируется порядок обработки сообщений в рамках одной группы сообщений. Значение по умолчанию `false`.
- `default_processing_timeout` — время обработки сообщения. Если за это время обработка сообщения не будет подтверждена и время обработки не будет продлено, сообщение вернется в очередь и будет передано на повторную обработку. Значение по умолчанию `Interval('PT30S')`.
- `max_processing_attempts` — максимальное количество попыток обработки одного сообщения (тип значения — `integer`). Опция поддерживается только при `dead_letter_policy = move` или `dead_letter_policy = delete`. Значение по умолчанию — `1000`.
- `dead_letter_policy` — действие, которое нужно выполнить с сообщением, если все попытки его обработки завершились неудачей (тип значения — `String`). Возможные значения: `delete`, `move`, `none`. Значение по умолчанию — `none`.
- `dead_letter_queue` — имя топика DLQ (тип значения — `String`). Обязателен при `dead_letter_policy = move` и не поддерживается при `dead_letter_policy = none` / `delete`.

Следующая команда добавит к топику читателя с настройками по умолчанию:

```
ALTER TOPIC `my_topic` ADD CONSUMER my_consumer;
```

Следующая команда добавит к топику важного читателя:

```
ALTER TOPIC `my_topic` ADD CONSUMER my_consumer2 WITH (important = true);
```

Следующая команда добавит к топику разделяемого (общего) читателя:

```
ALTER TOPIC `my_topic`
    ADD CONSUMER my_shared_consumer WITH (
        type = 'shared',
        keep_messages_order = false,
        default_processing_timeout = Interval('PT30S'),
        max_processing_attempts = 3,
        dead_letter_policy = 'move',
        dead_letter_queue = 'my_dlq_topic'
    );
```

Задать параметры читателя

`ALTER CONSUMER consumer_name SET (option = value[, ...])` — действие задает параметры читателя топика.

Общий вид команды:

```
ALTER TOPIC topic_path ALTER CONSUMER consumer_name SET (option = value[, ...]);
```

- `option` и `value` — параметр читателя и его значение.

Следующая команда сделает читателя важным:

```
ALTER TOPIC `my_topic` ALTER CONSUMER my_consumer SET (important = true);
```

В одной команде может быть указано несколько `ALTER CONSUMER` действий, настройки в них не должны повторяться:

```
ALTER TOPIC `my_topic`
    ALTER CONSUMER my_consumer SET (availability_period = Interval('PT48H'))
```

```
    ALTER CONSUMER my_consumer SET (read_from = 0);
```

Удалить читателя

`DROP CONSUMER` — действие удаляет читателя топика.

Общий вид команды:

```
ALTER TOPIC topic_path DROP CONSUMER consumer_name;
```

Следующая команда удалит читателя с именем `old_consumer`:

```
ALTER TOPIC `my_topic` DROP CONSUMER old_consumer;
```

ALTER TRANSFER

Вызов `ALTER TRANSFER` изменяет параметры и состояние экземпляра трансфера.

Синтаксис

```
ALTER TRANSFER <name> [SET USING lambda | SET (option = value [, ...])]
```

где:

- `name` — имя экземпляра трансфера.
- `lambda` — lambda-функция преобразования сообщений.
- `SET (option = value [, ...])` — параметры трансфера.

Параметры

- `STATE` — состояние трансфера. Возможные значения:
  - `PAUSED` — остановка трансфера.
  - `ACTIVE` — возобновление работы трансфера после приостановки.
- Параметры батчевания записи в таблицу позволяют настроить баланс между задержкой появления записей в таблице и ресурсами, требуемыми для работы трансфера. Параметры батчевания влияют на обработку каждой партиции топика независимо. К изменению параметров батчевания нужно подходить с осторожностью, так как их изменение может как улучшить скорость обработки потока сообщений, так и ухудшить её, и даже привести к отказу в обслуживании, если параметры будут подобраны неверно. Например, запись в таблицу маленькими по размеру батчами может привести к перегрузке таблицы и деградации скорости работы с ней, а слишком большой размер батча — к тому, что на сервере закончится вся доступная память.
  - `BATCH_SIZE_BYTES` — размер батча в байтах. По умолчанию — 8 МБ.
  - `FLUSH_INTERVAL` — периодичность записи в таблицу. По умолчанию — 60 секунд. Запись в таблицу будет осуществлена, даже если батч не достиг размера, заданного в параметре `BATCH_SIZE_BYTES`.
- Настройки для аутентификации в базе топика одним из способов:
  - С помощью токена:
    - `TOKEN_SECRET_PATH` — секрет, содержащий токен.
  - С помощью логина и пароля:
    - `USER` — имя пользователя.
    - `PASSWORD_SECRET_PATH` — секрет, содержащий пароль.
  - С помощью сервисного агента:
    - `SERVICE_ACCOUNT_ID` — идентификатор сервисного аккаунта.
    - `INITIAL_TOKEN_SECRET_PATH` — секрет, содержащий токен от сервисного аккаунта. Используется для первоначальной инициализации.

Разрешения

Для изменения трансфера требуется право изменять схемные объекты (`ALTER SCHEMA`).

Примеры

Следующий запрос изменяет lambda-функцию преобразования сообщений топика:

```
$new_lambda = ($msg) -> {
    return [
        <|
            partition: $msg._partition,
            offset: $msg._offset,
            message: CAST($msg._data || ' altered' AS Utf8)
        |>
    ];
};

ALTER TRANSFER my_transfer SET USING $new_lambda;
```

Следующий запрос временно приостанавливает работу трансфера:

```
ALTER TRANSFER my_transfer SET (STATE = "PAUSED");
```

Следующий запрос изменяет параметры батчевания:

```
ALTER TRANSFER my_transfer SET (
    BATCH_SIZE_BYTES = 1048576,
    FLUSH_INTERVAL = Interval('PT60S')
);
```

Следующий запрос изменяет секрет:

```
ALTER TRANSFER my_transfer SET (
    TOKEN_SECRET_PATH = "my_token"
);
```

lambda-функция

Lambda-функция преобразования сообщений принимает один параметр со структурой, содержащей сообщение из топика, и возвращает список структур, соответствующих строкам таблицы для вставки.

Пример:

```
$lambda = ($msg) -> {
  return [
    <|
      column_1: $msg._create_timestamp,
      column_2: $msg._data
    |>
  ];
};
```

В этом примере:

- `$msg` — сообщение, полученное из топика.
- `column_1` и `column_2` — названия колонок таблицы.
- `$msg._create_timestamp` и `$msg._data` — значения, которые будут записаны в таблицу. Типы значений должны совпадать с типами колонок таблицы. Например, если `column_2` имеет в таблице тип `String`, то и тип `$msg._data` должен быть именно `String`.

У сообщения топика доступны следующие поля:

Атрибут	Тип значения	Описание
`_create_timestamp`	`Timestamp`	Время создания сообщения
`_data`	`String`	Тело сообщения
`_offset`	`Uint64`	Смещение сообщения
`_partition`	`Uint32`	Номер партиции сообщения
`_producer_id`	`String`	Идентификатор писателя
`_seq_no`	`Uint64`	Порядковый номер сообщения
`_write_timestamp`	`Timestamp`	Время записи сообщения

Тестирование lambda-функций

Для тестирования lambda-функции при её разработке можно в качестве сообщения топика передавать структуру с такими же полями, как будут передаваться в трансфере. Пример:

```
$lambda = ($msg) -> {
  return [
    <|
      offset: $msg._offset,
      data: $msg._data
    |>
  ];
};

$msg = <|
  _data: "value",
  _offset: CAST(1 AS Uint64),
  _partition: CAST(2 AS Uint32),
  _producer_id: "producer",
  _seq_no: CAST(3 AS Uint64)
|>;

SELECT $lambda($msg);
```

Если lambda-функция содержит сложную логику преобразования, то её можно выделить в отдельную lambda-функцию, что упростит тестирование.

```
$extract_value = ($data) -> {
  -- сложные преобразования
  return $data;
};

$lambda = ($msg) -> {
  return [
    <|
      column: $extract_value($msg._data)
    |>
  ];
};

-- Тестировать lambda-функцию extract_value можно так

SELECT $extract_value('преобразуемое значение');
```

См. также

- CREATE TRANSFER
- DROP TRANSFER
- Трансфер данных

ALTER SECRET

Команда `ALTER SECRET` изменяет существующий секрет.

Синтаксис:

```
ALTER SECRET [IF EXISTS] secret_name
WITH (option = value[, ...])
```

- `IF EXISTS` — команда не возвращает ошибку, если секрет не существует; в этом случае она ничего не делает.
- `secret_name` — имя изменяемого секрета.
- `option` — опция команды:
  - `value` — строка со значением секрета.

Разрешения

Для изменения секрета требуется право `ALTER SCHEMA`.

Примеры

Изменить значение секрета `secret_name` на `secret_value_new`:

```
ALTER SECRET secret_name WITH (value = "secret_value_new");
```

Изменить значение секрета `secret_name` на `secret_value_new`, только если он существует; если он не существует, команда ничего не делает:

```
ALTER SECRET IF EXISTS secret_name WITH (value = "secret_value_new");
```

См. также

- CREATE SECRET
- DROP SECRET

ALTER USER

Изменяет пароль пользователя.

Синтаксис:

```
ALTER USER user_name [ WITH ] option [ ... ]
```

- `user_name` — имя пользователя.
- `option` — опция команды:
  - `PASSWORD 'password'` — изменяет пароль на `password`.
  - `PASSWORD NULL` — устанавливает пустой пароль.
  - `NOLOGIN` - запрет на логин пользователя (блокировка пользователя).
  - `LOGIN` - разрешение на логин пользователя (разблокировка пользователя).

Встроенные пользователи

YDB может иметь набор групп и пользователей уже с момента начального развёртывания.

Подробнее см. Первичные настройки безопасности и Начальная настройка безопасности кластера.

Такие пользователи и группы ничем не отличаются от пользователей и групп, созданных позже.

ANALYZE

`ANALYZE` принудительно запускает сбор статистики для стоимостного оптимизатора YDB.

Синтаксис

```
ANALYZE <path_to_table> [ (<column_name> [, ...]) ]
```

Эта команда принудительно запускает синхронный сбор статистики таблицы и статистики столбцов для указанных столбцов или для всех столбцов, если столбцы не указаны. `ANALYZE` возвращает результат после сбора и обновления всей запрашиваемой статистики.

- `path_to_table` — путь к таблице, для которой должна быть собрана статистика.
- `column_name` — сбор статистики столбцов только для указанных столбцов таблицы.

Текущий набор статистики описан в Статистики для стоимостного оптимизатора.

Фоновая операция

SQL операция `ANALYZE` выполняется синхронно и ожидает завершения. При этом создаётся фоновая операция для наблюдаемости. Используйте команду operation list для отслеживания прогресса фоновых операций `ANALYZE`.

BATCH DELETE FROM

> **Совет**
>
> Перед тем как изучать `BATCH DELETE FROM`, рекомендуется ознакомиться со стандартным DELETE FROM.

`BATCH DELETE FROM` позволяет удалять записи в таблицах большого размера, минимизируя риск отмены блокировок и отката транзакций за счёт ослабления гарантий. Удаление данных выполняется в виде серии транзакций для каждой партиции указанной таблицы отдельно, обрабатывая по 10 000 строк за итерацию. Каждый запрос обрабатывает не более 10 партиций одновременно.

Данный запрос, как и стандартный `DELETE FROM`, выполняется синхронно и завершается с некоторым статусом. В случае возникновения ошибки или отключения клиента удаление данных останавливается, применённые изменения не откатываются.

Семантика наследуется от стандартного `DELETE FROM` с ограничениями:

- Поддерживается только для строковых таблиц.
- Поддерживается только в режиме неявного контроля транзакций.
- Запрещено использование подзапросов и нескольких выражений в одном запросе.
- Недоступно ключевое слово `RETURNING`.

Пример

```
BATCH DELETE FROM my_table
WHERE Key1 > 1 AND Key2 >= "One";
```

BATCH UPDATE

> **Совет**
>
> Перед тем как изучать `BATCH UPDATE`, рекомендуется ознакомиться со стандартным UPDATE.

`BATCH UPDATE` позволяет обновлять записи в таблицах большого размера, минимизируя риск отмены блокировок и отката транзакций за счёт ослабления гарантий. Обновление данных выполняется в виде серии транзакций для каждой партиции указанной таблицы отдельно, обрабатывая по 10 000 строк за итерацию. Каждый запрос обрабатывает не более 10 партиций одновременно.

Данный запрос, как и стандартный `UPDATE`, выполняется синхронно и завершается с некоторым статусом. В случае возникновения ошибки или отключения клиента обновление данных останавливается, применённые изменения не откатываются.

Семантика наследуется от стандартного `UPDATE` с ограничениями:

- Поддерживается только для строковых таблиц.
- Поддерживается только в режиме неявного контроля транзакций.
- Поддерживаются только идемпотентные обновления: выражения после `SET` не должны зависеть от текущих значений изменяемых колонок.
- Запрещено использование подзапросов и нескольких выражений в одном запросе.
- Недоступно ключевое слово `RETURNING`.

Пример

```
BATCH UPDATE my_table
SET Value1 = "foo", Value2 = 0
WHERE Key1 > 1;
```

BACKUP

Выражение `BACKUP` создает резервную копию таблиц в коллекции резервных копий.

```
BACKUP collection_name [INCREMENTAL];
```

Параметры

- `collection_name` — имя коллекции резервных копий.
- `INCREMENTAL` — создание инкрементальной резервной копии вместо полной.

Типы резервных копий

Полная резервная копия

Полная резервная копия создает снимок всех таблиц в коллекции на определенный момент времени. Это служит основой для последующих инкрементальных резервных копий.

```
BACKUP production_backups;
```

Инкрементальная резервная копия

Инкрементальная резервная копия захватывает только изменения (вставки, обновления, удаления) с момента предыдущей резервной копии в цепочке. Коллекция должна быть создана с `INCREMENTAL_BACKUP_ENABLED = 'true'`.

```
BACKUP production_backups INCREMENTAL;
```

Важно

Инкрементальные резервные копии требуют предыдущей полной резервной копии в той же коллекции. Всегда сначала создавайте полную резервную копию, прежде чем делать инкрементальные резервные копии.

Примеры

Создание начальной полной резервной копии:

```
-- Сначала создайте коллекцию
CREATE BACKUP COLLECTION daily_backups (
    TABLE orders
) WITH (
    STORAGE = 'cluster',
    INCREMENTAL_BACKUP_ENABLED = 'true'
);

-- Затем создайте полную резервную копию
BACKUP daily_backups;
```

Создание инкрементальных резервных копий:

```
-- После начальной полной резервной копии создайте инкрементальные резервные копии
BACKUP daily_backups INCREMENTAL;
```

Мониторинг операций резервного копирования

Операции резервного копирования выполняются асинхронно в фоновом режиме. Вы можете отслеживать их прогресс с помощью YDB CLI:

```
# Список операций резервного копирования
ydb operation list incbackup

# Получить детали операции
ydb operation get <operation-id>
```

См. также

- Коллекции резервных копий.
- CREATE BACKUP COLLECTION.
- RESTORE.
- DROP BACKUP COLLECTION.
- Резервное копирование и восстановление.

CREATE ASYNC REPLICATION

Вызов `CREATE ASYNC REPLICATION` создает экземпляр асинхронной репликации.

Синтаксис

```
CREATE ASYNC REPLICATION <name>
FOR <remote_path> AS <local_path> [, <another_remote_path> AS <another_local_path>]
WITH (option = value [, ...])
```

где:

- `name` — имя экземпляра асинхронной репликации.
- `remote_path` — относительный или абсолютный путь до исходной таблицы или директории в базе-источнике.
- `local_path` — относительный или абсолютный путь до целевой таблицы или директории в текущей базе.
- `WITH (option = value [, ...])` — параметры асинхронной репликации.

Параметры

- `CONNECTION_STRING` — строка соединения с базой-источником. Обязательный параметр.
- `CA_CERT` — корневой сертификат для TLS. Необязательный параметр. Может быть указан, если база-источник поддерживает режим обмена данными с шифрованием (`CONNECTION_STRING` начинается с `grpcs://`).
- `INITIAL_SCAN`: выполнять ли первоначальное сканирование. Необязательный параметр типа `Bool`. Если не указан, используется настройка сервера; по умолчанию `TRUE`.
  - `TRUE`: исходные данные передаются через потоки изменений, объекты-реплики создаются автоматически.
  - `FALSE`: репликация создаётся в состоянии `PAUSED` и подготавливает потоки изменений без первоначального сканирования. Целевые таблицы не создаются и не подключаются к репликации до перевода в `STANDBY`. Сначала создайте резервную копию источника и восстановите данные в приёмнике, затем запустите репликацию с помощью ALTER ASYNC REPLICATION. К моменту запуска целевые таблицы должны существовать и соответствовать схемам источника с учётом ограничений. Порядок действий описан в руководстве по первоначальной синхронизации.
- `AVAILABILITY_PERIOD` — период доступности неподтверждённых изменений. Необязательный параметр: положительное значение типа `Interval` или строка `'UNLIMITED'`. По умолчанию — `'UNLIMITED'`.
  - `'UNLIMITED'`: изменения сохраняются до подтверждения их обработки репликацией.
  - `Interval`, например `Interval('PT48H')`: неподтверждённые изменения защищены от удаления в течение заданного периода. После его истечения они могут быть удалены.

  Параметр задаётся только при создании репликации. Выбрать значение поможет сравнение вариантов хранения.
- Настройки для аутентификации в базе-источнике одним из способов (обязательно):
  - С помощью токена:
    - `TOKEN_SECRET_PATH` — секрет, содержащий токен.
  - С помощью логина и пароля:
    - `USER` — имя пользователя.
    - `PASSWORD_SECRET_PATH` — секрет, содержащий пароль.
  - С помощью сервисного агента:
    - `SERVICE_ACCOUNT_ID` — идентификатор сервисного аккаунта.
    - `INITIAL_TOKEN_SECRET_PATH` — секрет, содержащий токен от сервисного аккаунта. Используется для первоначальной инициализации.
- `CONSISTENCY_LEVEL` — уровень согласованности реплицируемых данных:
  - `ROW` — согласованность данных уровня строки. Режим по умолчанию.
  - `GLOBAL` — глобальная согласованность данных. Дополнительно можно указать:
    - `COMMIT_INTERVAL` — периодичность фиксации изменений в формате ISO 8601. Значение по умолчанию — 10 секунд.

Примеры

> **Совет**
>
> Перед выполнением операции создайте секрет с аутентификационными данными для подключения или убедитесь в его существовании и наличии доступа к нему.

Создание экземпляра асинхронной репликации для таблицы `original_table` из базы `/Root/another_database` в текущую базу в таблицу `replica_table`:

```
CREATE ASYNC REPLICATION my_replication_for_single_table
FOR original_table AS replica_table
WITH (
    CONNECTION_STRING = 'grpcs://example.com:2135/?database=/Root/another_database',
    TOKEN_SECRET_PATH = 'my_secret'
);
```

Для подключения к базе `/Root/another_database` используется эндпоинт `grpcs://example.com:2135`, а для аутентификации — токен из секрета `my_secret`.

Создание экземпляра асинхронной репликации для таблиц `original_table_1` и `original_table_2` в `replica_table_1` и `replica_table_2`, соответственно:

```
CREATE ASYNC REPLICATION my_replication_for_multiple_tables
FOR original_table_1 AS replica_table_1, original_table_2 AS replica_table_2
WITH (
```

```
    CONNECTION_STRING = 'grpcs://example.com:2135/?database=/Root/another_database',
    TOKEN_SECRET_PATH = 'my_secret'
);
```

Создание экземпляра асинхронной репликации для содержимого директории `original_dir`:

```
CREATE ASYNC REPLICATION my_replication_for_dir
FOR original_dir AS replica_dir
WITH (
    CONNECTION_STRING = 'grpcs://example.com:2135/?database=/Root/another_database',
    TOKEN_SECRET_PATH = 'my_secret'
);
```

Создание экземпляра асинхронной репликации для содержимого базы `/Root/another_database`:

```
CREATE ASYNC REPLICATION my_replication_for_database
FOR `/Root/another_database` AS `/Root/my_database`
WITH (
    CONNECTION_STRING = 'grpcs://example.com:2135/?database=/Root/another_database',
    TOKEN_SECRET_PATH = 'my_secret'
);
```

Создание экземпляра асинхронной репликации с указанием корневого сертификата для TLS:

```
CREATE ASYNC REPLICATION my_consistent_replication
FOR original_table AS replica_table
WITH (
    CONNECTION_STRING = 'grpcs://example.com:2135/?database=/Root/another_database',
    TOKEN_SECRET_PATH = 'my_secret',
    CA_CERT = '-----BEGIN CERTIFICATE-----...'
);
```

Создание экземпляра асинхронной репликации в режиме глобальной согласованности данных (периодичность фиксации изменений по умолчанию — 10 секунд):

```
CREATE ASYNC REPLICATION my_consistent_replication
FOR original_table AS replica_table
WITH (
    CONNECTION_STRING = 'grpcs://example.com:2135/?database=/Root/another_database',
    TOKEN_SECRET_PATH = 'my_secret',
    CONSISTENCY_LEVEL = 'GLOBAL'
);
```

Создание экземпляра асинхронной репликации в режиме глобальной согласованности данных с минутной периодичностью фиксации изменений:

```
CREATE ASYNC REPLICATION my_consistent_replication_1min_commit_interval
FOR original_table AS replica_table
WITH (
    CONNECTION_STRING = 'grpcs://example.com:2135/?database=/Root/another_database',
    TOKEN_SECRET_PATH = 'my_secret',
    CONSISTENCY_LEVEL = 'GLOBAL',
    COMMIT_INTERVAL = Interval('PT1M')
);
```

Создание репликации с периодом доступности неподтверждённых изменений 48 часов:

```
CREATE ASYNC REPLICATION my_replication_with_availability_period
FOR original_table AS replica_table
WITH (
    CONNECTION_STRING = 'grpcs://example.com:2135/?database=/Root/another_database',
    TOKEN_SECRET_PATH = 'my_secret',
    AVAILABILITY_PERIOD = Interval('PT48H')
);
```

Создание репликации для первоначальной синхронизации с помощью резервного копирования и восстановления с сохранением неподтверждённых изменений без ограничения по возрасту:

```
CREATE ASYNC REPLICATION my_replication_without_initial_scan
FOR original_table AS replica_table
WITH (
    CONNECTION_STRING = 'grpcs://example.com:2135/?database=/Root/another_database',
    TOKEN_SECRET_PATH = 'my_secret',
    INITIAL_SCAN = FALSE,
    AVAILABILITY_PERIOD = 'UNLIMITED'
);
```

Репликация подготавливает потоки изменений в состоянии `PAUSED`. Дождитесь их готовности, выполните резервное копирование и восстановление, затем переведите репликацию в `STANDBY` с помощью ALTER ASYNC REPLICATION.

См. также

- Первоначальная синхронизация с помощью резервного копирования и восстановления
- ALTER ASYNC REPLICATION

- DROP ASYNC REPLICATION

CREATE BACKUP COLLECTION

Выражение `CREATE BACKUP COLLECTION` создает коллекцию резервных копий.

```
CREATE BACKUP COLLECTION collection_name (
    TABLE table_name [, TABLE another_table_name ...]
) WITH (option = value [, ...]);
```

Параметры

- `collection_name` — имя создаваемой коллекции резервных копий.
- `table_name` — полный путь к таблице для включения в коллекцию. Можно указать несколько таблиц.
- Опции:
  - `STORAGE` — тип хранилища для резервных копий. Поддерживаемые варианты:
    - `'cluster'` — хранение в кластере YDB.
  - `INCREMENTAL_BACKUP_ENABLED` — включает или отключает поддержку инкрементальных резервных копий. Установите в `'true'` для включения инкрементальных резервных копий, `'false'` — только для полных резервных копий.

> **Примечание**
>
> При выборе имени коллекции резервных копий учитывайте общие правила именования схемных объектов.

Примеры

Создание коллекции резервных копий с одной таблицей:

```
CREATE BACKUP COLLECTION daily_backups (
    TABLE orders
) WITH (
    STORAGE = 'cluster',
    INCREMENTAL_BACKUP_ENABLED = 'true'
);
```

Создание коллекции резервных копий с несколькими таблицами:

```
CREATE BACKUP COLLECTION production_backups (
    TABLE orders,
    TABLE products,
    TABLE customers
) WITH (
    STORAGE = 'cluster',
    INCREMENTAL_BACKUP_ENABLED = 'true'
);
```

См. также

- Коллекции резервных копий.
- BACKUP.
- RESTORE.
- DROP BACKUP COLLECTION.

CREATE GROUP

Команда `CREATE GROUP` создаёт группу с указанным именем. Есть возможность указать список пользователей, входящих в эту группу.

Синтаксис:

```
CREATE GROUP group_name [ WITH USER user_name [ , user_name [ ... ]] [ , ] ]
```

- `group_name` — имя группы. Может содержать только строчные буквы латинского алфавита (a–z), цифры (0–9). Имя не должно быть пустым.
- `user_name` — имя пользователя, который станет участником группы после её создания. Может содержать строчные буквы латинского алфавита и цифры.

Примеры

Успешно выполненные команды

Создание группы:

```
CREATE GROUP group1;
```

```
CREATE GROUP group2 WITH USER root;
```

```
CREATE GROUP group3 WITH USER root, root1;
```

```
CREATE GROUP group4 WITH USER root, root1, root2;
```

Обработка ошибок

Корректно обработанные ошибки:

Ситуация	Сообщение об ошибке
Повторное создание группы	Group already exists
Невалидное имя (например, `group1№`)	token recognition error at: '№'

Встроенные группы

YDB может иметь набор групп и пользователей уже с момента начального развёртывания.

Подробнее см. Первичные настройки безопасности и Начальная настройка безопасности кластера.

Такие пользователи и группы ничем не отличаются от пользователей и групп, созданных позже.

См. также

- ALTER GROUP
- DROP GROUP

CREATE EXTERNAL DATA SOURCE

Вызов `CREATE EXTERNAL DATA SOURCE` создает внешний источник данных.

```
CREATE [OR REPLACE] EXTERNAL DATA SOURCE [IF NOT EXISTS] external_data_source WITH (
  SOURCE_TYPE="source_type",
  LOCATION="ip_address_or_fqdn:port",
  USE_TLS="use_tls",
  AUTH_METHOD="auth_method",
  LOGIN="login",
  PASSWORD_SECRET_PATH="password_secret_path"
)
```

Где:

- `OR REPLACE` - если внешний источник данных с таким именем уже существует, он будет заменён новым определением; версия объекта при этом увеличивается.
- `IF NOT EXISTS` - не выводить ошибку, если внешний источник данных с таким именем уже существует; существующий объект останется без изменений.
- `external_data_source` - название внешнего источника данных.
- `source_type` - тип внешнего источника данных. Возможные значения: `ClickHouse`, `PostgreSQL`, `ObjectStorage`.
- `ip_address_or_fqdn:port` - полный сетевой адрес внешнего источника данных, включая порт. В качестве сетевого адреса можно указывать IP-адрес или FQDN.
- `use_tls` - флаг, указывающий требование подключения через безопасное соединение (TLS). Возможные значения: `TRUE`, `FALSE`.
- `auth_method` - способ аутентификации во внешнем источника данных. Для внешних источников типов `ClickHouse`, `PostgreSQL` поддерживается только тип аутентификации `BASIC`. Для внешнего источника `ObjectStorage` в данный момент поддерживает только тип аутентификации `NONE`.
- `login` - логин, используемый для подключения к внешнему источнику данных.
- `password_secret_path` - секрет, содержащий пароль для подключения к внешнему источнику данных.

При работе по защищенным TLS каналам связи используется системные сертификаты, расположенные на серверах YDB.

Пример

Запрос ниже создает внешний источник с именем `TestDataSource` к кластеру ClickHouse с IP-адресом `192.168.1.1` и портом `8443`, логином `admin` и секретом `test_secret`:

```
CREATE EXTERNAL DATA SOURCE TestDataSource WITH (
  SOURCE_TYPE="ClickHouse",
  LOCATION="192.168.1.1:8443",
  USE_TLS="TRUE",
  AUTH_METHOD="BASIC",
  LOGIN="admin",
  PASSWORD_SECRET_PATH="test_secret"
)
```

Подключение к ClickHouse

Для создания подключения к кластеру ClickHouse необходимо создать внешний источник данных `EXTERNAL DATA SOURCE`, указав:

- В поле `SOURCE_TYPE` значение `ClickHouse`.
- В поле `LOCATION` полный сетевой адрес кластера ClickHouse, включая порт. В качестве сетевого адреса можно указывать IP-адрес или FQDN. В данный момент подключение к кластеру ClickHouse всегда выполняется про HTTP протоколу.
- В поле `USE_TLS` флаг, указывающий требование подключения к кластеру ClickHouse через безопасное соединение (TLS).
- В поле `AUTH_METHOD` значение `BASIC`.
- В поле `LOGIN` логин, используемый для подключения к кластеру ClickHouse.
- В поле `PASSWORD_SECRET_PATH` секрет, содержащий пароль для подключения к кластеру ClickHouse.

При работе по защищенным TLS каналам связи используется системные сертификаты, расположенные на серверах YDB.

Пример

Запрос ниже создает внешний источник с именем `TestDataSource` к кластеру ClickHouse с IP-адресом `192.168.1.1` и портом `8443`, логином `admin` и секретом `test_secret`:

```
CREATE EXTERNAL DATA SOURCE TestDataSource WITH (
  SOURCE_TYPE="ClickHouse",
  LOCATION="192.168.1.1:8443",
  USE_TLS="TRUE",
  AUTH_METHOD="BASIC",
  LOGIN="admin",
  PASSWORD_SECRET_PATH="test_secret"
)
```

Подключение к PostgreSQL

Для создания подключения к кластеру PostgreSQL необходимо создать объект `EXTERNAL DATA SOURCE`, указав в полях:

- в поле `SOURCE_TYPE` значение `PostgreSQL`;

- в поле `LOCATION` полный сетевой адрес кластера PostgreSQL, включая порт. В качестве сетевого адреса можно указывать IP-адрес или FQDN;
- в поле `USE_TLS` флаг, указывающий требование подключения к кластеру PostgreSQL через безопасное соединение (TLS);
- в поле `AUTH_METHOD` значение `BASIC`;
- в поле `LOGIN` логин, используемый для подключения к кластеру PostgreSQL;
- в поле `PASSWORD_SECRET_PATH` секрет, содержащий пароль для подключения к кластеру PostgreSQL.

В данный момент подключение к кластеру PostgreSQL всегда выполняется про стандартному (Frontend/Backend Protocol) по транспорту TCP. При работе по защищенным TLS каналам связи используется системные сертификаты, расположенные на серверах YDB.

Пример

Запрос ниже создает внешний источник с именем `TestDataSource`, ведущий на кластер PostgreSQL с IP-адресом `192.168.1.1` и портом `5432`, логином `admin` и секретом `test_secret`:

```
CREATE EXTERNAL DATA SOURCE TestDataSource WITH (
  SOURCE_TYPE="PostgreSQL",
  LOCATION="192.168.1.1:5432",
  USE_TLS="TRUE",
  AUTH_METHOD="BASIC",
  LOGIN="admin",
  PASSWORD_SECRET_PATH="test_secret"
)
```

Подключение к S3 (Object Storage)

Для создания внешнего источника данных, ведущего на бакет с данными в S3 (Object Storage) необходимо создать объект `EXTERNAL DATA SOURCE`, указав в полях:

- в поле `SOURCE_TYPE` значение `ObjectStorage`;
- в поле `LOCATION` сетевой путь к бакету;
- в поле `AUTH_METHOD` значение `NONE`.

Примечание

В настоящий момент поддерживается работа только с бакетами, не защищенными аутентификацией.

Подключение возможно к любым источникам данных с протоколом доступа AWS S3. Возможно указание любых URL к системам, поддерживающим этот протокол.

Типовые значения поля `LOCATION` при подключении к различным системам к бакету `bucket`:

Название системы	URL
Object Storage	`https://storage.yandexcloud.net/bucket/`
AWS S3	`http://s3.amazonaws.com/bucket/`

Пример

Запрос ниже создает внешний источник данных с именем `TestDataSource`, ведущий в каталог `folder` в бакете `bucket` в Object Storage:

```
CREATE EXTERNAL DATA SOURCE TestDataSource WITH (
  SOURCE_TYPE="ObjectStorage",
  LOCATION="http://s3.amazonaws.com/bucket/folder/",
  AUTH_METHOD="NONE"
```

CREATE STREAMING QUERY

`CREATE STREAMING QUERY` создаёт потоковый запрос.

Синтаксис

```
CREATE [OR REPLACE] STREAMING QUERY [IF NOT EXISTS] <query_name> [WITH (
    <key1> = <value1>,
    <key2> = <value2>,
    ...
)] AS
DO BEGIN
    <query_statement1>;
    <query_statement2>;
    ...
END DO
```

Параметры

- `OR REPLACE` — если потоковый запрос с таким именем уже существует, то он будет заменён на новый запрос с сохранением смещений чтения из топика.
- `IF NOT EXISTS` — не выводить ошибку, если потоковый запрос с таким именем уже существует, в этом случае существующий запрос останется неизменённым.
- `query_name` — имя потокового запроса, который нужно создать.
- `WITH (<key> = <value>)` — список настроек нового потокового запроса, опционально.
- `AS DO BEGIN ... END DO` — полный текст нового потокового запроса, включая все необходимые SQL-выражения. Ограничения для текста запроса приведены в Ограничения, примеры текста см. ниже.

Настройки `OR REPLACE` и `IF NOT EXISTS` нельзя использовать одновременно.

Доступные параметры блока `WITH`:

- `RUN = (TRUE|FALSE)` — запустить запрос после создания, по умолчанию `TRUE`.
- `RESOURCE_POOL = <resource_pool_name>` — имя пула ресурсов, в котором будет выполняться запрос.

Примеры создания потокового запроса см. ниже.

Использование читателя

Читатель (consumer) — именованная подписка на топик, которая хранит текущую позицию чтения.

Читатель создаётся через CLI или при создании топика с помощью CREATE TOPIC. Имя читателя указывается в тексте запроса прагмой:

```
PRAGMA pq.Consumer="my_consumer";
```

Если читатель не указан, чтение из топика выполняется без него. Указание читателя позволяет отслеживать позицию чтения и лаг со стороны топика, например через CLI.

Независимо от наличия читателя позиция чтения потокового запроса сохраняется в чекпоинте.

Примеры

Запись в топик (JSON)

Запрос читает события из входного топика, формирует JSON-объект из отдельных полей и записывает результат в выходной топик.

Функция `AsStruct` создаёт структуру из указанных полей, `Yson::From` преобразует её в Yson, `Yson::SerializeJson` сериализует в JSON-строку, а `ToBytes` конвертирует в тип `String`, который требуется для записи в топик.

> **Примечание**
>
> Потоковые запросы могут работать с локальными и внешними топиками.
>
> В примере:
>
> - `ext_source` — это заранее созданный `external data source`;
> - `input_topic` и `output_topic` — локальные или внешние топики.

```
CREATE STREAMING QUERY my_streaming_query AS
DO BEGIN

    INSERT INTO output_topic -- или внешний топик ext_source.output_topic
    SELECT
        -- Формирование JSON из отдельных полей
        ToBytes(Unwrap(Yson::SerializeJson(Yson::From(
            AsStruct(Id AS id, Name AS name)
        ))))
    FROM
        ext_source.input_topic -- или локальный топик input_topic
    WITH (
        FORMAT = json_each_row,  -- Формат входных данных
        SCHEMA = (               -- Схема входных данных
            Id Uint64 NOT NULL,
            Name Utf8 NOT NULL
        )
```

```
    );

END DO
```

Запись в таблицу

Запрос читает события из топика и записывает их в таблицу `output_table`. Таблица должна быть создана заранее со схемой, соответствующей выбираемым колонкам.

> **Важно**
>
> Запись в таблицы в потоковых запросах поддерживается **только в режиме UPSERT**. Операция `INSERT INTO` не поддерживается, так как при повторной обработке событий (гарантия at-least-once) она привела бы к дублированию строк. При `UPSERT`, если строка с таким первичным ключом уже существует, она будет обновлена, иначе будет вставлена новая строка, а `INSERT INTO` завершится с ошибкой.

```
CREATE STREAMING QUERY my_streaming_query AS
DO BEGIN

    -- Запись в таблицу (только UPSERT, INSERT не поддерживается)
    UPSERT INTO output_table
    SELECT
        Id,
        Name
    FROM
        input_topic -- или внешний топик ext_source.input_topic
    WITH (
        FORMAT = json_each_row,  -- Формат входных данных
        SCHEMA = (               -- Схема входных данных
            Id Uint64 NOT NULL,
            Name Utf8 NOT NULL
        )
    );

END DO
```

Запуск в пуле ресурсов

Запрос создаётся в указанном пуле ресурсов, но не запускается автоматически (`RUN = FALSE`). Это позволяет проверить конфигурацию перед запуском или запустить запрос позже через ALTER STREAMING QUERY.

```
CREATE STREAMING QUERY my_streaming_query WITH (
    RUN = FALSE,                      -- Не запускать автоматически
    RESOURCE_POOL = my_resource_pool  -- Пул ресурсов для выполнения
) AS
DO BEGIN

    INSERT INTO output_topic -- или внешний топик ext_source.output_topic
    SELECT
        ToBytes(Unwrap(Yson::SerializeJson(Yson::From(
            AsStruct(Id AS id, Name AS name)
        ))))
    FROM
        ext_source.input_topic -- или локальный топик input_topic
    WITH (
        FORMAT = json_each_row,
        SCHEMA = (
            Id Uint64 NOT NULL,
            Name Utf8 NOT NULL
        )
    );

END DO
```

Другие примеры: Типичные шаблоны потоковых запросов.

См. также

- Типичные шаблоны потоковых запросов
- Потоковые запросы
- ALTER STREAMING QUERY
- DROP STREAMING QUERY

CREATE EXTERNAL TABLE

Вызов `CREATE EXTERNAL TABLE` создает внешнюю таблицу с указанной схемой данных.

```
CREATE [OR REPLACE] EXTERNAL TABLE [IF NOT EXISTS] table_name (
  column1 type1,
  column2 type2 NOT NULL,
  ...
  columnN typeN NULL
) WITH (
  DATA_SOURCE="data_source_name",
  LOCATION="path",
  FORMAT="format_name",
  COMPRESSION="compression_name"
);
```

Где:

- `OR REPLACE` - если внешняя таблица с таким именем уже существует, она будет заменена новым определением; версия объекта при этом увеличивается.
- `IF NOT EXISTS` - не выводить ошибку, если внешняя таблица с таким именем уже существует; существующая таблица останется без изменений.
- `column1 type1`, `columnN typeN NULL` - колонка данных и ее тип;
- `data_source_name` - имя подключения к S3 (Object Storage).
- `path` - путь внутри бакета с данными. Путь должен вести на существующий каталог внутри бакета.
- `format_name` - один из допустимых типов хранения данных.
- `compression_name` - один из допустимых алгоритмов сжатия.

Допускается использование только ограниченного подмножества типов данных:

- `Bool`.
- `Int8`, `Uint8`, `Int16`, `Uint16`, `Int32`, `Uint32`, `Int64`, `Uint64`.
- `Float`, `Double`.
- `Date`, `DateTime`.
- `String`, `Utf8`.

Без дополнительных модификаторов колонка приобретает опциональный тип тип, и допускает запись `NULL` в качестве значений. Для получения неопционального типа необходимо использовать `NOT NULL`.

Пример

Следующий SQL-запрос создает внешнюю таблицу с именем `s3_test_data`, в котором расположены файлы в формате `CSV` со строковыми полями `key` и `value`, находящиеся внутри бакета по пути `test_folder`, при этом для указания реквизитов подключения используется объект подключение `bucket`:

```
CREATE EXTERNAL TABLE s3_test_data (
  key Utf8 NOT NULL,
  value Utf8 NOT NULL
) WITH (
  DATA_SOURCE="bucket",
  LOCATION="folder",
  FORMAT="csv_with_names",
  COMPRESSION="gzip"
);
```

CREATE OBJECT (TYPE SECRET)

Внимание

Данная команда устарела и будет удалена в будущих версиях YDB. Рекомендуемый синтаксис работы с секретами описан в разделе Секреты.

Для создания секрета используется следующий SQL-запрос:

```
CREATE OBJECT <secret_name> (TYPE SECRET) WITH value = "<secret_value>";
```

Где:

- `secret_name` - имя секрета.
- `secret_value` - содержимое секрета.

Пример

Следующий запрос создает секрет с именем `MySecretName` и значением `MySecretData`.

```
CREATE OBJECT MySecretName (TYPE SECRET) WITH value = "MySecretData";
```

CREATE OBJECT (TYPE SECRET_ACCESS)

Внимание

Данная команда устарела и будет удалена в будущих версиях YDB. Рекомендуемый синтаксис работы с секретами описан в разделе Секреты.

Все права на использование секрета принадлежат создателю секрета. Создатель секрета может предоставить право чтения секрета другому пользователю с помощью управления доступом к секретам.

Для управления доступами к секретам используются специальные объекты `SECRET_ACCESS`. Для выдачи разрешения на использование секрета `secret_name` пользователю `user_name` необходимо создать объект `SECRET_ACCESS` с именем `secret_name:user_name`.

```
CREATE OBJECT `secret_name:user_name` (TYPE SECRET_ACCESS);
```

Где:

- `secret_name` - имя секрета.
- `user_name` - имя пользователя, которому выдается доступ.

Пример

Следующий SQL-запрос выдаст права на использование секрета `MySecretName` пользователю `another_user`:

```
CREATE OBJECT `MySecretName:another_user` (TYPE SECRET_ACCESS);
```

CREATE SECRET

Команда `CREATE SECRET` создаёт секрет.

Синтаксис:

```
CREATE [OR REPLACE] SECRET [IF NOT EXISTS] secret_name
WITH (option = value[, ...])
```

- `OR REPLACE` — если секрет с таким именем уже существует, он будет заменён новым определением.
- `IF NOT EXISTS` — команда не возвращает ошибку, если секрет с таким именем уже существует; существующий объект останется без изменений.
- `secret_name` — имя создаваемого секрета.
- `option` — опция команды:
  - `value` — строка со значением секрета.
  - `inherit_permissions` — опция, при включении которой права на секрет наследуются от директории, в которой секрет создаётся. При отключении опции от директории наследуется только право `DESCRIBE SCHEMA`. Владелец секрета получает все возможные права на него в любом случае. По умолчанию — `False`.

Важно

Конструкции `OR REPLACE` и `IF NOT EXISTS` нельзя использовать одновременно.

Разрешения

Для создания секрета требуется право `CREATE TABLE`.

При использовании `CREATE OR REPLACE SECRET` для существующего секрета требуется право `ALTER SCHEMA` на секрет, так как эта форма `CREATE` изменяет секрет. Если секрет не существует, достаточно права `CREATE TABLE` на родительскую директорию.

Примеры

Создать секрет в корне базы с именем `secret_name` и значением `secret_value`:

```
CREATE SECRET secret_name WITH (value = "secret_value");
```

Создать секрет в директории `dir` в корне базы с именем `secret_name` и значением `secret_value`. Если директория `dir` не существует, она будет создана:

```
CREATE SECRET `dir/secret_name` WITH (value = "secret_value");
```

Создать секрет в корне базы с именем `secret_name` и значением `secret_value` с правами такими же, как у родительской директории секрета:

```
CREATE SECRET secret_name WITH (value = "secret_value", inherit_permissions = True);
```

Создать секрет с именем `secret_name`, только если он не существует; если он существует, существующий секрет останется без изменений:

```
CREATE SECRET IF NOT EXISTS secret_name WITH (value = "secret_value");
```

Создать или заменить секрет с именем `secret_name`; если он существует, он будет заменён новым определением:

```
CREATE OR REPLACE SECRET secret_name WITH (value = "secret_value");
```

См. также

- ALTER SECRET
- DROP SECRET

CREATE TABLE

Вызов `CREATE TABLE` создает таблицу с указанной схемой данных и ключевыми колонками (`PRIMARY KEY`). Позволяет определить вторичные индексы на создаваемой таблице.

```
CREATE TABLE [IF NOT EXISTS] <table_name> (
  [<column_name> <column_data_type>] [FAMILY <family_name>] [NULL | NOT NULL] [DEFAULT <default_value>]
  [COMPRESSION([algorithm=<algorithm_name>[, level=<value>]])]
  [ENCODING([OFF|DICT])]
  [, ...],
    INDEX <index_name>
      [GLOBAL|LOCAL]
      [UNIQUE]
      [SYNC|ASYNC]
      [USING <index_type>]
      ON ( <index_columns> )
      [COVER ( <cover_columns> )]
      [WITH ( <parameter_name> = <parameter_value>[, ...])]
    [, ...]
  PRIMARY KEY ( <column>[, ...]),
  [FAMILY <column_family> ( family_options[, ...])]
)
[PARTITION BY HASH ( <column>[, ...])]
[WITH (<setting_name> = <setting_value>[, ...])]

[AS SELECT ...]
```

Параметры запроса

table_name

Путь создаваемой таблицы.

При выборе имени для таблицы учитывайте общие правила именования схемных объектов.

IF NOT EXISTS

Если таблица с указанным именем уже существует, выполнение оператора полностью пропускается — не происходит никаких проверок или сопоставления схемы, и никакой ошибки не возникает. Обратите внимание, что существующая таблица может отличаться по структуре от той, которую вы хотели бы создать этим запросом — сравнение или проверка эквивалентности не производится.

column_name

Имя колонки, создаваемой в новой таблице.

При выборе имени для колонки учитывайте общие правила именования колонок.

column_data_type

Тип данных колонки. Полный список типов данных, которые поддерживает YDB доступен в разделе Типы данных YQL.

FAMILY <family_name> (настройка колонки)

> **Важно**
>
> Поддерживается только для строковых таблиц.

Указание принадлежности данной колонки к указанной группе колонок. Подробнее в разделе Группы колонок.

DEFAULT <default_value>

> **Важно**
>
> Опция `DEFAULT` поддерживается:
>
> - Только для строковых таблиц.
> - Только с литеральными значениями.

Позволяет задать значение по умолчанию для колонки. Если при вставке строки значение для данной колонки не указано, будет использовано указанное значение по умолчанию. Значение по умолчанию должно соответствовать типу данных колонки.

Конструкция `DEFAULT false NOT NULL` недопустима по причине неоднозначности интерпретации. В таком случае следует использовать перечисление через запятые или изменить порядок опций.

NULL

Опция колонки, которое разрешает запись `NULL` в качестве значений колонки. Это является поведением по умолчанию.

NOT NULL

Ограничение колонки, которое запрещает запись `NULL` в качестве значений колонки. По умолчанию это ограничение отсутствует.

COMPRESSION([algorithm=<algorithm_name>[, level=]])

> **Важно**
>
> Поддерживается только для колоночных таблиц.

Для колонок можно задать следующие параметры сжатия:

- `algorithm` — алгоритм сжатия данных. Допустимые значения: `off` (отключение сжатия), `lz4`, `zstd`.
- `level` — уровень сжатия, поддерживается только для алгоритма `zstd` (допустимы значения от 0 до 22).

Если `COMPRESSION()` указан без параметров, для колонки используется сжатие по умолчанию. Сейчас это `lz4`; в будущих версиях появится возможность настраивать сжатие по умолчанию на уровне кластера или таблицы.

ENCODING([OFF|DICT])

> **Важно**
>
> Поддерживается только для колоночных таблиц.

Позволяет задать способ кодирования данных колонки.

Доступные варианты:

- `ENCODING(DICT)` — включает словарное кодирование (dictionary encoding). Повторяющиеся значения заменяются небольшими целочисленными идентификаторами, а сами значения хранятся в словаре. Словарное кодирование эффективно для колонок с низкой кардинальностью (небольшим количеством уникальных значений). Позволяет сократить объем хранимых данных и ускорить выполнение некоторых операций. Поддерживается только для сравнимых типов данных, таких как `String`, `Timestamp`, `UInt64` и других. Для несравнимых типов, таких как `Json`, `JsonDocument` или `Yson`, использование `ENCODING(DICT)` приведет к ошибке.
- `ENCODING(OFF)` — отключает специальное кодирование. Данные будут храниться в стандартном формате без дополнительного кодирования.

Если `ENCODING()` задается без параметров, то для колонки будет использоваться кодирование по умолчанию. Сейчас это `OFF`; в будущих версиях появится возможность настраивать кодирование по умолчанию на уровне базы или таблицы.

INDEX

Определение индекса на таблице. Поддерживаются:

- вторичные индексы,
- векторные индексы,
- полнотекстовые индексы,
- Блум-индексы,
- min_max-индекс,
- JSON-индексы.

PRIMARY KEY

Определение первичного ключа таблицы. Указывает колонки, которые составляют первичный ключ в порядке перечисления. Подробнее о выборе первичного ключа в разделе Выбор первичного ключа.

PARTITION BY HASH

Определение ключей партиционирования для **колоночных** таблиц. Указывает колонки, по хэшу которых выполняется партиционирование данных. Колонки должны быть частью первичного ключа. При этом колонки не обязательно должны быть префиксом или суффиксом -- требование быть частью первичного ключа.

Если параметр не будет указан, таблица будет разбита на партиции по тем же колонкам, которые входят в первичный ключ. Как правильно выбирать ключи для партиционирования в колоночных таблицах, читайте в статье Выбор ключей для максимальной производительности колоночных таблиц.

Подробнее о партиционировании колоночных таблиц читайте в разделе Партицирование колоночной таблицы.

FAMILY <column_family> (настройка группы колонок)

Определение группы колонок с заданными параметрами. Подробнее в разделе Группы колонок.

WITH

Дополнительные параметры создания таблицы. Подробнее в разделе Дополнительные параметры (WITH).

Примечание

YDB поддерживает два типа таблиц:

- Строковые.
- Колоночные.

Тип таблицы при создании задается параметром `STORE` в блоке `WITH`, где `ROW` означает строковую таблицу, а `COLUMN` — колоночную:

```
CREATE <table_name> (
  columns
  ...
)

WITH (
  STORE = COLUMN -- Default value ROW
)
```

По умолчанию, если параметр `STORE` не указан, создается строковая таблица.

Примечание

При выборе имени для таблицы учитывайте общие правила именования схемных объектов.

AS SELECT

Создание и заполнение таблицы на основе результатов запроса `SELECT`. Подробнее в разделе Создание и заполнение таблицы на основе результатов запроса.

Примеры создания таблиц

Создание строковой таблицы

```
  CREATE TABLE <table_name> (
    a Uint64,
    b Uint64,
    c Float,
    PRIMARY KEY (a, b)
  );
```

Пример создания таблицы с использованием значения по умолчанию (DEFAULT):

```
  CREATE TABLE table_with_default (
  id Uint64,
  name String DEFAULT "unknown",
  score Double NOT NULL DEFAULT 0.0,
  PRIMARY KEY (id)
);
```

Для ключевых колонок допускаются только примитивные и серийные типы данных, для неключевых колонок допускаются только примитивные.

Без дополнительных модификаторов колонка приобретает опциональный тип, и допускает запись `NULL` в качестве значений. Для получения неопционального типа необходимо использовать `NOT NULL`.

Обязательно указание `PRIMARY KEY` с непустым списком колонок. Эти колонки становятся частью ключа в порядке перечисления.

Пример создания строковой таблицы с использованием опций партиционирования:

```
CREATE TABLE <table_name> (
  a Uint64,
  b Uint64,
  c Float,
  PRIMARY KEY (a, b)
)
WITH (
  AUTO_PARTITIONING_BY_SIZE = ENABLED,
  AUTO_PARTITIONING_PARTITION_SIZE_MB = 512
);
```

Такой код создаст строковую таблицу с включенным автоматическим партиционированием по размеру партиции (`AUTO_PARTITIONING_BY_SIZE`) и предпочитаемым размером каждой партиции (`AUTO_PARTITIONING_PARTITION_SIZE_MB`) в 512 мегабайт. Полный список опций партиционирования строковой таблицы находится в разделе Партиционирование строковой таблицы статьи Таблица.

Создание колоночной таблицы

```
CREATE TABLE table_name (
  a Uint64 NOT NULL,
  b Timestamp NOT NULL,
  c Float,
  PRIMARY KEY (a, b)
```

```
)
PARTITION BY HASH(b)
WITH (
  STORE = COLUMN
);
```

Для колоночных таблиц можно явно указать, по каким колонкам будет происходить партиционирование, с помощью конструкции `PARTITION BY HASH`. Обычно для этого выбирают колонки первичного ключа с большим числом уникальных значений, например, `Timestamp`. Если `PARTITION BY HASH` не указать, партиционирование произойдёт автоматически по всем колонкам, входящим в первичный ключ. Подробнее о выборе и работе ключей партиционирования в колоночных таблицах читайте в статье Выбор ключей для максимальной производительности колоночных таблиц.

В настоящий момент колоночные таблицы не поддерживают автоматического репартицирования, поэтому важно указывать правильное число партиций при создании таблицы с помощью параметра `AUTO_PARTITIONING_MIN_PARTITIONS_COUNT`:

```
CREATE TABLE table_name (
  a Uint64 NOT NULL,
  b Timestamp NOT NULL,
  c Float,
  PRIMARY KEY (a, b)
)
PARTITION BY HASH(b)
WITH (
  STORE = COLUMN,
  AUTO_PARTITIONING_MIN_PARTITIONS_COUNT = 10
);
```

Такой код создаст колоночную таблицу с 10-ю партициями. С полным списком опций партиционирования колоночных таблиц можно ознакомиться в разделе Партицирование колоночной таблицы статьи Таблица.

При создании строковых таблиц возможно задать:

- Вторичный индекс.
- Векторный индекс.
- Полнотекстовый индекс.
- JSON-индекс.
- Блум-индекс.
- Группы колонок.
- Дополнительные параметры.
- Создание и заполнение таблицы на основе результатов запроса.

Для колоночных таблиц при их создании возможно задать:

- Блум-индекс.
- Дополнительные параметры.
- Создание и заполнение таблицы на основе результатов запроса.

INDEX

Конструкция `INDEX` используется для определения вторичного индекса для строковых таблиц:

```
CREATE TABLE `<table_name>` (
  ...
    INDEX `<index_name>`
    [GLOBAL|LOCAL]
    [UNIQUE]
    [SYNC|ASYNC]
    [USING <index_type>]
    ON ( <index_columns> )
    [COVER ( <cover_columns> )]
    [WITH ( <parameter_name> = <parameter_value>[, ...])]
  [,   ...]
)
```

где:

- `GLOBAL/LOCAL` — глобальный или локальный индекс, в зависимости от типа индекса (`<index_type>`) может быть доступен только один из них:
  - `GLOBAL` — индекс, реализованный в виде отдельной таблицы или набора таблиц. Синхронное обновление такого индекса требует распределённых транзакций.
  - `LOCAL` — локальный индекс в рамках шарда колоночной или строковой таблицы, не требует распределённых транзакций при обновлении, однако не обеспечивает прюнинг при поиске.
- `<index_name>` — уникальное имя индекса, по которому будет возможно обращение к данным.
- `SYNC/ASYNC` — признак синхронности индекса.
  - `SYNC` - синхронный индекс. Значение по умолчанию.
  - `ASYNC` - асинхронный индекс.
- `UNIQUE` — признак уникального вторичного индекса. Уникальный индекс должен быть глобальным синхронным (`GLOBAL UNIQUE SYNC`) и не должен содержать конструкцию `USING <index_type>` .
- `<index_type>` - тип индекса, в настоящее время поддерживаются:
  - `secondary` — вторичный индекс. Для вторичных индексов доступен только режим `GLOBAL` . Это тип индекса по умолчанию.
  - `vector_kmeans_tree` — векторный индекс. Подробнее описан в разделе Векторный индекс.
  - `fulltext_plain` — базовый полнотекстовый индекс. Подробнее описан в Полнотекстовый индекс.
  - `fulltext_relevance` — полнотекстовый индекс со статистикой BM25 для расчёта релевантности. Подробнее описан в Полнотекстовый индекс.
  - `json` — JSON-индекс для ускорения предикатов `JSON_EXISTS` и `JSON_VALUE` по колонке типа `Json` или `JsonDocument` . Подробнее описан в JSON-индекс.
  - `bloom_filter` — локальный Блум-индекс. Доступен только `LOCAL` . См. ALTER TABLE ADD INDEX.
  - `bloom_ngram_filter` — локальный N-граммный Блум-индекс. Доступен только `LOCAL` . См. ALTER TABLE ADD INDEX.
  - `min_max` — локальный min/max-индекс. Доступен только `LOCAL` . См. ALTER TABLE ADD INDEX.
- `<index_columns>` — список имён колонок создаваемой таблицы через запятую, по которому определяется состав и порядок включения колонок в ключ индекса. Обязательно должен быть указан. Ключ индекса будет состоять из этих колонок с добавлением колонок первичного ключа таблицы.
- `<cover_columns>` — список имён колонок создаваемой таблицы через запятую, которые будут сохранены в индексе дополнительно к колонкам ключа индекса, давая возможность получить дополнительные данные без обращения за ними в таблицу. По умолчанию пуст.
- `<parameter_name>` и `<parameter_value>` — параметры индекса, специфичные для конкретного `<index_type>` . Некоторые параметры индекса нельзя задать при его создании. См. Изменение параметров индекса.

Пример

```
CREATE TABLE my_table (
    a Uint64,
    b Bool,
    c Utf8,
    d Date,
    INDEX idx_d GLOBAL ON (d),
    INDEX idx_ba GLOBAL ASYNC ON (b, a) COVER (c),
    INDEX idx_uniq GLOBAL UNIQUE SYNC ON (c),
    PRIMARY KEY (a)
)
```

Векторный индекс

Векторный индекс в строковых таблицах создаётся с помощью того же синтаксиса, что и вторичные индексы, при указании `vector_kmeans_tree` в качестве типа индекса. Подмножество доступного для векторных индексов синтаксиса:

```
CREATE TABLE `<table_name>` (
    ...
    INDEX `<index_name>`
        GLOBAL
        [SYNC]
        USING vector_kmeans_tree
        ON ( <index_columns> )
        [COVER ( <cover_columns> )]
        [WITH ( <parameter_name> = <parameter_value>[, ...])]
    [,   ...]
)
```

Где:

- `<index_name>` - уникальное имя индекса для доступа к данным
- `SYNC` - указывает на синхронную запись данных в индекс. Это единственная доступная на данный момент опция, явно указывать не обязательно.
- `<index_columns>` - список колонок таблицы через запятую, используемых для поиска по индексу (последняя колонка используется как эмбеддинг, остальные - как фильтрующие колонки)
- `<cover_columns>` - список дополнительных колонок создаваемой таблицы, которые будут сохранены в индексе для возможности их извлечения без обращения к основной таблице
- `<parameter_name>` и `<parameter_value>` - список параметров в формате ключ-значение:

- общие параметры для всех векторных индексов:
  - `vector_dimension` - размерность вектора эмбеддинга (значение от 1 до 65536);
  - `vector_type` - тип значений вектора (`float` , float16, bfloat16, `uint8` , `int8` или `bit`);
  - `distance` - функция расстояния (`cosine` , `manhattan` или `euclidean`), взаимоисключающий с `similarity` ;
  - `similarity` - функция схожести (`inner_product` или `cosine`), взаимоисключающий с `distance` ;
- специфичные параметры для `vector_kmeans_tree` (подробнее о типе индекса):
  - `clusters` - количество центроидов для алгоритма k-means (значение от 2 до 2048);
  - `levels` - количество уровней в дереве (значение от 1 до 16);
  - `overlap_clusters` - максимальное число ближайших листовых кластеров, в которые добавляется каждый вектор; оно не должно превышать `clusters` . Если параметр не указан, YDB может автоматически включить перекрытие для больших наборов данных. Если перекрытие не включено, каждый вектор попадает в один кластер;
  - `adaptive_clusters` - для индексов с фильтрацией автоматически выбирать количество кластеров для каждого значения фильтруемой колонки в зависимости от количества векторов у этого значения (`true` или `false` , по умолчанию `false` ; включается при автоматическом подборе параметров). Подробнее см. Адаптивное количество кластеров;
  - общее количество узлов в дереве, рассчитываемое как `clusters` в степени `levels` , должно быть не более чем 1073741824;
  - произведение `vector_dimension` на `clusters` должно быть не более чем 4194304.

Важно

Создавать векторный индекс рекомендуется после загрузки данных в таблицу, так как индекс, созданный на пустой таблице, будет вырожденным и не будет ускорять поиск. Подробности смотрите в разделе Обновление векторных индексов.

Важно

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Пример

```
CREATE TABLE user_articles (
    article_id Uint64,
    user String,
    title String,
    text String,
    embedding String,
    INDEX emb_cosine_idx GLOBAL SYNC USING vector_kmeans_tree
    ON (user, embedding) COVER (title, text)
    WITH (
        distance="cosine",
        vector_type="float",
        vector_dimension=512,
        clusters=128,
        levels=2,
        overlap_clusters=3
    ),
    PRIMARY KEY (article_id)
)
```

Полнотекстовый индекс

Полнотекстовые индексы в строковых таблицах создаются с помощью того же синтаксиса, что и вторичные индексы, при указании `fulltext_plain` или `fulltext_relevance` в качестве типа индекса. Подмножество доступного для полнотекстовых индексов синтаксиса:

```
CREATE TABLE `<table_name>` (
    ...
    INDEX `<index_name>`
        GLOBAL
        [SYNC]
        USING fulltext_plain | fulltext_relevance
        ON ( [<prefix_columns>,] <text_column> )
        [COVER ( <cover_columns> )]
        WITH ( <parameter_name> = <parameter_value>[, ...])
    [,   ...]
)
```

Где:

- `<index_name>` - уникальное имя индекса для доступа к данным
- `SYNC` - указывает на синхронную запись данных в индекс. Это единственная доступная на данный момент опция, явно указывать не обязательно.
- `<prefix_columns>` - необязательные колонки фильтрации для полнотекстового индекса с фильтрацией; в запросе требуется равенство для каждой колонки префикса
- `<text_column>` - одна колонка типа `String` или `Utf8`, указанная последней в `ON`
- `<cover_columns>` - список дополнительных колонок создаваемой таблицы, которые будут сохранены в индексе для возможности их извлечения без обращения к основной таблице
- `<parameter_name>` и `<parameter_value>` - список параметров в формате ключ-значение:
- общие параметры для всех полнотекстовых индексов:
  - `analyzer` - готовый анализатор: `standard` (стандартный токенизатор, нижний регистр и удаление английских стоп-слов), `snowball` (стандартный токенизатор, нижний регистр, удаление стоп-слов и лемматизация Snowball; требуется `language`) или `keyword` (весь текст становится одним токеном с сохранением регистра)
  - `tokenizer` - тип токенизатора (`standard`, `whitespace`, `alphanumeric` или `keyword`)
  - `use_filter_lowercase` - фильтр приведения к нижнему регистру (`true` или `false`)
  - `use_filter_stopwords` - удаление частотных слов, например `the` и `and` (`true` или `false`). Поддерживаются английский и русский языки; если `language` не указан, используется английский
  - `use_filter_length` - фильтр по длине токена (`true` или `false`); при значении `true` токены короче `filter_length_min` или длиннее `filter_length_max` не индексируются
  - `filter_length_min` - минимальная длина токена (положительное целое); применяется только при `use_filter_length=true`
  - `filter_length_max` - максимальная длина токена (положительное целое); применяется только при `use_filter_length=true`
  - `use_filter_snowball` - фильтр лемматизации Snowball (`true` или `false`)
  - `language` - язык или список языков через запятую для лемматизатора Snowball. Пробелы вокруг запятых допустимы, например `"english, russian"`. Готовый анализатор `snowball` дополнительно удаляет стоп-слова, поэтому вместе с ним можно указать только `english` и `russian`
  - `use_filter_ngram` - фильтр N-грамм (`true` или `false`)
  - `use_filter_edge_ngram` - фильтр краевых N-грамм (`true` или `false`)
  - `filter_ngram_min_length` - минимальная длина N-граммы (положительное целое)
  - `filter_ngram_max_length` - максимальная длина N-граммы (положительное целое)

> **Важно**
>
> Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Примеры

Базовый пример

```
CREATE TABLE articles (
    id Uint64,
    title String,
    body String,
    INDEX ft_idx GLOBAL USING fulltext_relevance
    ON (body) COVER (title)
    WITH (
        tokenizer=standard,
        use_filter_lowercase=true
    ),
    PRIMARY KEY (id)
)
```

Лемматизация для нескольких языков

Готовый анализатор Snowball выполняет лемматизацию и позволяет искать формы английских и русских слов в одной текстовой колонке. В `language` можно перечислить несколько поддерживаемых языков через запятую, в том числе с пробелами:

```
CREATE TABLE documents (
    id Uint64,
    body Utf8,
    PRIMARY KEY (id),
    INDEX ft_idx GLOBAL USING fulltext_relevance ON (body)
    WITH (analyzer="snowball", language="english, russian")
);
```

Пример с N-граммами

N-граммы позволяют находить подстроки внутри слов. При использовании фильтра N-грамм каждое слово разбивается на все возможные подстроки заданной длины. Например, слово "search" с параметрами `filter_ngram_min_length=3` и `filter_ngram_max_length=5` будет разбито на: "sea", "ear", "arc", "rch" (3-граммы), "sear", "earc", "arch" (4-граммы), "searc", "earch" (5-граммы). Это позволяет находить документы по частичному совпадению в любой части слова.

```
CREATE TABLE products (
    id Uint64,
    name String,
    description String,
    INDEX ft_ngram_idx GLOBAL USING fulltext_plain
    ON (name)
    WITH (
        tokenizer=standard,
        use_filter_lowercase=true,
        use_filter_ngram=true,
        filter_ngram_min_length=3,
        filter_ngram_max_length=5
    ),
    PRIMARY KEY (id)
)
```

Пример с краевыми N-граммами

Краевые N-граммы (edge n-grams) создают подстроки только от начала слова. Например, слово "search" с параметрами `filter_ngram_min_length=2` и `filter_ngram_max_length=10` будет разбито на: "se", "sea", "sear", "searc", "search". Это особенно полезно для реализации функции автодополнения (autocomplete), когда нужно находить слова, начинающиеся с введённого пользователем префикса.

```
CREATE TABLE users (
    id Uint64,
    username String,
    email String,
    INDEX ft_edge_ngram_idx GLOBAL USING fulltext_plain
    ON (username)
    WITH (
        tokenizer=standard,
        use_filter_lowercase=true,
        use_filter_edge_ngram=true,
        filter_ngram_min_length=2,
        filter_ngram_max_length=10
    ),
    PRIMARY KEY (id)
)
```

JSON-индекс

JSON-индексы в строковых таблицах создаются с помощью того же синтаксиса, что и вторичные индексы, при указании `json` в качестве типа индекса. Подмножество доступного для JSON-индексов синтаксиса:

```
CREATE TABLE `<table_name>` (
    ...
    INDEX `<index_name>`
        GLOBAL
        [SYNC]
        USING json
        ON ( <json_column> )
    [,   ...]
)
```

Где:

- `<index_name>` — уникальное имя индекса для доступа к данным.
- `SYNC` — указывает на синхронное обновление индекса. Это единственный доступный для JSON-индексов режим, явно указывать не обязательно.
- `<json_column>` — колонка таблицы типа `Json` или `JsonDocument`. JSON-индекс строится только по одной колонке.

JSON-индекс не поддерживает выражение `COVER` — попытка указать его приведёт к ошибке.

> **Важно**
>
> Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Пример

```
CREATE TABLE documents (
    id Uint64 NOT NULL,
    payload JsonDocument NOT NULL,
    INDEX json_idx GLOBAL USING json ON (payload),
    PRIMARY KEY (id)
)
```

В этом примере создаётся таблица `documents` с JSON-индексом `json_idx` по колонке `payload`. Индекс будет использоваться запросами, в предикате `WHERE` которых содержатся вызовы `JSON_EXISTS` или `JSON_VALUE` по колонке `payload`.

Блум-индекс

Блум-индексы — локальные индексы, их можно задать только с ключевым словом `LOCAL`. При создании таблицы в секции `INDEX` используются типы `bloom_filter` или `bloom_ngram_filter` (по аналогии с вторичным индексом, но с обязательным `LOCAL` и соответствующим `USING`). См. также локальные индексы.

```
CREATE TABLE `<table_name>` (
    ...
    INDEX `<index_name>`
        LOCAL
        USING bloom_filter | bloom_ngram_filter
        ON ( <index_columns> )
        [WITH ( <parameter_name> = <parameter_value>[, ...])]
    [,   ...]
)
```

Где:

- `<index_name>` — имя индекса.
- `LOCAL` — обязательное ключевое слово для Блум-индексов.
- `<index_columns>` — список колонок, по которым строится индекс; количество колонок зависит от типа таблицы и типа индекса. На строковых таблицах индексируемые колонки `bloom_filter` должны образовывать левый префикс первичного ключа.
- Для Блум-индексов не поддерживаются колонки покрытия (`COVER (...)`).

Параметры `WITH (...)`:

- `bloom_filter`
  - `false_positive_probability` — целевая вероятность ложноположительного срабатывания фильтра: доля фрагментов, которые фильтр не отсечёт, хотя искомого значения в них нет (диапазон `(0, 1)`). Меньшее значение уменьшает число лишних чтений, но увеличивает размер индекса.
    - Если параметр не указан, для строковых таблиц используется `0.0001`, для колоночных — `0.1`. Более строгий порог по умолчанию в строковых таблицах соответствует OLTP-сценарию точечных чтений; в колоночных — аналитическим сканам, где допустим больший компромисс между размером индекса и отсечением фрагментов.
  - Индексируемые колонки — типы YQL, для которых определено сравнение на равенство, кроме `Yson`, `Json` и `JsonDocument` (см. операторы сравнения). В колоночной таблице допускается одна колонка; в строковой — несколько. На строковых таблицах индексируемые колонки должны образовывать левый префикс первичного ключа.
- `bloom_ngram_filter` (колонки `String`, `Utf8`; только колоночные таблицы)
  - `ngram_size` — размер n-граммы, целое число от `3` до `8` (по умолчанию `3`).
  - `false_positive_probability` — целевая вероятность ложноположительного срабатывания (диапазон `(0, 1)`; по умолчанию `0.1`).
  - `case_sensitive` — учёт регистра при построении n-грамм: `true` или `false` (по умолчанию `true`).

Создание и изменение таких индексов для уже существующей таблицы описаны в разделе `ALTER TABLE ADD INDEX`.

Примеры

Индекс `bloom_filter`

```
CREATE TABLE events (
    id Uint64,
    resource_id Utf8,
    PRIMARY KEY (id),
    INDEX idx_bloom LOCAL USING bloom_filter
        ON (resource_id)
        WITH (false_positive_probability = 0.01)
);
```

Индекс `bloom_ngram_filter`

```
CREATE TABLE events (
    id Uint64,
    message Utf8,
    PRIMARY KEY (id),
    INDEX idx_ngram LOCAL USING bloom_ngram_filter
        ON (message)
        WITH (
            ngram_size = 3,
            false_positive_probability = 0.01,
            case_sensitive = true
        )
);
```

min_max-индекс

min_max-индекс — локальный индекс, его можно задать только с ключевым словом `LOCAL`. При создании таблицы в секции `INDEX` используется тип `min_max` (по аналогии с вторичным индексом, но с обязательным `LOCAL` и соответствующим `USING`). См. также локальные индексы.

```
CREATE TABLE `<table_name>` (
    ...
    INDEX `<index_name>`
        LOCAL
        USING min_max
        ON ( <index_column> )
    [,   ...]
)
```

Где:

- `<index_name>` — имя индекса.
- `LOCAL` — обязательное ключевое слово для min_max-индекса.
- `<index_column>` — колонка, по которой строится индекс. Нужно указать ровно одну колонку.
- Для min_max-индекса не поддерживаются колонки покрытия (`COVER (...)`) и дополнительные колонки данных.

Параметры `WITH (...)`:

У min_max-индекса нет специфичных параметров `WITH (...)`.

Создание min_max-индекса для уже существующей таблицы описано в разделе `ALTER TABLE ADD INDEX`.

Пример

```
CREATE TABLE events (
    id Uint64,
    created_at Timestamp,
    level Int32,
    PRIMARY KEY (id),
    INDEX idx_created_at LOCAL USING min_max
        ON (created_at),
    INDEX idx_level LOCAL USING min_max
        ON (level)
)
WITH (
    STORE = COLUMN
);
```

Группы колонок

> **Важно**
>
> Поддерживается только для строковых таблиц.

Колонки одной таблицы можно объединять в группы, чтобы задать следующие параметры:

- `DATA` — тип устройства хранения для данных колонок этой группы. Допустимые значения: `"ssd"`, `"rot"`.
- `COMPRESSION` — кодек сжатия данных. Допустимые значения: `"off"`, `"lz4"`.
- `CACHE_MODE` — режим кэширования. Допустимые значения: `"in_memory"`, `"regular"`.

По умолчанию все колонки находятся в одной группе с именем `default`. При необходимости параметры этой группы тоже можно переопределить. В противном случае применяются предопределённые значения.

Пример

В примере ниже для создаваемой таблицы добавляется группа колонок `family_large`, которая устанавливается для колонки `series_info`, а также переопределяются параметры для группы `default`, которая по умолчанию применяется ко всем остальным колонкам:

```
CREATE TABLE series_with_families (
    series_id Uint64,
    title Utf8,
    series_info Utf8 FAMILY family_large,
    release_date Uint64,
    PRIMARY KEY (series_id),
    FAMILY default (
        DATA = "ssd",
        COMPRESSION = "off",
        CACHE_MODE = "in_memory"
    ),
    FAMILY family_large (
        DATA = "rot",
        COMPRESSION = "lz4",
        CACHE_MODE = "regular"
    )
);
```

> **Примечание**
>
> Доступные типы устройств хранения зависят от конфигурации кластера YDB.

Дополнительные параметры (WITH)

Для таблицы может быть указан ряд специфичных для YDB параметров. При создании таблицы такие параметры перечисляются в блоке `WITH`:

```
CREATE TABLE table_name (...)
WITH (
    key1 = value1,
    key2 = value2,
    ...
)
```

Здесь key — это название параметра, а value — его значение.

Перечень допустимых имен параметров и их значений приведен на странице описания таблицы YDB.

Например, такой запрос создаст строковую таблицу с включенным автоматическим партиционированием по размеру партиции и предпочитаемым размером каждой партиции 512 мегабайт:

```
CREATE TABLE my_table (
    id Uint64,
    title Utf8,
    PRIMARY KEY (id)
)
WITH (
    AUTO_PARTITIONING_BY_SIZE = ENABLED,
    AUTO_PARTITIONING_PARTITION_SIZE_MB = 512
);
```

Колоночная таблица создаётся путём указанием параметра `STORE = COLUMN` в блоке `WITH`:

```
 CREATE TABLE table_name (
    a Uint64 NOT NULL,
    b Timestamp NOT NULL,
    c Float,
    PRIMARY KEY (a, b)
  )
  PARTITION BY HASH(b)
  WITH (
    STORE = COLUMN
  );
```

Свойства и возможности колоночных таблиц описаны в статье Таблица, а специфика их создания через YQL описана на странице CREATE TABLE.

Time to Live (TTL) и вытеснение данных во внешнее хранилище

В блоке `WITH` можно задать TTL (Time to Live) — время жизни строки для строковых и колоночных таблиц. TTL автоматически удаляет или вытесняет во внешнее хранилище строки, когда проходит указанное количество секунд от времени, записанного в TTL-колонку. TTL можно задать при создании строковой и колоночной таблицы или добавить позже командой `ALTER TABLE` только в строковую таблицу.

Краткая форма значения TTL для задания времени удаления строк:

```
Interval("<literal>") ON column [AS <unit>]
```

Общий вид значения TTL:

```
Interval("<literal1>") action1, Interval("<literal1>") action2, ..., Interval("<literal1>") actionN ON column
[AS <unit>]
```

- `action` — действие, которое выполняется при срабатывании TTL-выражения. Допустимые значения:
  - `DELETE` — удалить строку;
  - `TO EXTERNAL DATA SOURCE <path>` — вытеснить строку во внешнее хранилище, заданное внешним источником данных по пути `<path>`.
- `<unit>` — единица измерения, указывается только для колонок с числовым типом:
  - `SECONDS`;
  - `MILLISECONDS`;
  - `MICROSECONDS`;
  - `NANOSECONDS`.

Пример создания строковой и колоночной таблицы с TTL:

Создание строковой таблицы с TTL

```
CREATE TABLE my_table (
    id Uint64,
    title Utf8,
    expire_at Timestamp,
    PRIMARY KEY (id)
)
WITH (
```

```
    TTL = Interval("PT0S") ON expire_at
);
```

Создание колоночной таблицы с TTL

```
CREATE TABLE table_name (
    a Uint64 NOT NULL,
    b Timestamp NOT NULL,
    c Float,
    PRIMARY KEY (a, b)
)
PARTITION BY HASH(b)
WITH (
    STORE = COLUMN,
    TTL = Interval("PT0S") ON b
);
```

Пример создания колоночной таблицы с вытеснением строк во внешнее хранилище:

> **Важно**
>
> Поддерживается только для колоночных таблиц. Поддержка функциональности для строковых таблиц находится в разработке.

```
CREATE TABLE table_name (
    a Uint64 NOT NULL,
    b Timestamp NOT NULL,
    c Float,
    PRIMARY KEY (a, b)
)
PARTITION BY HASH(b)
WITH (
    STORE = COLUMN,
    TTL =
        Interval("PT1D") TO EXTERNAL DATA SOURCE `/Root/s3`,
        Interval("P2D") DELETE
    ON b
);
```

Создание и заполнение таблицы на основе результатов запроса

> **Важно**
>
> Поддерживается только для колоночных таблиц. Поддержка функциональности для строковых таблиц находится в разработке.

Вызов `CREATE TABLE AS` создает новую таблицу, которая заполнена данными из результатов запроса.

```
CREATE TABLE table_name (
    PRIMARY KEY ( column, ... )
)
WITH ( key = value, ... )
AS SELECT ...
```

Имена и типы колонок будут соответствовать результатам `SELECT`.
Для колонок неопционального типа также будет выставлен модификатор `NOT NULL`.

Синтаксис `CREATE TABLE AS` позволяет задать только первичный ключ и параметры `WITH`, поэтому при создании таблицы не поддерживается указание имен колонок, вторичных индексов, векторных индексов, полнотекстовых индексов, локальных блум-индексов, локальных min_max-индексов, групп колонок. Имена и типы данных для столбцов новой таблицы автоматически наследуются из результирующего набора запроса SELECT. Все вышеперечисленное можно изменять при помощи `ALTER TABLE` после создания таблицы. При этом поддерживаются дополнительные параметры.

Особенности

> **Важно**
>
> Запись строк производится с полной перезаписью строки, как при использовании `REPLACE INTO`, но при этом отсутствуют гарантии на порядок записи строк в таблицу.
>
> Если `SELECT` вернул 2 или более строки с одним и тем же значением первичного ключа, то после завершения выполнения `CREATE TABLE AS` в созданной таблице будет только одна запись с таким значением первичного ключа. При этом какая именно из записей найденных в `SELECT` будет добавлена - в общем случае не определено.

- `CREATE TABLE AS` поддерживается только в режиме неявного контроля транзакций. Таблица появится по указанному пути уже заполненной.
- `CREATE TABLE AS` может быть только единственным DML/DDL выражением в запросе. Допустимо использование PRAGMA, DECLARE и именованных выражений.
- `CREATE TABLE AS` не конфликтует с другими транзакциями. При выполнении запроса не используются блокировки, а все чтения производятся из консистентного снапшота. Балансировка или разделение таблеток не приводят к ошибкам.
- `CREATE TABLE AS` позволяет использовать в одном запросе и колоночные таблицы, и строковые таблицы.
- `CREATE TABLE AS` создаёт таблицу во временной директории `.tmp/sessions`, а после успешной записи данных перемещает её в указанное место. Если операция прервётся из-за ошибки, временная таблица не удаляется мгновенно, а остаётся в системе ещё на некоторое время.

Примеры

- Создание колоночной таблицы из результатов запроса

```
CREATE TABLE my_table (
    PRIMARY KEY (key1, key2)
) WITH (
    STORE=COLUMN
) AS SELECT
    key AS key1,
    Unwrap(other_key) AS key2,
    value,
    String::Contains(value, "test") AS has_test
FROM other_table;
```

CREATE TRANSFER

Создает трансфер из топика в таблицу.

Синтаксис:

```
CREATE TRANSFER transfer_name
FROM topic_name TO table_name USING lambda
WITH (option = value[, ...])
```

- `transfer_name` — имя создаваемого трансфера.
- `topic_name` — имя топика, содержащего исходные сообщения для последующего преобразования и записи в таблицу.
- `table_name` — имя таблицы, в которую будут записываться данные.
- `lambda` — lambda-функция преобразования сообщений.
- `option` — опция команды:
  - `CONNECTION_STRING` — строка соединения с базой данных, содержащей топик. Указывается только если топик находится в другой базе YDB.
  - Настройки для аутентификации в базе топика одним из способов (обязательно, если топик находится в другой базе):
    - С помощью токена:
      - `TOKEN_SECRET_PATH` — секрет, содержащий токен.
    - С помощью логина и пароля:
      - `USER` — имя пользователя.
      - `PASSWORD_SECRET_PATH` — секрет, содержащий пароль.
    - С помощью сервисного агента:
      - `SERVICE_ACCOUNT_ID` — идентификатор сервисного аккаунта.
      - `INITIAL_TOKEN_SECRET_PATH` — секрет, содержащий токен от сервисного аккаунта. Используется для первоначальной инициализации.
  - `CONSUMER` — имя читателя топика-источника. Если имя задано, то в топике уже должен существовать читатель с указанным именем, и трансфер начнёт обрабатывать сообщения, начиная с первого незакоммиченного сообщения в топике. Если имя не задано, то читатель будет добавлен в топик автоматически, и трансфер начнёт обрабатывать сообщения, начиная с первого хранящегося сообщения в топике. Имя автоматически созданного читателя можно получить из описания экземпляра трансфера.
  - Параметры батчевания записи в таблицу позволяют настроить баланс между задержкой появления записей в таблице и ресурсами, требуемыми для работы трансфера. Параметры батчевания влияют на обработку каждой партиции топика независимо. К изменению параметров батчевания нужно подходить с осторожностью, так как их изменение может как улучшить скорость обработки потока сообщений, так и ухудшить её, и даже привести к отказу в обслуживании, если параметры будут подобраны неверно. Например, запись в таблицу маленькими по размеру батчами может привести к перегрузке таблицы и деградации скорости работы с ней, а слишком большой размер батча — к тому, что на сервере закончится вся доступная память.
    - `BATCH_SIZE_BYTES` — размер батча в байтах. По умолчанию — 8 МБ.
    - `FLUSH_INTERVAL` — периодичность записи в таблицу. По умолчанию — 60 секунд. Запись в таблицу будет осуществлена, даже если батч не достиг размера, заданного в параметре `BATCH_SIZE_BYTES`.

Разрешения

Для создания трансфера требуются следующие права:

- `CREATE TABLE` — для создания экземпляра трансфера;
- `ALTER SCHEMA` — для автоматического создания читателя топика (если применимо);
- `SELECT ROW` — для чтения сообщений из топика, содержащего исходные сообщения;
- `UPDATE ROW` — для обновления строк в таблице, в которую будут записываться данные.

Примеры

Создание экземпляра трансфера из топика `example_topic` в таблицу `example_table` текущей базы данных:

```
CREATE TABLE example_table (
    partition Uint32 NOT NULL,
    offset Uint64 NOT NULL,
    message Utf8,
    PRIMARY KEY (partition, offset)
);

CREATE TOPIC example_topic;

$transformation_lambda = ($msg) -> {
    return [
        <|
            partition: $msg._partition,
            offset: $msg._offset,
            message: CAST($msg._data AS Utf8)
        |>
    ];
};

CREATE TRANSFER example_transfer
    FROM example_topic TO example_table USING $transformation_lambda;
```

Создание экземпляра трансфера из топика `example_topic` базы данных `/Root/another_database` в таблицу `example_table` текущей базы данных. Перед созданием трансфера необходимо в текущей базе создать таблицу в которую будут записываться данные; в базе данных `/Root/another_database` создать топик, из которого будут обрабатываться сообщения:

Совет

Перед выполнением операции создайте секрет с аутентификационными данными для подключения или убедитесь в его существовании и наличии доступа к нему.

```
$transformation_lambda = ($msg) -> {
    return [
        <|
            partition: $msg._partition,
            offset: $msg._offset,
            message: CAST($msg._data AS Utf8)
        |>
    ];
};

CREATE TRANSFER example_transfer
    FROM example_topic TO example_table USING $transformation_lambda
WITH (
    CONNECTION_STRING = 'grpcs://example.com:2135/?database=/Root/another_database',
    TOKEN_SECRET_PATH = 'my_secret'
);
```

Создание экземпляра трансфера с явным указанием имени консьюмера `existing_consumer_of_topic`:

```
$transformation_lambda = ($msg) -> {
    return [
        <|
            partition: $msg._partition,
            offset: $msg._offset,
            message: CAST($msg._data AS Utf8)
        |>
    ];
};
CREATE TRANSFER example_transfer
    FROM example_topic TO example_table USING $transformation_lambda
WITH (
    CONSUMER = 'existing_consumer_of_topic'
);
```

Пример обработки сообщения в формате JSON

```
// example message:
// {
//   "update": {
//     "operation":"value_1"
//   },
//   "key": [
//     "id_1",
//     "2019-01-01T15:30:00.000000Z"
//   ]
// }

$transformation_lambda = ($msg) -> {
    $json = CAST($msg._data AS JSON);
    return [
        <|
            timestamp: CAST(Yson::ConvertToString($json.key[1]) AS Timestamp),
            object_id: CAST(Yson::ConvertToString($json.key[0]) AS Utf8),
            operation: CAST(Yson::ConvertToString($json.update.operation) AS Utf8)
        |>
    ];
};

CREATE TRANSFER example_transfer
    FROM example_topic TO example_table USING $transformation_lambda;
```

Создание экземпляра трансфера с явным указанием опции батчевания:

```
$transformation_lambda = ($msg) -> {
    return [
        <|
            partition: $msg._partition,
            offset: $msg._offset,
            message: CAST($msg._data AS Utf8)
        |>
    ];
};
CREATE TRANSFER example_transfer
    FROM example_topic TO example_table USING $transformation_lambda
WITH (
    BATCH_SIZE_BYTES = 1048576,
    FLUSH_INTERVAL = Interval('PT60S')
);
```

lambda-функция

Lambda-функция преобразования сообщений принимает один параметр со структурой, содержащей сообщение из топика, и возвращает список структур, соответствующих строкам таблицы для вставки.

Пример:

```
$lambda = ($msg) -> {
  return [
    <|
      column_1: $msg._create_timestamp,
      column_2: $msg._data
    |>
  ];
};
```

В этом примере:

- `$msg` — сообщение, полученное из топика.
- `column_1` и `column_2` — названия колонок таблицы.
- `$msg._create_timestamp` и `$msg._data` — значения, которые будут записаны в таблицу. Типы значений должны совпадать с типами колонок таблицы. Например, если `column_2` имеет в таблице тип `String`, то и тип `$msg._data` должен быть именно `String`.

У сообщения топика доступны следующие поля:

Атрибут	Тип значения	Описание
`_create_timestamp`	`Timestamp`	Время создания сообщения
`_data`	`String`	Тело сообщения
`_offset`	`Uint64`	Смещение сообщения
`_partition`	`Uint32`	Номер партиции сообщения
`_producer_id`	`String`	Идентификатор писателя
`_seq_no`	`Uint64`	Порядковый номер сообщения
`_write_timestamp`	`Timestamp`	Время записи сообщения

Тестирование lambda-функций

Для тестирования lambda-функции при её разработке можно в качестве сообщения топика передавать структуру с такими же полями, как будут передаваться в трансфере. Пример:

```
$lambda = ($msg) -> {
  return [
    <|
      offset: $msg._offset,
      data: $msg._data
    |>
  ];
};

$msg = <|
  _data: "value",
  _offset: CAST(1 AS Uint64),
  _partition: CAST(2 AS Uint32),
  _producer_id: "producer",
  _seq_no: CAST(3 AS Uint64)
|>;

SELECT $lambda($msg);
```

Если lambda-функция содержит сложную логику преобразования, то её можно выделить в отдельную lambda-функцию, что упростит тестирование.

```
$extract_value = ($data) -> {
  -- сложные преобразования
  return $data;
};

$lambda = ($msg) -> {
  return [
    <|
      column: $extract_value($msg._data)
    |>
  ];
};

-- Тестировать lambda-функцию extract_value можно так

SELECT $extract_value('преобразуемое значение');
```

См. также

- ALTER TRANSFER
- DROP TRANSFER
- Трансфер данных

CREATE VIEW

`CREATE VIEW` создаёт представление.

Представление - это обёртка над `SELECT` запросом, позволяющая обращаться к нему как к таблице, в которую сохранены актуальные результаты выполнения этого запроса. Представление не хранит никаких данных, кроме текста запроса. Чтение из представления аналогично чтению из подзапроса.

Синтаксис

```
CREATE VIEW [IF NOT EXISTS] <имя>
[ WITH ( <имя_параметра_представления> [= <значение_параметра_представления>] [, ... ] ) ]
AS <запрос>
```

Параметры

- `IF NOT EXISTS` - при использовании этой конструкции, выражение не возвращает ошибку, если представление с указанным именем уже существует.
- `имя` - имя создаваемого представления. Должно быть уникально. Допускается запись в виде абсолютного пути до представления.
- `запрос` - `SELECT` запрос, который выдаёт столбцы и строки представления.
- `WITH ( <имя_параметра_представления> [= <значение_параметра_представления>] [, ... ] )` позволяет задать значения параметров, определяющих поведение представления. Поддерживаются следующие опции:
  - `security_invoker` (Bool) - опция, при включении которой запрос, хранящийся в представлении, будет исполнен от имени пользователя, читающего из представления, а не от имени создателя представления.

Примечание

При выборе имени для представления учитывайте общие правила именования схемных объектов.

Замечания

`security_invoker` опция должна быть всегда выставлена в `TRUE`, потому что поведением по умолчанию для представления является исполнение сохранённого запроса от имени создателя представления, но эта возможность на данный момент не реализована в YDB.

Контекст исполнения запроса, хранящегося в представлении, отличается от контекста объемлющего запроса, читающего из представления. Внутренний контекст создаётся заново и ничего не наследует из объемлющего. Он не "видит" ранее определённые PRAGMA (например, TablePathPrefix), именованные выражения и прочее. Самое важное следствие из этого состоит в том, что названия таблиц (и, возможно, других представлений), на которые ссылается создаваемое представление, должны быть записаны в виде абсолютного пути. В приведенных ниже примерах видно, что пути до таблиц, из которых читает создаваемое представление, записаны в абсолютном виде: `/domain/database/path/to/underlying_table`. В следующих релизах эта недоработка будет исправлена.

Если вы хотите задать список интересующих вас колонок или, например, переименовать одну из них в выдаче при чтении из представления, то сделать это можно, сохранив в представление подходящий запрос:

```
CREATE VIEW view_with_a_renamed_column WITH (security_invoker = TRUE) AS
SELECT
    original_column_name AS custom_column_name
FROM `/domain/database/path/to/underlying_table`;
```

Интерпретация символа `*` для получения списка колонок при чтении из подлежащих таблиц происходит каждый раз при чтении из представления. Это означает, что список колонок в результате следующего запроса:

```
/*
CREATE VIEW view_with_an_asterisk WITH (security_invoker = TRUE) AS
SELECT
    *
FROM `/domain/database/path/to/underlying_table`;
*/

SELECT * FROM view_with_an_asterisk;
```

изменится при добавлении или удалении колонок из подлежащей таблицы.

Примеры

Создание представления, которое будет возвращать список современных сериалов:

```
CREATE VIEW recent_series WITH (security_invoker = TRUE) AS
SELECT
    *
FROM `/domain/database/path/to/series`
WHERE
    release_date > Date("2020-01-01");
```

Создание представления, которое будет возвращать названия первых эпизодов современных сериалов:

```
CREATE VIEW recent_series_first_episodes_titles WITH (security_invoker = TRUE) AS
SELECT
    episodes.title AS first_episode
FROM `/domain/database/path/to/recent_series`
    AS recent_series
```

```
JOIN `/domain/database/path/to/episodes`
    AS episodes
USING(series_id)
WHERE episodes.season_id = 1 AND episodes.episode_id = 1;
```

См. также

- ALTER VIEW
- DROP VIEW

CREATE RESOURCE POOL

`CREATE RESOURCE POOL` создаёт пул ресурсов.

Синтаксис

```
CREATE RESOURCE POOL <name>
WITH ( <parameter_name> [= <parameter_value>] [, ... ] )
```

- `name` - имя создаваемого пула ресурсов. Должно быть уникально. Не допускается запись в виде пути (т.е. не должно содержать `/`).
- `WITH ( <parameter_name> [= <parameter_value>] [, ... ] )` позволяет задать значения параметров, определяющих поведение пула ресурсов.

Параметры

- `CONCURRENT_QUERY_LIMIT` (Int32) — опциональное поле, задающее количество параллельно выполняющихся запросов в пуле ресурсов. Если значение `-1`, то ограничений нет. Значение по умолчанию: `-1`. Допустимые значения: .
- `QUEUE_SIZE` (Int32) — опциональное поле, определяющее размер очереди ожидания. Всего в системе может находиться не более чем запросов одновременно. Если значение `-1`, ограничений нет. Значение по умолчанию: `-1`. Допустимые значения: .
- `DATABASE_LOAD_CPU_THRESHOLD` (Int32) — опциональное поле, задающее порог загрузки CPU всей базы данных, после которого запросы не отправляются на выполнение и остаются в очереди. Если значение `-1`, ограничений нет. Значение по умолчанию: `-1`. Допустимые значения: .
- `TOTAL_CPU_LIMIT_PERCENT_PER_NODE` (Double) — опциональное поле, задающее процент доступного CPU, который могут использовать все запросы на узле в данном пуле ресурсов. Если значение `-1`, ограничений нет. Значение по умолчанию: `-1`. Допустимые значения: .

Замечания

Запросы всегда выполняются в каком-либо пуле ресурсов. По умолчанию все запросы отправляются в пул ресурсов `default`, который создаётся автоматически и не может быть удалён — он всегда присутствует в системе.

Если для параметра `CONCURRENT_QUERY_LIMIT` установить значение 0, то все запросы, отправленные в этот пул, будут немедленно завершены со статусом `PRECONDITION_FAILED`.

Разрешения

Требуется разрешение `CREATE TABLE` на директорию `.metadata/workload_manager/pools`, пример выдачи такого разрешения:

```
GRANT 'CREATE TABLE' ON `.metadata/workload_manager/pools` TO `user1@domain`;
```

```
CREATE RESOURCE POOL olap WITH (
    CONCURRENT_QUERY_LIMIT=20,
    QUEUE_SIZE=1000,
    DATABASE_LOAD_CPU_THRESHOLD=80,
    TOTAL_CPU_LIMIT_PERCENT_PER_NODE=70
)
```

В примере выше создаётся пул ресурсов со следующими ограничениями:

- Максимальное число параллельных запросов — 20.
- Максимальный размер очереди ожидания — 1000.
- При достижении загрузки базы данных в 80%, запросы перестают запускаться параллельно.
- Общее ограничение на доступный CPU для всех запросов в пуле на узле составляет 70%.

См. также

- Workload Manager — управление потреблением ресурсов
- ALTER RESOURCE POOL
- DROP RESOURCE POOL

CREATE RESOURCE POOL CLASSIFIER

`CREATE RESOURCE POOL CLASSIFIER` создаёт классификатор пулов ресурсов.

Синтаксис

```
CREATE RESOURCE POOL CLASSIFIER <name>
WITH ( <parameter_name> [= <parameter_value>] [, ... ] )
```

- `name` — имя создаваемого классификатора пула ресурсов. Должно быть уникальным. Имя не должно содержать символы, запрещённые для схемных объектов.
- `WITH ( <parameter_name> [= <parameter_value>] [, ... ] )` — позволяет задавать значения параметров, определяющих поведение классификатора пула ресурсов.

Общие параметры

- `RANK` (Int64) — опциональное поле, задающее порядок выбора классификатора пула ресурсов. Если значение не указано, берётся максимальный существующий `RANK` и к нему прибавляется 1000. Допустимые значения: уникальное число в диапазоне .
- `RESOURCE_POOL` (String) — имя пула ресурсов, в который направляются запросы, удовлетворяющие предикатам классификатора.
- `ACTION` (Enum) — действие, применяемое к запросу при срабатывании классификатора. Допустимое значение — `reject` : запрос отклоняется, пользователь получает ошибку вида `Request is rejected by classifier '<name>' (rank=<rank>)` .

> **Примечание**
>
> Параметры `RESOURCE_POOL` и `ACTION` взаимоисключающие: в классификаторе должен быть указан ровно один из них.

Параметры-предикаты

Предикат — условие, проверяемое для входящего запроса. Классификатор срабатывает, если выполнены **все** его предикаты (логическое **AND**). Для реализации логики **OR** необходимо создать несколько классификаторов с разными значениями `RANK` . Классификаторы обрабатываются в порядке возрастания `RANK` ; обработка останавливается на первом сработавшем — к запросу применяется его `ACTION` или он направляется в его `RESOURCE_POOL` .

Параметры-предикаты необязательны. Классификатор без предикатов срабатывает на любой запрос — это удобно для «закрывающего» классификатора с максимальным `RANK` , например, направить весь неклассифицированный трафик в конкретный пул (`RESOURCE_POOL`) или отклонить его (`ACTION='reject'`).

> **Важно**
>
> Будьте внимательны при создании «закрывающего» классификатора: все классификаторы, следующие за ним по `RANK` , никогда не сработают — обработка останавливается на первом совпадении, а «закрывающий» срабатывает всегда.

Список предикатов:

- `MEMBER_NAME` (String) — SID пользователя или группы, от имени которых поступил запрос. Подробнее — ниже.
- `HAS_PATH` (String) — путь к объекту YDB, к которому обращается запрос; поддерживает подстановочные знаки `*` и `?` . Подробнее — ниже.
- `HAS_APP_NAME` (String) — идентификатор клиентского приложения. Подробнее — ниже.
- `HAS_FULL_SCAN` (String) — путь к объекту, по которому ожидается полное сканирование; поддерживает подстановочные знаки `*` и `?` . Подробнее — ниже.
- `HAS_STREAM` (Bool) — признак стримингового запроса. Подробнее — ниже.

MEMBER_NAME

`MEMBER_NAME` сравнивается посимвольно с SID пользователя или любым SID группы из его токена аутентификации. Формат SID зависит от того, каким способом пользователь пришёл в систему.

- **Встроенные пользователи YDB (логин/пароль)** — SID совпадает с именем пользователя, без суффикса. Например, `user1` . Подробнее — Аутентификация по логину и паролю.
- **Облачные пользователи (Access Service)** — SID имеет вид `<subject_id>@as` , где `<subject_id>` — идентификатор пользователя в IAM. Суффикс задаётся параметром `access_service_domain` (по умолчанию `as`). Например, `ajeb89hv69nujke769fa@as` . Подробнее — Аутентификация с использованием стороннего IAM-провайдера.
- **LDAP** — SID имеет вид `<логин>@<домен>` , где домен задаётся параметром `ldap_authentication_domain` (по умолчанию `ldap`). Например, `user1@ldap` . Подробнее — Аутентификация с использованием LDAP-каталога.
- **Внешние провайдеры идентификации (OIDC)** — SID имеет вид `<логин>@<домен>` , где домен задаётся параметром `external_idp_authentication_domain` в конфигурации аутентификации (по умолчанию `sso`). Например, `user1@sso` .

В качестве значения можно указать как SID конкретного пользователя, так и SID группы. Ко всем аутентифицированным пользователям автоматически добавляется группа `all-users@well-known` — её удобно использовать, если нужно направить в пул запросы от всех аутентифицированных клиентов.

Пример. Направить запросы пользователя `user1@ldap` в пул `olap` :

```
CREATE RESOURCE POOL CLASSIFIER cl_user WITH (
    RANK=100,
    RESOURCE_POOL='olap',
```

```
    MEMBER_NAME='user1@ldap'
);
```

HAS_PATH

`HAS_PATH` сравнивает пути объектов YDB, к которым обращается запрос, с указанной маской. Маска поддерживает подстановочные знаки: `*` — любая последовательность символов, `?` — один любой символ. Предикат срабатывает, если хотя бы один объект в плане запроса соответствует маске.

Пример. Направить запросы к архивным таблицам в пул `pool_archive`:

```
CREATE RESOURCE POOL CLASSIFIER cl_archive WITH (
    RANK=700,
    RESOURCE_POOL='pool_archive',
    HAS_PATH='/Root/db/archive/*'
);
```

HAS_APP_NAME

`HAS_APP_NAME` сравнивает значение с идентификатором клиентского приложения. Значение передаётся клиентом в метаданных gRPC-запроса через заголовок `x-ydb-application-name`; SDK YDB предоставляют способ задать его на стороне клиента. Сравнение — на точное совпадение (без подстановочных знаков).

> **Важно**
>
> Значение `HAS_APP_NAME` задаётся клиентом и не аутентифицируется сервером — не используйте его как средство контроля доступа. Для эффективной изоляции сочетайте с `MEMBER_NAME` или направляйте нераспознанные запросы в sandbox-пул с жёсткими лимитами.

Установка идентификатора приложения в клиенте:

- **YDB UI** — фиксированное значение `ydb-ui`, задаётся YDB UI и не настраивается пользователем.
- **YDB CLI** — не поддерживается: идентификатор клиентского приложения в запросе не отправляется.
- **YDB C++ SDK** — на каждом запросе через параметр `Header` настроек `TRequestSettings`: `settings.Header({{ NYdb::YDB_APPLICATION_NAME, "my-app" }})`, где константа `YDB_APPLICATION_NAME` равна `x-ydb-application-name`.
- **YDB Go SDK** — на драйвере через опцию `WithApplicationName` в вызове `ydb.Open`.
- **YDB Java SDK** — на транспорте через метод `GrpcTransportBuilder.withApplicationName`.
- **YDB Python SDK** — отдельного параметра нет; значение задаётся на каждом запросе через дополнительный заголовок: `settings.with_header("x-ydb-application-name", "my-app")` (метод `BaseRequestSettings.with_header`).

Пример. Направить запросы от YDB UI в пул `pool_adhoc`:

```
CREATE RESOURCE POOL CLASSIFIER cl_adhoc_ui WITH (
    RANK=200,
    RESOURCE_POOL='pool_adhoc',
    HAS_APP_NAME='ydb-ui'
);
```

HAS_FULL_SCAN

`HAS_FULL_SCAN` определяет запросы, содержащие полное сканирование указанных объектов. Полное сканирование — чтение таблицы без ограничения по ключу или диапазону ключей. Аргумент — маска пути к объекту с поддержкой подстановочных знаков `*` и `?`; предикат срабатывает, если в плане запроса есть хотя бы один такой объект. Поддерживаются как строковые, так и колоночные таблицы. Объекты, к которым понятие полного сканирования неприменимо (например, топики), не учитываются.

Особенности определения полного сканирования:

- `LIMIT` **не отменяет полное сканирование.** Без условия по ключу формируется физический план, обрабатывающий всю таблицу целиком; `LIMIT` ограничивает только размер результата.
- **Вторичные индексы.** Полное сканирование индексной таблицы вторичного индекса засчитывается по её собственному пути. Например, у таблицы

  ```
  CREATE TABLE orders (
      Id Uint64 NOT NULL,
      Status Utf8,
      PRIMARY KEY (Id),
      INDEX by_status GLOBAL ON (Status)
  );
  ```

  индексная таблица имеет путь `/Root/orders/by_status/indexImplTable`. Запрос `SELECT * FROM orders VIEW by_status` выполняет полное сканирование именно индексной таблицы — основная `/Root/orders` не сканируется. Поэтому:

  - `HAS_FULL_SCAN='/Root/orders'` — **не сработает**;
  - `HAS_FULL_SCAN='/Root/orders/by_status/indexImplTable'` или `HAS_FULL_SCAN='/Root/orders/*'` — **сработает**.

Пример. Отклонять запросы, вызывающие полное сканирование архива заказов:

```
CREATE RESOURCE POOL CLASSIFIER cl_fullscan_reject WITH (
    RANK=100,
    ACTION='reject',
    HAS_FULL_SCAN='/Root/db/orders_archive/*'
);
```

HAS_STREAM

`HAS_STREAM` определяет, является ли запрос стриминговым — то есть выполняет длительное непрерывное чтение и/или запись из потоков данных. Допустимые значения:

- `true` — классификатор срабатывает на стриминговых запросах;
- `false` — классификатор срабатывает на нестриминговых запросах.

Пример. Направить стриминговые запросы в пул `pool_stream`:

```
CREATE RESOURCE POOL CLASSIFIER cl_stream WITH (
    RANK=500,
    RESOURCE_POOL='pool_stream',
    HAS_STREAM=true
);
```

Замечания

Если в DDL для создания классификатора пула ресурсов не указан `RANK`, то по умолчанию ему будет присвоено значение . Все значения `RANK` должны быть уникальными, чтобы обеспечить строго детерминированный порядок выбора пула ресурсов в случае конфликтующих условий. Такое поведение выбрано для возможности добавлять новые классификаторы пулов ресурсов между уже существующими.

Также возможно наличие классификатора, который ссылается на несуществующий пул ресурсов или к которому у пользователя нет доступа. В таком случае он пропускается.

С ограничениями на число классификаторов можно ознакомиться на странице ограничений.

Разрешения

Требуется разрешение `USE` на базу данных.

Пример выдачи такого разрешения:

```
GRANT 'USE' ON `/my_db` TO `user1@domain`;
```

Примеры

Ниже — сводный пример, комбинирующий несколько классификаторов и предикатов: отклонение полных сканов архивных таблиц, изоляция стриминговых запросов и выделение пула под интерактивные запросы админа из YDB UI.

Создание ресурсных пулов:

```
CREATE RESOURCE POOL pool_stream WITH (
    TOTAL_CPU_LIMIT_PERCENT_PER_NODE=20
);

CREATE RESOURCE POOL pool_adhoc_admin WITH (
    TOTAL_CPU_LIMIT_PERCENT_PER_NODE=10
);
```

Создание классификаторов:

```
-- Отклонять полные сканы архивных таблиц.
CREATE RESOURCE POOL CLASSIFIER cl_fullscan_reject WITH (
    RANK=100,
    ACTION='reject',
    HAS_FULL_SCAN='/Root/db/orders_archive/*'
);

-- Стриминговые запросы направлять в выделенный пул.
CREATE RESOURCE POOL CLASSIFIER cl_stream WITH (
    RANK=200,
    RESOURCE_POOL='pool_stream',
    HAS_STREAM=true
);

-- Запросы админа из YDB UI — в пул интерактивных запросов.
-- Условие AND: и MEMBER_NAME, и HAS_APP_NAME должны совпасть.
CREATE RESOURCE POOL CLASSIFIER cl_adhoc_admin WITH (
    RANK=300,
    RESOURCE_POOL='pool_adhoc_admin',
    MEMBER_NAME='admin',
    HAS_APP_NAME='ydb-ui'
);
```

Классификаторы обрабатываются в порядке возрастания `RANK`; для запроса срабатывает первый подходящий. Запрос, не удовлетворяющий ни одному классификатору, направляется в пул `default`.

См. также

- Workload Manager — управление потреблением ресурсов
- ALTER RESOURCE POOL CLASSIFIER
- DROP RESOURCE POOL CLASSIFIER

CREATE TOPIC

С помощью оператора `CREATE TOPIC` можно создать топик, а также читателей для него.

Общий вид команды:

```
CREATE TOPIC topic_path (
    CONSUMER consumer_name [WITH (consumer_option = value[, ...])]
    ) WITH (topic_option = value[, ...]);
```

- `consumer_option` — параметр читателя;
- `topic_option` — параметр топика.

Все параметры команды, кроме `topic_path` не обязательны. По умолчанию топик создается без читателей. Все не указанные явно параметры также выставляются по умолчанию (и для топика, и для читателя).

Параметры читателя:

- `type` — тип читателя. Возможные значения: `STREAMING` и `SHARED`. Значение по умолчанию `STREAMING`.
- `important` — признак важного читателя. Данные из топика не будут удалены, пока их не обработают все важные читатели. Тип значения — `boolean`, значение по умолчанию — `false`.
- `availability_period` — определяет время доступности сообщений для читателя. Опция позволяет продлить время хранения сообщений в топике сверх retention_period, вплоть до `availability_period`, если читатель не подтверждает их обработку. Тип значения — `Interval`. Несовместим с параметром `important`. Значение по умолчанию отсутствует.
- `read_from` — определяет момент записи сообщений, начиная с которого читатель будет получать данные. Сообщения, записанные ранее этого момента, читатель не получит. Тип значения — `Datetime`, `Timestamp` или `integer` (Unix timestamp в виде числа). Значение по умолчанию — `0` (чтение с самого раннего доступного в топике момента времени).
- `supported_codecs` — список кодеков, поддерживаемых читателем.

Параметры читателя, доступные только для разделяемого (общего) читателя:

- `keep_messages_order` — сохраняет порядок чтения сообщений. Если значение равно `true`, гарантируется порядок обработки сообщений в рамках одной группы сообщений. Значение по умолчанию `false`.
- `default_processing_timeout` — время обработки сообщения. Если за это время обработка сообщения не будет подтверждена и время обработки не будет продлено, сообщение вернется в очередь и будет передано на повторную обработку. Значение по умолчанию `Interval('PT30S')`.
- `max_processing_attempts` — максимальное количество попыток обработки одного сообщения (тип значения — `integer`). Опция поддерживается только при `dead_letter_policy = move` или `dead_letter_policy = delete`. Значение по умолчанию — `1000`.
- `dead_letter_policy` — действие, которое нужно выполнить с сообщением, если все попытки его обработки завершились неудачей (тип значения — `String`). Возможные значения: `delete`, `move`, `none`. Значение по умолчанию — `none`.
- `dead_letter_queue` — имя топика DLQ (тип значения — `String`). Обязателен при `dead_letter_policy = move` и не поддерживается при `dead_letter_policy = none` / `delete`.

Параметры топика

- `metering_mode` — способ метеринга ресурсов (`RESERVED_CAPACITY` - по выделенным ресурсам или `REQUEST_UNITS` - по фактическому использованию). Актуально для топиков в serverless базах данных. Тип значения - `String`.
- `min_active_partitions` — минимальное количество активных партиций топика. Автопартиционирование не будет уменьшать количество активных партиций ниже этого количества. Тип значения — `integer`, значение по умолчанию — `1`.
- `max_active_partitions` — максимальное количество активных партиций топика. Автопартиционирование не будет увеличивать количество активных партиций выше этого количества. Тип значения — `integer`, по умолчанию равно `min_active_partitions`.
- `retention_period` — время хранения данных в топике. Тип значения — `Interval`, значение по умолчанию — `24h`.
- `retention_storage_mb` — ограничение на максимальное место на диске, занимаемое данными топика. При превышении этого значения старые данные будут удаляться, как по retention. При включенном автоматическом партиционировании потребляемое место может превышать установленное значение. Тип значения — `integer`, значение по умолчанию — `0` (не ограничено).
- `partition_write_burst_bytes` — размер запаса квоты на запись в партицию на случай всплесков записи. При выставлении в `0` фактическое значение write_burst принимается равным значению квоты (что позволяет всплески записи длительностью до 1 секунды). Тип значения — `integer`, значение по умолчанию: `0`.
- `partition_write_speed_bytes_per_second` — максимальная разрешенная скорость записи в 1 партицию. Если поток записи в партицию превысит это значение, запись будет квотироваться. Тип значения — `integer`, значение по умолчанию — `2097152` (2 МБ).
- `auto_partitioning_strategy` — режим автопартиционирования.
  Допустимые значения: `disabled`, `paused`, `scale_up`, значение по умолчанию — `disabled`.
- `auto_partitioning_up_utilization_percent` — определяет порог загрузки партиции в процентах от максимальной скорости записи, при достижении которого будет инициировано автоматическое **увеличение** числа партиций. Тип значения — `integer`, значение по умолчанию — `90`.
- `auto_partitioning_stabilization_window` — определяет временной интервал, в течение которого уровень нагрузки должен оставаться выше установленного порога (`auto_partitioning_up_utilization_percent`), прежде чем будет выполнено автоматическое увеличение количества партиций. Тип значения — `Interval`, значение по умолчанию — `5m`.

Важно

Значение `max_active_partitions`, превышающее `min_active_partitions`, само по себе не включает автопартиционирование. Чтобы использовать разные минимальную и максимальную границы, дополнительно задайте `auto_partitioning_strategy` равным `scale_up`. При стратегии `disabled`, используемой по умолчанию, число активных партиций топика фиксировано.

- `supported_codecs` — список кодеков, поддерживаемых топиком. Тип значения — `String`.

Примечание

При выборе имени для топика учитывайте общие правила именования схемных объектов.

Следующая команда создаст топик без читателей с настройками по умолчанию:

```
CREATE TOPIC `my_topic`;
```

Чтобы создать топик с важным читателем и временем хранения данных 1 сутки, выполните команду:

```
CREATE TOPIC `my_topic` (
    CONSUMER my_consumer WITH (important = true)
) WITH (
    retention_period = Interval('P1D')
);
```

Чтобы создать топик с временем хранения данных 1 сутки и двумя читателями, для одного из которых данные могут по необходимости храниться до 2-х суток, выполните команду:

```
CREATE TOPIC `my_topic` (
    CONSUMER my_consumer1,
    CONSUMER my_consumer2 WITH (availability_period = Interval('P2D'))
) WITH (
    retention_period = Interval('P1D')
);
```

Чтобы создать топик с разделяемым (общим) читателем, выполните команду:

```
CREATE TOPIC `my_topic` (
    CONSUMER my_consumer WITH (
        type = 'shared',
        keep_messages_order = false,
        default_processing_timeout = Interval('PT30S'),
        max_processing_attempts = 3,
        dead_letter_policy = 'move',
        dead_letter_queue = 'my_dlq_topic'
    )
);
```

CREATE USER

Создает пользователя с указанным именем и паролем.

Синтаксис:

```
CREATE USER user_name [option]
```

- `user_name` — имя пользователя. Может содержать строчные буквы латинского алфавита и цифры.
- `option` — опция команды:
  - `PASSWORD 'password'` — создает пользователя с паролем `password`.
  - `PASSWORD NULL` — создает пользователя с пустым паролем (по умолчанию).
  - `NOLOGIN` - запрет на логин пользователя (блокировка пользователя).
  - `LOGIN` - разрешение на логин пользователя (по умолчанию).

Примечание

Область действия команд `CREATE USER`, `ALTER USER`, `DROP USER` не распространяется на внешние каталоги пользователей.
Учитывайте это, если к YDB подключаются пользователи со сторонней аутентификацией (например, LDAP).
Например, команда `CREATE USER` не создаст пользователя в LDAP-каталоге.
Подробнее про взаимодействие YDB с LDAP-каталогом.

Встроенные пользователи

YDB может иметь набор групп и пользователей уже с момента начального развёртывания.

Подробнее см. Первичные настройки безопасности и Начальная настройка безопасности кластера.

Такие пользователи и группы ничем не отличаются от пользователей и групп, созданных позже.

COMMIT

По умолчанию весь YQL запрос выполняется в рамках одной транзакции и независимые его части внутри выполняются по возможности параллельно.
С помощью ключевого слова `COMMIT;` можно добавить барьер в процесс выполнения, чтобы отложить выполнение идущих следом выражений до тех пор, пока не выполнятся все предшествующие.

Чтобы коммит выполнялся аналогичным образом автоматически после каждого выражения в запросе, можно использовать `PRAGMA autocommit;`.

Примеры

```
INSERT INTO result1 SELECT * FROM my_table;
INSERT INTO result2 SELECT * FROM my_table;
COMMIT;
-- В result2 уже будет содержимое SELECT со второй строки:
INSERT INTO result3 SELECT * FROM result2;
```

DECLARE

Объявляет типизированное именованное выражение, значение для которого будет передано параметром отдельно от текста запроса. Параметризация дает возможность отделить разработку аналитического решения задачи от её запуска с различными вариантами входных значений.

Для транзакционной нагрузки использование параметров позволяет избежать перекомпиляции запросов при повторении однотипных вызовов, снижая потребление ресурсов сервера, и исключая время компиляции из общего времени исполнения запроса.

Передача параметров поддерживается в SDK, CLI, и в графических интерфейсах.

Синтаксис

```
DECLARE $named-node AS data_type;
```

1. Ключевое слово `DECLARE`.
2. `$named-node` — имя, по которому можно будет обращаться к переданному значению, должно начинаться с символа `$`.
3. Ключевое слово `AS`.
4. `data_type` — тип данных в виде строки в принятом формате.

Допустимы только сериализуемые типы данных:

- Примитивные типы.
- Опциональные типы.
- Контейнеры за исключением `Stream<Type>`.
- `Void` и `Null` — поддерживаемые специальные типы.

Пример

```
DECLARE $x AS String;
DECLARE $y AS String?;
DECLARE $z AS List<String>;

SELECT $x, $y, $z;
```

Примеры передачи параметров для DECLARE можно найти здесь.

DELETE FROM

Важно

В настоящее время одновременное использование колоночных и строковых таблиц поддерживается в транзакциях, в которых данные только читаются, но не изменяются. Поддержка транзакций с возможностью модификации данных при одновременном использовании строковых и колоночных таблиц находится в разработке.

Если попытаться выполнить операцию записи в транзакции, в которой задействованы и колоночные, и строковые таблицы, транзакция завершится с ошибкой:

```
Write transactions that use both row-oriented and column-oriented tables are disabled at current t
ime
```

Удаляет строки из таблицы, подходящие под условия, заданные в `WHERE`.

Пример

```
DELETE FROM my_table
WHERE Key1 == 1 AND Key2 >= "One";
```

DELETE FROM ... ON

Используется для удаления данных на основе результатов подзапроса. Набор колонок, возвращаемых подзапросом, должен быть подмножеством колонок обновляемой таблицы, и в составе возвращаемых колонок обязательно должны присутствовать все колонки первичного ключа таблицы. Типы данных возвращаемых подзапросом колонок должны совпадать с типами данных соответствующих колонок таблицы.

Для поиска удаляемых из таблицы записей используется значение первичного ключа. Присутствие других (неключевых) колонок таблицы в составе выходных колонок подзапроса не влияет на результаты операции удаления.

Пример

```
$to_delete = (
    SELECT Key, SubKey FROM my_table WHERE Value = "ToDelete" LIMIT 100
);

DELETE FROM my_table ON
SELECT * FROM $to_delete;
```

DELETE FROM ... RETURNING

Используется для удаления строк и одновременного возврата значений из них. Это позволяет получить информацию об удаляемых записях за один запрос, избавляя от необходимости выполнять предварительный SELECT.

Примеры

- Возврат всех значений удаленных строк

```
DELETE FROM orders
WHERE status = 'cancelled'
RETURNING *;
```

- Возврат конкретных столбцов

```
DELETE FROM orders
WHERE status = 'cancelled'
RETURNING order_id, order_date;
```

См. также

- BATCH DELETE

DISCARD

Вычисляет `SELECT`, но не возвращает результат ни в клиент, ни в таблицу.

Полезно использовать в сочетании с `Ensure` для проверки выполнения пользовательских условий на финальный результат вычислений.

Запрос с `DISCARD` выполняется полностью — со всеми фильтрами, агрегациями и проверками `Ensure`, — но результирующий набор (result set) не возвращается клиенту. Данные при этом не пересылаются по сети: клиент получает только статус выполнения запроса. На больших выборках это заметно экономит трафик и память.

В запросе из нескольких операторов `DISCARD` действует на отдельный оператор:

```
SELECT 1;          -- вернётся
DISCARD SELECT 2;  -- выполнится, но в ответ не попадёт
SELECT 3;          -- вернётся
```

Клиент получит два результирующих набора — от первого и третьего оператора.

`DISCARD` применяется только к оператору целиком, использовать его внутри выражений нельзя. В частности, недопустимо:

- в подзапросах — `SELECT * FROM (DISCARD SELECT 1)`;
- в `WHERE ... IN (...)` — `SELECT * FROM my_table WHERE Key IN (DISCARD SELECT 1)`;
- в операндах `UNION ALL` — `SELECT 1 UNION ALL DISCARD SELECT 2`.

> **Примечание**
>
> `DISCARD` поддерживается при выполнении запросов через сервис исполнения запросов (Query Service) начиная с версии YDB 26.2. При выполнении через устаревшие интерфейсы (Table Service, скан-запросы) `DISCARD` игнорируется, и результат возвращается клиенту — поведение сохранено для обратной совместимости.

Примеры

```
DISCARD SELECT 1;
```

```
DISCARD SELECT Ensure(
    Data,
    Data < 1000000,
    "Value too big"
) FROM `result_table`;
```

Если условие `Ensure` нарушено, запрос завершится ошибкой. Если всё в порядке — запрос отработает успешно, а клиент не получит результирующий набор, который иначе пришлось бы выкачивать целиком.

DROP ASYNC REPLICATION

Вызов `DROP ASYNC REPLICATION` удаляет экземпляр асинхронной репликации. Вместе с экземпляром асинхронной репликации удаляются:

- автоматически созданные потоки изменений;
- объекты-реплики (опционально).

Синтаксис

```
DROP ASYNC REPLICATION <name> [CASCADE]
```

где:

- `name` — имя экземпляра асинхронной репликации.
- `CASCADE` — каскадное удаление объектов-реплик, созданных в рамках данного экземпляра асинхронной репликации.

Примеры

Рассмотрим примеры удаления экземпляра асинхронной репликации, созданного следующим запросом:

```
CREATE ASYNC REPLICATION my_replication
FOR original_table AS replica_table
WITH (
    CONNECTION_STRING = 'grpcs://example.com:2135/?database=/Root/another_database',
    TOKEN_SECRET_PATH = 'my_secret'
);
```

Удаление экземпляра асинхронной репликации и автоматически созданного потока изменений в таблице `original_table`, таблица `replica_table` остается:

```
DROP ASYNC REPLICATION my_replication;
```

Удаление экземпляра асинхронной репликации, автоматически созданного потока изменений в таблице `original_table` и таблицы `replica_table`:

```
DROP ASYNC REPLICATION my_replication CASCADE;
```

См. также

- CREATE ASYNC REPLICATION
- ALTER ASYNC REPLICATION

DROP BACKUP COLLECTION

Выражение `DROP BACKUP COLLECTION` удаляет коллекцию резервных копий и все содержащиеся в ней резервные копии.

```
DROP BACKUP COLLECTION collection_name;
```

Параметры

- `collection_name` — имя удаляемой коллекции резервных копий.

> **Важно**
>
> Эта операция безвозвратно удаляет коллекцию резервных копий и все содержащиеся в ней резервные копии из кластера. Это действие не может быть отменено.

> **Примечание**
>
> Удаление коллекции резервных копий затрагивает только данные в кластере. Резервные копии, ранее экспортированные во внешнее хранилище (S3 или файловую систему), не затрагиваются и остаются доступными для восстановления.

Примеры

Удаление коллекции резервных копий:

```
DROP BACKUP COLLECTION old_backups;
```

См. также

- Коллекции резервных копий.
- CREATE BACKUP COLLECTION.
- BACKUP.
- RESTORE.

DROP GROUP

Команда `DROP GROUP` удаляет указанную группу. Для одного оператора вы можете указать несколько групп.

Синтаксис:

```
DROP GROUP [ IF EXISTS ] group_name [, ...]
```

- `IF EXISTS` — не выводить ошибку, если группа не существует.
- `group_name` — имя группы, которую нужно удалить. Может содержать только строчные буквы латинского алфавита (a–z), цифры (0–9). Имя не должно быть пустым.

Примеры

Успешно выполненные команды

Удаление группы:

```
DROP GROUP group1;
```

```
DROP GROUP IF EXISTS group1;
```

Обработка ошибок

Корректно обработанные ошибки:

Ситуация	Сообщение об ошибке
Удаление несуществующей группы	Group not found

Встроенные группы

YDB может иметь набор групп и пользователей уже с момента начального развёртывания.

Подробнее см. Первичные настройки безопасности и Начальная настройка безопасности кластера.

Такие пользователи и группы ничем не отличаются от пользователей и групп, созданных позже.

См. также

- CREATE GROUP
- ALTER GROUP

DROP EXTERNAL DATA SOURCE

Удаляет указанный внешний источник данных.

Если внешнего источника данных с таким именем не существует, возвращается ошибка.

Пример

```
DROP EXTERNAL DATA SOURCE my_external_data_source;
```

DROP EXTERNAL TABLE

Удаляет указанную внешнюю таблицу.

Если внешней таблицы с таким именем не существует, возвращается ошибка.

Пример

```
DROP EXTERNAL TABLE my_table;
```

DROP OBJECT (TYPE SECRET)

> **Внимание**
>
> **Данная команда устарела** и будет удалена в будущих версиях YDB. Рекомендуемый синтаксис работы с секретами описан в разделе Секреты.

Удаляет указанный секрет.

Если секрета с таким именем не существует, возвращается ошибка.

Пример

```
DROP OBJECT my_secret (TYPE SECRET);
```

DROP OBJECT (TYPE SECRET_ACCESS)

Внимание

Данная команда устарела и будет удалена в будущих версиях YDB. Рекомендуемый синтаксис работы с секретами описан в разделе Секреты.

Удаляет указанное правило доступа к секрету.

Если правила с таким именем не существует, возвращается ошибка.

Пример

```
DROP OBJECT (TYPE SECRET_ACCESS) `MySecretName:another_user`;
```

DROP STREAMING QUERY

`DROP STREAMING QUERY` удаляет потоковый запрос.

Синтаксис

```
DROP STREAMING QUERY [IF EXISTS] <query_name>
```

Параметры

- `IF EXISTS` — не выводить ошибку, если потокового запроса не существует.
- `query_name` — имя потокового запроса, подлежащего удалению.

Разрешения

Требуется разрешение `REMOVE SCHEMA` на потоковый запрос, пример выдачи такого разрешения для запроса `my_streaming_query`:

```
GRANT REMOVE SCHEMA ON my_streaming_query TO `user@domain`
```

Примеры

Удаление запроса `my_streaming_query`:

```
DROP STREAMING QUERY my_streaming_query
```

См. также

- Потоковые запросы
- CREATE STREAMING QUERY
- ALTER STREAMING QUERY

DROP SECRET

Команда `DROP SECRET` удаляет существующий секрет.

Синтаксис:

```
DROP SECRET [IF EXISTS] secret_name
```

- `IF EXISTS` — команда не возвращает ошибку, если секрет не существует; в этом случае она ничего не делает.
- `secret_name` — имя удаляемого секрета.

Разрешения

Для удаления секрета требуются права `REMOVE SCHEMA` и `ALTER SCHEMA`.

Примеры

Удалить секрет с именем `secret_name`:

```
DROP SECRET secret_name;
```

Удалить секрет с именем `secret_name`, только если он существует; если он не существует, команда ничего не делает:

```
DROP SECRET IF EXISTS secret_name;
```

См. также

- CREATE SECRET
- ALTER SECRET

DROP TABLE

Удаляет указанную таблицу.

Если таблицы с таким именем не существует, возвращается ошибка.

Примеры

```
DROP TABLE my_table;
```

DROP TRANSFER

Вызов `DROP TRANSFER` удаляет экземпляр трансфера. Вместе с экземпляром трансфера удалится и читатель, если он был создан автоматически при создании трансфера. Попытки удаления читателя будут продолжаться до его успешного удаления.

Вызов `DROP TRANSFER` не удаляет таблицу, в которую записываются данные, и топик, из которого данные читаются.

Синтаксис

```
DROP TRANSFER <name>
```

где:

- `name` — имя экземпляра трансфера.

Разрешения

Для удаления трансфера требуются следующие права:

- `REMOVE SCHEMA` — для удаления экземпляра трансфера;
- `ALTER SCHEMA` — для удаления автоматически созданного читателя топика (если применимо).

Примеры

Следующий запрос удаляет трансфер с именем `my_transfer`:

```
DROP TRANSFER my_transfer;
```

См. также

- CREATE TRANSFER
- ALTER TRANSFER
- Трансфер данных

DROP VIEW

`DROP VIEW` удаляет представление.

Синтаксис

```
DROP VIEW [IF EXISTS] <имя>
```

Параметры

- `IF EXISTS` - при использовании этой конструкции, выражение не возвращает ошибку, если представления с указанным именем не существует.
- `имя` - имя представления, подлежащего удалению.

Примеры

Следующая команда удалит представление со списком современных сериалов:

```
DROP VIEW recent_series;
```

См. также

- CREATE VIEW
- ALTER VIEW

DROP RESOURCE POOL

`DROP RESOURCE POOL` удаляет пул ресурсов.

Синтаксис

```
DROP RESOURCE POOL <name>
```

Параметры

- `name` - имя пула ресурсов, подлежащего удалению.

Разрешения

Требуется разрешение `REMOVE SCHEMA` до пула в директории `.metadata/workload_manager/pools`, пример выдачи такого разрешения:

```
GRANT 'REMOVE SCHEMA`' ON `.metadata/workload_manager/pools` TO `user1@domain`;
```

Примеры

Следующая команда удалит пул ресурсов с именем "olap":

```
DROP RESOURCE POOL olap;
```

См. также

- Workload Manager — управление потреблением ресурсов
- CREATE RESOURCE POOL
- ALTER RESOURCE POOL

DROP RESOURCE POOL CLASSIFIER

`DROP RESOURCE POOL CLASSIFIER` удаляет классификатор пулов ресурсов.

Синтаксис

```
DROP RESOURCE POOL CLASSIFIER <name>
```

Параметры

- `name` - имя классификатора пула ресурсов, подлежащего удалению.

Разрешения

Требуется разрешение `ALL` на базу данных, пример выдачи такого разрешения:

```
GRANT 'ALL' ON `/my_db` TO `user1@domain`;
```

Примеры

Следующая команда удалит классификатор пула ресурсов с именем "olap_classifier":

```
DROP RESOURCE POOL CLASSIFIER olap_classifier;
```

См. также

- Workload Manager — управление потреблением ресурсов
- CREATE RESOURCE POOL CLASSIFIER
- ALTER RESOURCE POOL CLASSIFIER

DROP TOPIC

С помощью оператора `DROP TOPIC` можно удалить топик.

Синтаксис

```
DROP TOPIC topic_path;
```

Примеры

Следующая команда удалит топик с именем `my_topic`:

```
DROP TOPIC `my_topic`;
```

См. также

- CREATE TOPIC
- ALTER TOPIC

DROP USER

Удаляет указанного пользователя. Для одного оператора вы можете указать несколько пользователей.

Синтаксис:

```
DROP USER [ IF EXISTS ] user_name [, ...]
```

- `IF EXISTS` — не выводить ошибку, если пользователь не существует.
- `user_name` — имя удаляемого пользователя. Поддерживается возможность задать список пользователей через запятую, например: `DROP USER user1, user2, user3;`.

Встроенные пользователи

YDB может иметь набор групп и пользователей уже с момента начального развёртывания.

Подробнее см. Первичные настройки безопасности и Начальная настройка безопасности кластера.

Такие пользователи и группы ничем не отличаются от пользователей и групп, созданных позже.

GRANT

Команда `GRANT` позволяет установить права доступа к объектам схемы для пользователя или группы пользователей.

Синтаксис:

```
GRANT {{permission_name} [, ...] | ALL [PRIVILEGES]} ON {path_to_scheme_object [, ...]} TO {role_name [, ...]} [WITH GRANT OPTION]
```

- `permission_name` - имя права доступа к объектам схемы, которое нужно назначить.
- `path_to_scheme_object` - путь до объекта схемы, на который выдаются права.
- `role_name` - имя пользователя или группы, для которого выдаются права на объект схемы.

`WITH GRANT OPTION` - использование этой конструкции наделяет пользователя или группу пользователей правом управлять правами доступа - назначать или отзывать определенные права. Конструкция имеет функциональность аналогичную выдаче права `"ydb.access.grant"` или `GRANT`.
Субъект, обладающий правом `ydb.access.grant`, не может выдавать права шире, чем обладает сам на объекте доступа `path_to_scheme_object`.

Права доступа

В качестве имён прав доступа можно использовать имена YDB прав или соответствующие им ключевые слова YQL.
В таблице ниже перечислены возможные имена прав.

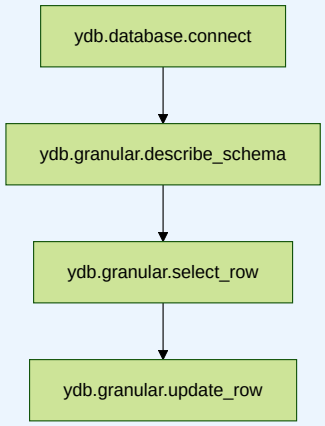
YDB право	Ключевое слово YQL	Описание
Права уровня баз данных		
`ydb.database.connect`	`CONNECT`	Право подключаться к базе данных
`ydb.database.create`	`CREATE`	Право создавать новые базы данных в кластере
`ydb.database.drop`	`DROP`	Право удалять базы данных в кластере
Элементарные права на объекты базы данных		
`ydb.granular.select_row`	`SELECT ROW`	Право читать строки из таблицы (select), читать сообщения из топиков, использовать значения секретов
`ydb.granular.update_row`	`UPDATE ROW`	Право обновлять строки в таблице (insert, update, upsert, replace), писать сообщения в топики. На корне базы также разрешает завершать чужие сессии через KILL SESSION при наличии `CONNECT`
`ydb.granular.erase_row`	`ERASE ROW`	Право удалять строки из таблицы (delete)
`ydb.granular.create_directory`	`CREATE DIRECTORY`	Право создавать и удалять директории, в том числе существующие и вложенные
`ydb.granular.create_table`	`CREATE TABLE`	Право создавать таблицы (в том числе индексные, внешние, колоночные), представления, последовательности
`ydb.granular.create_queue`	`CREATE QUEUE`	Право создавать топики
`ydb.granular.remove_schema`	`REMOVE SCHEMA`	Право удалять объекты (директории, таблицы, топики), которые были созданы посредством использования прав
`ydb.granular.describe_schema`	`DESCRIBE SCHEMA`	Право просмотра имеющихся прав доступа (ACL) на объект доступа, просмотра описания объектов доступа (директории, таблицы, топики)
`ydb.granular.alter_schema`	`ALTER SCHEMA`	Право изменять объекты доступа (директории, таблицы, топики), в том числе права пользователей на объекты доступа
Дополнительные флаги		
`ydb.access.grant`	`GRANT`	Право предоставлять или отзывать права у других пользователей в объёме, не превышающем текущий объём прав пользователя на объекте доступа
Права, основанные на других правах		
`ydb.tables.modify`	`MODIFY TABLES`	`ydb.granular.update_row` + `ydb.granular.erase_row`
`ydb.tables.read`	`SELECT TABLES`	Синоним `ydb.granular.select_row`
`ydb.generic.list`	`LIST`	Синоним `ydb.granular.describe_schema`
`ydb.generic.read`	`SELECT`	`ydb.granular.select_row` + `ydb.generic.list`
`ydb.generic.write`	`INSERT`	`ydb.granular.update_row` + `ydb.granular.erase_row` + `ydb.granular.create_directory` + `ydb.granular.create_table` + `ydb.granular.create_queue` + `ydb.granular.remove_schema` + `ydb.granular.alter_schema`
`ydb.generic.use_legacy`	`USE LEGACY`	`ydb.generic.read` + `ydb.generic.write` + `ydb.access.grant`

`ydb.generic.use`	`USE`	`ydb.generic.use_legacy` + `ydb.database.connect`
`ydb.generic.manage`	`MANAGE`	`ydb.database.create` + `ydb.database.drop`
`ydb.generic.full_legacy`	`FULL LEGACY`	`ydb.generic.use_legacy` + `ydb.generic.manage`
`ydb.generic.full`	`FULL`	`ydb.generic.use` + `ydb.generic.manage`

- `ALL [PRIVILEGES]` - используется для указания всех возможных прав на объекты схемы для пользователей или групп. `PRIVILEGES` является необязательным ключевым словом, необходимым для совместимости с SQL стандартом.

Примечание

Права `ydb.database.connect`, `ydb.granular.describe_schema`, `ydb.granular.select_row`, `ydb.granular.update_row` необходимо рассматривать как слои прав.



Например, для изменения строк необходимо не только право `ydb.granular.update_row`, но и все вышележащие права.

Примеры

- Назначить право `ydb.generic.read` на таблицу `/shop_db/orders` для пользователя `user1`:

```
GRANT 'ydb.generic.read' ON `/shop_db/orders` TO user1;
```

Та же команда, с использованием ключевого слова

```
GRANT SELECT ON `/shop_db/orders` TO user1;
```

- Назначить права `ydb.database.connect`, `ydb.generic.list` на корень базы `/shop_db` для пользователя `user2` и группы `group1`:

```
GRANT LIST, CONNECT ON `/shop_db` TO user2, group1;
```

- Назначить право `ydb.generic.use` на таблицы `/shop_db/orders` и `/shop_db/sellers` для пользователей `user1@domain`, `user2@domain`:

```
GRANT 'ydb.generic.use' ON `/shop_db/orders`, `/shop_db/sellers` TO `user1@domain`, `user2@domain`;
```

- Назначить все права на таблицу `/shop_db/sellers` для пользователя `admin_user`:

```
GRANT ALL ON `/shop_db/sellers` TO admin_user;
```

INSERT INTO

Важно

В настоящее время одновременное использование колоночных и строковых таблиц поддерживается в транзакциях, в которых данные только читаются, но не изменяются. Поддержка транзакций с возможностью модификации данных при одновременном использовании строковых и колоночных таблиц находится в разработке.

Если попытаться выполнить операцию записи в транзакции, в которой задействованы и колоночные, и строковые таблицы, транзакция завершится с ошибкой:

```
Write transactions that use both row-oriented and column-oriented tables are disabled at current t
ime
```

Добавляет строки в таблицу. При попытке вставить в таблицу строку с уже существующим значением первичного ключа операция завершится ошибкой с кодом `PRECONDITION_FAILED` и текстом `Operation aborted due to constraint violation: insert_pk`.

`INSERT INTO` позволяет выполнять следующие операции:

- Добавление константных значений с помощью `VALUES`.

```
INSERT INTO my_table (Key1, Key2, Value1, Value2)
VALUES (345987,'ydb', 'Яблочный край', 1414);
COMMIT;
```

```
INSERT INTO my_table (key, value)
VALUES ("foo", 1), ("bar", 2);
```

- Сохранение результата выборки `SELECT`.

```
INSERT INTO my_table
SELECT Key AS Key1, "Empty" AS Key2, Value AS Value1
FROM my_table1;
```

При работе с внешними файловыми источниками данных можно дополнительно указывать ряд параметров:

- `FORMAT` - формат хранимых данных в файловых хранилищах в федеративных запросах. Допустимые значения: `csv_with_names`, `tsv_with_names`, `json_list`, `json_each_row`, `json_as_string`, `parquet`, `raw`.
- `COMPRESSION` - формат сжатия файлов в файловых хранилищах в федеративных запросах. Допустимые значения: gzip, zstd, lz4, brotli, bzip2, xz.
- `PARTITIONED_BY` - список колонок партиционирования данных в файловых хранилищах в федеративных запросах. Содержит список колонок в порядке их размещения в файловом хранилище.
- `projection.enabled` - флаг включения расширенного партиционирования данных. Допустимые значения: `true`, `false`.
- `projection.<field_name>.type` - тип поля расширенного партиционирования данных. Допустимые значения: `integer`, `enum`, `date`.
- `projection.<field_name>.<options>` - расширенные свойства поля расширенного партиционирования данных.

Пример

```
INSERT INTO `connection`.`test/`
WITH
(
  FORMAT = "csv_with_names"
)
SELECT
    "value" AS value, "name" AS name
```

Где:

- `connection` — название соединения с S3 (Yandex Object Storage).
- `test/` — путь внутри бакета, куда будут записаны данные. При записи создаются файлы со случайными именами.

INSERT INTO ... RETURNING

Используется для вставки строк и одновременного возврата значений из них. Это позволяет получить информацию о вставленных записях за один запрос, избавляя от необходимости выполнять последующий SELECT.

Примеры

- Возврат всех значений вставленной строки

```
INSERT INTO some_table (id, year, color, price)
VALUES (1103, 2023, 'blue', 400)
RETURNING *;
```

- Возврат конкретных столбцов

```
INSERT INTO some_table (id, color, price)
VALUES
    (1101, 'red', 200),
    (1102, 'green', 300)
RETURNING id, price;
```

INTO RESULT

Позволяет задать пользовательскую метку для SELECT.

Примеры

```
SELECT 1 INTO RESULT foo;
```

```
SELECT * FROM
my_table
WHERE value % 2 == 0
INTO RESULT `Название результата`;
```

KILL SESSION

`KILL SESSION` принудительно завершает сессию выполнения запросов в текущей базе данных. Команда позволяет остановить запрос, создающий нежелательную нагрузку, или закрыть ненужную долгоживущую сессию.

Команда выполняется через Query Service и может завершать сессии как Query Service, так и Table Service, в том числе на другой ноде. Сессии чтения и записи топиков и сессии сервиса координации этой командой не завершаются.

Синтаксис

Для завершения одной сессии укажите её идентификатор в бэктиках:

```
KILL SESSION `<session_id>`;
```

Где `<session_id>` - значение колонки `SessionId` из системного представления `.sys/query_sessions`.

Для программного вызова передавайте идентификатор отдельным параметром типа `Utf8`, объявленным с помощью `DECLARE`:

```
DECLARE $session_id AS Utf8;
KILL SESSION $session_id;
```

Значение `$session_id` передаётся через SDK или другой клиент отдельно от текста запроса. Параметр должен содержать полный ID существующей сессии; `NULL` и пустая строка недопустимы. В этой форме бэктики не используются.

Текущая версия поддерживает один ID на оператор. Список ID и выборка ID из подзапроса не поддерживаются.

Пример: найти и завершить сессию

Сначала найдите сессии, в которых запрос выполняется не менее 30 секунд. Следующий `SELECT` использует `CurrentUtcTimestamp` и литерал `Interval`:

```
SELECT SessionId, UserSID, QueryStartAt, Query
FROM `.sys/query_sessions`
WHERE State = "EXECUTING"
  AND QueryStartAt <= CurrentUtcTimestamp() - Interval("PT30S")
ORDER BY QueryStartAt
LIMIT 20;
```

Выберите нужный `SessionId` из результата и выполните отдельный запрос. Например:

```
KILL SESSION `ydb://session/3?node_id=52910&id=MWFmNGYwYTAtYzJkN2RhOWEtZmFkMDhlMTUtZjU0ZDE0OTA%3D`;
```

ID в примере необходимо заменить на идентификатор сессии из вашей базы.

Права доступа

Для выполнения команды проверяются права доступа на корне текущей базы данных:

- Пользователь с правом `CONNECT` (`ydb.database.connect`) может завершать свои сессии. Владение определяется совпадением непустых SID вызывающего пользователя и создателя сессии. Общая группа пользователей сама по себе не означает владение.
- Для завершения чужих сессий дополнительно требуется `UPDATE ROW` (`ydb.granular.update_row`) на корне базы. Это право на таблице или поддиректории недостаточно.
- Администраторы кластера, а также администраторы базы при включённом `enable_database_admin`, могут завершать любые сессии в этой базе.

Например, следующий `GRANT` разрешает пользователю `session_operator` завершать любые сессии базы `/Root/my_database`:

```
GRANT CONNECT, UPDATE ROW ON `/Root/my_database` TO session_operator;
```

Права могут быть выданы непосредственно пользователю или его группе. Наборы прав, включающие `UPDATE ROW`, также подходят при наличии `CONNECT`.

Результат и ограничения

Команда не возвращает результирующий набор. Статус `SUCCESS` означает, что сессия удалена после завершения её очистки, включая откат открытых транзакций.

Если выполняющийся на сессии запрос удалось отменить, его клиент получает `CANCELLED` и сообщение о завершении через `KILL SESSION`. Сообщение содержит SID инициатора, если он известен. Обработка ошибок должна учитывать, что административно отменённый запрос не следует автоматически запускать заново.

> **Важно**
>
> Завершение сессии не отменяет уже зафиксированные изменения. Если запись дошла до стадии, на которой отмена уже невозможна, она может завершиться успешно. DDL-операция, уже принятая на исполнение, также может продолжить выполнение после закрытия сессии.

`KILL SESSION` выполняется вне пользовательской транзакции, поскольку закрытие другой сессии нельзя откатить вместе с изменениями данных. В SDK используйте режим без явно заданной транзакции, например `TTxControl::NoTx()` в C++ Query Client. Через устаревший `TableService/ExecuteSchemeQuery` команда не выполняется.

Нельзя завершить сессию, на которой выполняется сама команда. Используйте другую сессию для вызова `KILL SESSION`.

Отправляйте `KILL SESSION` отдельным запросом, не объединяя его с запросами к данным.

Если ожидание завершения истекло по таймауту или связь оборвалась, удаление могло уже начаться и продолжиться. Проверьте наличие сессии повторно; повторное завершение уже удалённой сессии возвращает ошибку.

Ошибки

При отказе команда возвращает статус и сообщение с причиной. Основные случаи:

Статус	Причина
`BAD_REQUEST`	Неверный формат ID или недопустимое значение параметра. Ошибки типа параметра могут быть обнаружены раньше, при компиляции запроса.
`UNAUTHORIZED`	Недостаточно прав. Для пользователя без права завершать чужие сессии отсутствие сессии и отказ доступа возвращают одинаковое сообщение `Session not found or access denied`.
`PRECONDITION_FAILED`	Сессия не найдена при наличии права завершать любые сессии; попытка завершить сессию самой команды; выполнение внутри пользовательской транзакции.
`GENERIC_ERROR`	Неподдерживаемое смешивание `KILL SESSION` с запросами к данным.
`TIMEOUT`	Истёк срок ожидания завершения операции.
`UNAVAILABLE`	Не удалось связаться с нодой, владеющей сессией, или доставить запрос.
`UNSUPPORTED`	Функциональность выключена фичефлагом.

PRAGMA

Определение

Переопределение настроек.

Синтаксис

`PRAGMA x.y = "z";` или `PRAGMA x.y("z", "z2", "z3");`:

- `x` — (опционально) категория настройки.
- `y` — название настройки.
- `z` — (опционально для флагов) значение настройки. Допустимо использование следующих суффиксов:
  - `Kb`, `Mb`, `Gb` — для объема информации.
  - `sec`, `min`, `h`, `d` — для временных значений.

Примеры

```
PRAGMA AutoCommit;
```

```
PRAGMA TablePathPrefix = "home/yql";
```

```
PRAGMA Warning("disable", "1101");
```

За некоторым исключением, значение настроек можно вернуть в состояние по умолчанию с помощью `PRAGMA my_pragma = default;`.

Полный список доступных настроек см. в таблице ниже.

Область действия

Если не указано иное, прагма влияет на все идущие следом выражения вплоть до конца модуля, в котором она встречается. При необходимости и логической возможности допустимо менять значение настройки несколько раз в одном запросе, чтобы оно было разным на разных этапах выполнения.
Существуют также специальные scoped прагмы, область действия которых определяется по тем же правилам, что и область видимости именованных выражений.
В отличие от scoped прагм, обычные прагмы могут использоваться только в глобальной области видимости (не внутри лямбда-функций, ACTION и т.п.).

Глобальные

AutoCommit

Тип значения	По умолчанию
Флаг	false

Автоматически выполнять COMMIT после каждого выражения.

TablePathPrefix

Тип значения	По умолчанию
Строка	—

Добавить указанный префикс к путям таблиц внутри кластеров. Работает по принципу объединения путей в файловой системе: поддерживает ссылки на родительский каталог `..` и не требует добавления слеша справа. Например,

```
PRAGMA TablePathPrefix = "home/yql";
SELECT * FROM test;
```

Префикс не добавляется, если имя таблицы указано как абсолютный путь (начинается с /).

RelativePathPrefix

Тип значения	По умолчанию
Строка	—

Задаёт префикс путей схемных объектов относительно корня базы данных. Значение `RelativePathPrefix` должно быть строковым литералом и не должно начинаться с `/`. Прагму можно указать только один раз, только на верхнем уровне и только до SQL-операторов. Её нельзя использовать вместе с `TablePathPrefix`. Прагма применяется только к относительным путям схемных объектов в SQL-запросе; абсолютные пути не изменяются.

```
PRAGMA RelativePathPrefix = "folder";
SELECT * FROM test;
```

UseTablePrefixForEach

Тип значения	По умолчанию
Флаг	false

`EACH` использует TablePathPrefix для каждого элемента списка.

Warning

Тип значения	По умолчанию
1. Действие 2. Код предупреждения либо символ "*"	—

Действие:

- `disable` — отключить;
- `error` — приравнять к ошибке;
- `default` — вернуть поведение по умолчанию.

Код предупреждения возвращается вместе с самим текстом (в веб-интерфейсе отображается справа).

Примеры:

```
PRAGMA Warning("error", "*");
PRAGMA Warning("disable", "1101");
PRAGMA Warning("default", "4503");
```

В данном случае все предупреждения будут считаться ошибками, за исключением предупреждение с кодом `1101`, которое будет отключено, и `4503`, которое будет обрабатываться по умолчанию (то есть останется предупреждением). Поскольку предупреждения могут добавляться в новых релизах YQL, следует с осторожностью пользоваться конструкцией `PRAGMA Warning("error", "*");` (как минимум покрывать такие запросы автотестами).

Greetings

Тип значения	По умолчанию
Текст	—

Выдать указанный текст в качестве Info сообщения запроса.

Пример:
```
PRAGMA Greetings("It's a good day!");
```

WarningMsg

Тип значения	По умолчанию
Текст	—

Выдать указанный текст в качестве Warning сообщения запроса.

Пример:
```
PRAGMA WarningMsg("Attention!");
```

SimpleColumns

`SimpleColumns` / `DisableSimpleColumns`

Тип значения	По умолчанию
Флаг	true

При использовании `SELECT foo.* FROM ... AS foo` убрать префикс `foo.` у имен результирующих колонок.

Работает в том числе и для JOIN, но в этом случае имеет право упасть в случае конфликта имен (который можно разрешить с помощью WITHOUT и переименования колонок). Для JOIN в режиме SimpleColumns производится неявный Coalesce ключевых колонок: запрос `SELECT * FROM T1 AS a JOIN T2 AS b USING(key)` в режиме SimpleColumns работает аналогично `SELECT a.key ?? b.key AS key, ... FROM T1 AS a JOIN T2 AS b USING(key)`.

CoalesceJoinKeysOnQualifiedAll

`CoalesceJoinKeysOnQualifiedAll` / `DisableCoalesceJoinKeysOnQualifiedAll`

Тип значения	По умолчанию
Флаг	true

Управляет неявным Coalesce для ключевых колонок `JOIN` в режиме SimpleColumns. Если флаг установлен, то Coalesce ключевых колонок происходит при наличии хотя бы одного выражения вида `foo.*` или `*` в SELECT - например `SELECT a.* FROM T1 AS a JOIN T2 AS b USING(key)`. Если флаг сброшен, то Coalesce ключей JOIN происходит только при наличии '*' после `SELECT`

StrictJoinKeyTypes

`StrictJoinKeyTypes` / `DisableStrictJoinKeyTypes`

Тип значения	По умолчанию
Флаг	false

Если флаг установлен, то JOIN будет требовать строгого совпадения типов ключей.
По умолчанию JOIN предварительно конвертирует ключи к общему типу, что может быть нежелательно с точки зрения производительности.
StrictJoinKeyTypes является scoped настройкой.

AnsiInForEmptyOrNullableItemsCollections

Тип значения	По умолчанию
Флаг	false

Наличие этой прагмы приводит поведение оператора `IN` в соответствие со стандартом в случае наличия `NULL` в левой или правой части оператора `IN`. Также изменено поведение `IN` в случае, когда справа был Tuple с элементами разных типов. Примеры:

```
1 IN (2, 3, NULL) = NULL (было Just(False))
NULL IN () = Just(False) (было NULL)
(1, null) IN ((2, 2), (3, 3)) = Just(False) (было NULL)
```

Подробнее про поведение `IN` при наличии `NULL`ов в операндах можно почитать здесь. Явным образом выбрать старое поведение можно указав прагму `DisableAnsiInForEmptyOrNullableItemsCollections`. Если никакой прагмы не задано, то выдается предупреждение и работает старый вариант.

AnsiRankForNullableKeys

Тип значения	По умолчанию
Флаг	false

Приводит поведение RANK/DENSE_RANK в соответствие со стандартом при наличии опциональных типов в ключах сортировки окна или в аргументе этих оконных функций. А именно:

- типом результата всегда является Uint64, а не Uint64?;
- null-ы в ключах считаются равными друг другу (текущая реализация возвращает NULL).

Явным образом выбрать старое поведению можно указав прагму `DisableAnsiRankForNullableKeys`. Если никакой прагмы не задано, то выдается предупреждение и работает старый вариант.

AnsiCurrentRow

Тип значения	По умолчанию
Флаг	false

Приводит неявное задание рамки окна при наличии ORDER BY в соответствие со стандартом.
Если AnsiCurrentRow не установлен, то окно `(ORDER BY key)` эквивалентно `(ORDER BY key ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)`.
Стандарт же требует, чтобы такое окно вело себя как `(ORDER BY key RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)`.
Разница состоит в трактовке `CURRENT ROW`. В режиме `ROWS` `CURRENT ROW` трактуется буквально – текущая строка в партиции.
А в режиме `RANGE` конец рамки `CURRENT ROW` означает "последняя строка в партиции с ключом сортировки, равным текущей строке".

OrderedColumns

`OrderedColumns` / `DisableOrderedColumns`

Выводить порядок колонок в SELECT/JOIN/UNION ALL и сохранять его при записи результатов. По умолчанию порядок колонок не определен.

PositionalUnionAll

Включить соответствующий стандарту поколоночный режим выполнения UNION ALL. При этом автоматически включается упорядоченность колонок.

RegexUseRe2

Тип значения	По умолчанию
Флаг	false

Использовать Re2 UDF вместо Pcre для выполнения SQL операторов `REGEX`, `MATCH`, `RLIKE`. Re2 UDF поддерживает корректную обработку Unicode-символов в отличие от используемой по умолчанию Pcre UDF.

ClassicDivision

Тип значения	По умолчанию
Флаг	true

В классическом варианте результат целочисленного деления остаётся целочисленным (по умолчанию).
Если отключить — результат всегда становится Double.
ClassicDivision является scoped настройкой.

CheckedOps

Тип значения	По умолчанию
Флаг	false

При включенном режиме если в результате выполнения агрегационных функций SUM/SUM_IF, бинарных операций `+`, `-`, `*`, `/`, `%` или унарной операции `-` над целыми числами происходит выход за границы целевого типа аргументов или результата, то возвращается `NULL`.

Если отключить - переполнение не проверяется.
Не влияет на операции с числами с плавающей точкой или `Decimal`.
CheckedOps является scoped настройкой.

UnicodeLiterals

`UnicodeLiterals` / `DisableUnicodeLiterals`

Тип значения	По умолчанию
Флаг	false

При включенном режиме строковые литералы без суффиксов вида "foo"/'bar'/@@multiline@@ будут иметь тип `Utf8`, при выключенном - `String`.
UnicodeLiterals является scoped настройкой.

WarnUntypedStringLiterals

`WarnUntypedStringLiterals` / `DisableWarnUntypedStringLiterals`

Тип значения	По умолчанию
Флаг	false

При включенном режиме для строковых литералов без суффиксов вида "foo"/'bar'/@@multiline@@ будет генерироваться предупреждение. Его можно подавить, если явно выбрать суффикс `s` для типа `String`, либо `u` для типа `Utf8`.
WarnUntypedStringLiterals является scoped настройкой.

AllowDotInAlias

Тип значения	По умолчанию
Флаг	false

Разрешить использовать точку в именах результирующих колонок. По умолчанию отключено, т.к. дальнейшее использование таких колонок в JOIN полностью не реализовано.

WarnUnnamedColumns

Тип значения	По умолчанию
Флаг	false

Генерировать предупреждение если для безымянного выражения в `SELECT` было автоматически сгенерировано имя колонки (вида `column[0-9]+`).

GroupByLimit

Тип значения	По умолчанию
Положительное число	32

Увеличение лимита на число группировок в GROUP BY.

GroupByCubeLimit

Тип значения	По умолчанию
Положительное число	5

Увеличение лимита на число размерностей GROUP BY.

Использовать нужно аккуратно, так как вычислительная сложность запроса растет экспоненциально по отношению к числу размерностей.

Yson

Управление поведением Yson UDF по умолчанию, подробнее см. в документации по ней и в частности Yson::Options.

`yson.AutoConvert`

Тип значения	По умолчанию
Флаг	false

Автоматическое конвертация значений в требуемый тип данных во всех вызовах Yson UDF, в том числе и неявных.

`yson.Strict`

Тип значения	По умолчанию
Флаг	true

Управление строгим режимом во всех вызовах Yson UDF, в том числе и неявных. Без значения или при значении `"true"` - включает строгий режим. Со значением `"false"` - отключает.

`yson.DisableStrict`

Тип значения	По умолчанию
Флаг	false

Инвертированная версия `yson.Strict`. Без значения или при значении `"true"` - отключает строгий режим. Со значением `"false"` - включает.

YDB

`ydb.CostBasedOptimization`

Значение	Поведение оптимизатора
on	Стоимостный оптимизатор выключен для текущего запроса
off	Стоимостный оптимизатор выключен для текущего запроса
auto	Стоимостный оптимизатор работает в соотвествии с текущем уровнем CostBasedOptimizationLevel

`ydb.CostBasedOptimizationLevel`

Уровень	Поведение оптимизатора
0	Стоимостный оптимизатор выключен
1	Стоимостный оптимизатор выключен, считаются предсказания оптимизатора
2	Стоимостный оптимизатор включается только для запросов, где участвуют колоночные таблицы
3	Стоимостный оптимизатор включается для всех запросов, но для строковых таблиц предпочитается алгоритм джоина LookupJoin
4	Стоимостный оптимизатор включен для всех запросов

Примечание

По умолчанию выставлен уровень 2

`ydb.OptimizerHints`

Значение прагмы описано в отдельном разделе.

REPLACE INTO

Важно

В настоящее время одновременное использование колоночных и строковых таблиц поддерживается в транзакциях, в которых данные только читаются, но не изменяются. Поддержка транзакций с возможностью модификации данных при одновременном использовании строковых и колоночных таблиц находится в разработке.

Если попытаться выполнить операцию записи в транзакции, в которой задействованы и колоночные, и строковые таблицы, транзакция завершится с ошибкой:

```
Write transactions that use both row-oriented and column-oriented tables are disabled at current t
ime
```

В отличие от `INSERT INTO` и `UPDATE`, запросы `UPSERT INTO` и `REPLACE INTO` не требуют предварительного чтения данных, поэтому выполняются быстрее. `REPLACE INTO` сохраняет данные в таблицу с перезаписью строк по первичному ключу. Если заданный первичный ключ отсутствует, в таблицу будет добавлена новая строка. Если задан существующий первичный ключ, строка будет перезаписана. При этом значения столбцов, не определенных в операции `REPLACE INTO`, заменяются на значения по умолчанию.

Примеры

- Задание значений для `REPLACE INTO` с помощью `VALUES`:

```
REPLACE INTO my_table (Key1, Key2, Value2) VALUES
    (1u, "One", 101),
    (2u, "Two", 102);
COMMIT;
```

- Получение значений для `REPLACE INTO` с помощью выборки `SELECT`:

```
REPLACE INTO my_table
SELECT Key AS Key1, "Empty" AS Key2, Value AS Value1
FROM my_table1;
COMMIT;
```

REVOKE

Команда `REVOKE` позволяет отозвать права доступа к объектам схемы для пользователей или групп пользователей.

Синтаксис:

```
REVOKE [GRANT OPTION FOR] {{permission_name} [, ...] | ALL [PRIVILEGES]} ON {path_to_scheme_object [, ...]} F
ROM {role_name [, ...]}
```

- `permission_name` - имя права доступа к объектам схемы, которое нужно отозвать.
- `path_to_scheme_object` - путь до объекта схемы, с которого отзываются права.
- `role_name` - имя пользователя или группы, для которого отзываются права на объект схемы.

`GRANT OPTION FOR` - использование этой конструкции отзывает у пользователя или группы право управлять правами доступа. Все ранее выданные этим пользователем права остаются в силе. Конструкция имеет функцианальность аналогичную отзыву права `"ydb.access.grant"` или `GRANT`.

Права доступа

В качестве имён прав доступа можно использовать имена YDB прав или соответствующие им ключевые слова YQL.
В таблице ниже перечислены возможные имена прав.

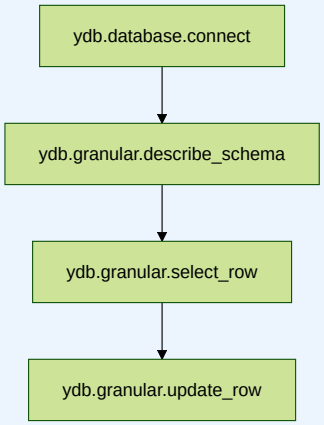
YDB право	Ключевое слово YQL	Описание
Права уровня баз данных		
`ydb.database.connect`	`CONNECT`	Право подключаться к базе данных
`ydb.database.create`	`CREATE`	Право создавать новые базы данных в кластере
`ydb.database.drop`	`DROP`	Право удалять базы данных в кластере
Элементарные права на объекты базы данных		
`ydb.granular.select_row`	`SELECT ROW`	Право читать строки из таблицы (select), читать сообщения из топиков, использовать значения секретов
`ydb.granular.update_row`	`UPDATE ROW`	Право обновлять строки в таблице (insert, update, upsert, replace), писать сообщения в топики. На корне базы также разрешает завершать чужие сессии через KILL SESSION при наличии `CONNECT`
`ydb.granular.erase_row`	`ERASE ROW`	Право удалять строки из таблицы (delete)
`ydb.granular.create_directory`	`CREATE DIRECTORY`	Право создавать и удалять директории, в том числе существующие и вложенные
`ydb.granular.create_table`	`CREATE TABLE`	Право создавать таблицы (в том числе индексные, внешние, колоночные), представления, последовательности
`ydb.granular.create_queue`	`CREATE QUEUE`	Право создавать топики
`ydb.granular.remove_schema`	`REMOVE SCHEMA`	Право удалять объекты (директории, таблицы, топики), которые были созданы посредством использования прав
`ydb.granular.describe_schema`	`DESCRIBE SCHEMA`	Право просмотра имеющихся прав доступа (ACL) на объект доступа, просмотра описания объектов доступа (директории, таблицы, топики)
`ydb.granular.alter_schema`	`ALTER SCHEMA`	Право изменять объекты доступа (директории, таблицы, топики), в том числе права пользователей на объекты доступа
Дополнительные флаги		
`ydb.access.grant`	`GRANT`	Право предоставлять или отзывать права у других пользователей в объёме, не превышающем текущий объём прав пользователя на объекте доступа
Права, основанные на других правах		
`ydb.tables.modify`	`MODIFY TABLES`	`ydb.granular.update_row` + `ydb.granular.erase_row`
`ydb.tables.read`	`SELECT TABLES`	Синоним `ydb.granular.select_row`
`ydb.generic.list`	`LIST`	Синоним `ydb.granular.describe_schema`
`ydb.generic.read`	`SELECT`	`ydb.granular.select_row` + `ydb.generic.list`
`ydb.generic.write`	`INSERT`	`ydb.granular.update_row` + `ydb.granular.erase_row` + `ydb.granular.create_directory` + `ydb.granular.create_table` + `ydb.granular.create_queue` + `ydb.granular.remove_schema` + `ydb.granular.alter_schema`
`ydb.generic.use_legacy`	`USE LEGACY`	`ydb.generic.read` + `ydb.generic.write` + `ydb.access.grant`
`ydb.generic.use`	`USE`	`ydb.generic.use_legacy` + `ydb.database.connect`

`ydb.generic.manage`	`MANAGE`	`ydb.database.create` + `ydb.database.drop`
`ydb.generic.full_legacy`	`FULL LEGACY`	`ydb.generic.use_legacy` + `ydb.generic.manage`
`ydb.generic.full`	`FULL`	`ydb.generic.use` + `ydb.generic.manage`

- `ALL [PRIVILEGES]` - используется для указания всех возможных прав на объекты схемы для пользователей или групп. `PRIVILEGES` является необязательным ключевым словом, необходимым для совместимости с SQL стандартом.

Примечание

Права `ydb.database.connect`, `ydb.granular.describe_schema`, `ydb.granular.select_row`, `ydb.granular.update_row` необходимо рассматривать как слои прав.



Например, для изменения строк необходимо не только право `ydb.granular.update_row`, но и все вышележащие права.

Примеры

- Отозвать право `ydb.generic.read` на таблицу `/shop_db/orders` у пользователя `user1`:

```
REVOKE 'ydb.generic.read' ON `/shop_db/orders` FROM user1;
```

Та же команда, с использованием ключевого слова

```
REVOKE SELECT ON `/shop_db/orders` FROM user1;
```

- Отозвать права `ydb.database.connect`, `ydb.generic.list` на корень базы `/shop_db` у пользователя `user2` и группы `group1`:

```
REVOKE LIST, CONNECT ON `/shop_db` FROM user2, group1;
```

- Отозвать право `ydb.generic.use` на таблицы `/shop_db/orders` и `/shop_db/sellers` у пользователей `user1@domain`, `user2@domain`:

```
REVOKE 'ydb.generic.use' ON `/shop_db/orders`, `/shop_db/sellers` FROM `user1@domain`, `user2@domain`;
```

- Отозвать все права на таблицу `/shop_db/sellers` для пользователя `user`:

```
REVOKE ALL ON `/shop_db/sellers` FROM user;
```

RESTORE

Выражение `RESTORE` восстанавливает таблицы из коллекции резервных копий.

```
RESTORE collection_name;
```

Параметры

- `collection_name` — имя коллекции резервных копий для восстановления.

Поведение восстановления

Операция восстановления:

- Находит в коллекции **актуальную цепочку** резервных копий (полная копия и следующие за ней инкрементальные).
- Восстанавливает таблицы в состояние на момент **самой поздней** резервной копии в этой цепочке: последовательно применяет полную копию и все последующие инкрементальные.
- Завершается с ошибкой, если хотя бы одна из восстанавливаемых таблиц уже существует по тому же пути.

Произвольная точка восстановления «между» двумя сохранёнными копиями не задаётся: восстанавливается состояние, зафиксированное одной из копий в цепочке.

> **Важно**
>
> Операция восстановления завершится с ошибкой, если хотя бы одна из восстанавливаемых таблиц уже существует по тому же пути. Переименуйте или удалите конфликтующие таблицы перед восстановлением.

Примеры

Восстановление из коллекции резервных копий:

```
-- Восстановить все таблицы из коллекции
RESTORE production_backups;
```

Мониторинг операций восстановления

Операции восстановления выполняются асинхронно в фоновом режиме. Вы можете отслеживать их прогресс с помощью YDB CLI:

```
# Список операций восстановления
ydb operation list incbackup

# Получить детали операции
ydb operation get <operation-id>
```

См. также

- Коллекции резервных копий
- CREATE BACKUP COLLECTION
- BACKUP
- DROP BACKUP COLLECTION
- Резервное копирование и восстановление

SELECT

Возвращает результат вычисления выражений, указанных после `SELECT`.

Может использоваться в сочетании с другими операциями для получения иного эффекта.

Примеры

```
SELECT "Hello, world!";
```

```
SELECT 2 + 2;
```

Процедура выполнения SELECT

Результат запроса `SELECT` вычисляется следующим образом:

- определяется набор входных таблиц – вычисляются выражения после FROM;
- выполняется FLATTEN COLUMNS или FLATTEN BY; алиасы, заданные во `FLATTEN BY`, становятся видны после этой точки;
- выполняются все JOIN;
- к полученным данным добавляются (или заменяются) колонки, заданные в GROUP BY ... AS ...;
- выполняется WHERE: все данные, не удовлетворяющие предикату, отфильтровываются;
- выполняется GROUP BY, вычисляются значения агрегатных функций;
- выполняется фильтрация HAVING;
- вычисляются значения оконных функций;
- вычисляются выражения в `SELECT`;
- выражениям в `SELECT` назначаются имена заданные алиасами;
- к полученным таким образом колонкам применяется top-level DISTINCT;
- таким же образом вычисляются все подзапросы в UNION ALL, выполняется их объединение (см. PRAGMA AnsiOrderByLimitInUnionAll);
- выполняется сортировка согласно ORDER BY;
- к полученному результату применяются OFFSET и LIMIT.

Порядок колонок в YQL

В стандартном SQL порядок колонок указанных в проекции (в `SELECT`) имеет значение. Помимо того, что порядок колонок должен сохраняться при отображении результатов запроса или при записи в новую таблицу, некоторые конструкции SQL этот порядок используют.
Это относится в том числе к UNION ALL и к позиционному ORDER BY (ORDER BY ordinal).

По умолчанию в YQL порядок колонок игнорируется:

- порядок колонок в выходных таблицах и в результатах запроса не определен
- схема данных результата `UNION ALL` выводится по именам колонок, а не по позициям

При включении `PRAGMA OrderedColumns;` порядок колонок сохраняется в результатах запроса и выводится из порядка колонок во входных таблицах по следующим правилам:

- `SELECT` с явным перечислением колонок задает соответствующий порядок;
- `SELECT` со звездочкой (`SELECT * FROM ...`) наследует порядок из своего входа;
- порядок колонок после JOIN: сначала колонки левой стороны, потом правой. Если порядок какой-либо из сторон, присутствующей в выходе `JOIN`, не определен, порядок колонок результата также не определен;
- порядок `UNION ALL` зависит от режима выполнения UNION ALL;
- порядок колонок для AS_TABLE не определен;

Комбинация запросов

Результаты нескольких SELECT (или подзапросов) могут быть объединены с помощью ключевых слов `UNION` и `UNION ALL`.

```
query1 UNION [ALL] query2 (UNION [ALL] query3 ...)
```

Объединение более двух запросов интерпретируется как левоассоциативная операция, то есть

```
query1 UNION query2 UNION ALL query3
```

интерпретируется как

```
(query1 UNION query2) UNION ALL query3
```

При наличии `ORDER BY/LIMIT/DISCARD/INTO RESULT` в объединяемых подзапросах применяются следующие правила:

- `ORDER BY/LIMIT/INTO RESULT` допускается только после последнего подзапроса;
- `DISCARD` допускается только перед первым подзапросом;
- указанные операторы действуют на результат `UNION [ALL]`, а не на подзапрос;
- чтобы применить оператор к подзапросу, подзапрос необходимо взять в скобки.

Обращение к нескольким таблицам в одном запросе

В стандартном SQL для выполнения запроса по нескольким таблицам используется UNION ALL, который объединяет результаты двух и более `SELECT`. Это не совсем удобно для сценария использования, в котором требуется выполнить один и тот же запрос по нескольким таблицам (например, содержащим данные на разные даты). В YQL, чтобы было удобнее, в

`SELECT` после `FROM` можно указывать не только одну таблицу или подзапрос, но и вызывать встроенные функции, позволяющие объединять данные нескольких таблиц.

Для этих целей определены следующие функции:

`CONCAT(\`table1\`, \`table2\`, \`table3\` VIEW view_name, ...)` — объединяет все перечисленные в аргументах таблицы.

`EACH($list_of_strings)` или `EACH($list_of_strings VIEW view_name)` — объединяет все таблицы, имена которых перечислены в списке строк. Опционально можно передать несколько списков в отдельных аргументах по аналогии с `CONCAT`.

`RANGE(\`prefix\`, \`min\`, \`max\`, \`suffix\`, \`view\`)` — объединяет диапазон таблиц. Аргументы:

- prefix — каталог для поиска таблиц, указывается без завершающего слеша. Единственный обязательный аргумент, если указан только он, то используются все таблицы в данном каталоге.
- min, max — следующие два аргумента задают диапазон имен для включения таблиц. Диапазон инклюзивный с обоих концов. Если диапазон не указан, используются все таблицы в каталоге prefix. Имена таблиц или директорий, находящихся в указанной в prefix директории, сравниваются с диапазоном `[min, max]` лексикографически, а не конкатенируются, таким образом важно указывать диапазон без лидирующих слешей.
- suffix — имя таблицы. Ожидается без начального слеша. Если suffix не указан, то аргументы `[min, max]` задают диапазон имен таблиц. Если suffix указан, то аргументы `[min, max]` задают диапазон папок, в которых существует таблица с именем, указанным в аргументе suffix.

`LIKE(\`prefix\`, \`pattern\`, \`suffix\`, \`view\`)` и `REGEXP(\`prefix\`, \`pattern\`, \`suffix\`, \`view\`)` — аргумент pattern задается в формате, аналогичном одноименным бинарным операторам: LIKE и REGEXP.

`FILTER(\`prefix\`, \`callable\`, \`suffix\`, \`view\`)` — аргумент callable должен являться вызываемым выражением с сигнатурой `(String)->Bool`, который будет вызван для каждой таблицы/подкаталога в каталоге prefix. В запросе будут участвовать только те таблицы, для которых вызываемое значение вернуло `true`. В качестве вызываемого значения удобнее всего использовать лямбда функции.

Важно

Порядок, в котором будут объединены таблицы, всеми вышеперечисленными функциями не гарантируется.

Список таблиц вычисляется **до** запуска самого запроса. Поэтому созданные в процессе запроса таблицы не попадут в результаты функции.

По умолчанию схемы всех участвующих таблиц объединяются по правилам UNION ALL. Если объединение схем не желательно, то можно использовать функции с суффиксом `_STRICT`, например `CONCAT_STRICT` или `RANGE_STRICT`, которые работают полностью аналогично оригинальным, но считают любое расхождение в схемах таблиц ошибкой.

Для указания кластера объединяемых таблиц нужно указать его перед названием функции.

Все аргументы описанных выше функций могут быть объявлены отдельно через именованные выражения. В этом случае в них также допустимы и простые выражения посредством неявного вызова EvaluateExpr.

Имя исходной таблицы, из которой изначально была получена каждая строка, можно получить при помощи функции TablePath().

Примеры

```
SELECT * FROM CONCAT(
  `table1`,
  `table2`,
  `table3`);
```

```
$indices = ListFromRange(1, 4);
$tables = ListMap($indices, ($index) -> {
    RETURN "table" || CAST($index AS String);
});
SELECT * FROM EACH($tables); -- идентично предыдущему примеру
```

```
SELECT * FROM RANGE(`my_folder`);
```

```
SELECT * FROM some_cluster.RANGE( -- Кластер можно указать перед названием функции
  `my_folder`,
  `from_table`,
  `to_table`);
```

```
SELECT * FROM RANGE(
  `my_folder`,
  `from_folder`,
  `to_folder`,
  `my_table`);
```

```
SELECT * FROM RANGE(
  `my_folder`,
  `from_table`,
  `to_table`,
  ``,
  `my_view`);
```

```
SELECT * FROM LIKE(
  `my_folder`,
  "2017-03-%"
);
```

```
SELECT * FROM REGEXP(
  `my_folder`,
  "2017-03-1[2-4]?"
);
```

```
$callable = ($table_name) -> {
    return $table_name > "2017-03-13";
};

SELECT * FROM FILTER(
  `my_folder`,
  $callable
);
```

Поддерживаемые конструкции в SELECT

- FROM
- FROM AS_TABLE
- FROM SELECT
- DISTINCT
- UNIQUE DISTINCT
- UNION
- WITH
- WITHOUT
- WHERE
- ORDER BY
- ASSUME ORDER BY
- LIMIT OFFSET
- JOIN
- GROUP BY
- FLATTEN
- WINDOW
- VIEW secondary_index
- VIEW vector_index
- VIEW fulltext_index
- Гибридный поиск
- VIEW json_index

FROM

Источник данных для `SELECT`. В качестве аргумента может принимать имя таблицы, результат другого `SELECT` или именованное выражение. К именованным выражениям можно обращаться как к таблицам(`FROM AS_TABLE`).

Ещё в YQL можно выполнить запрос по нескольким таблицам. Для этого в `SELECT` после `FROM` можно указывать не только одну таблицу или подзапрос, но и вызывать встроенные функции, позволяющие объединять данные нескольких таблиц.

Также вы можете обходить не таблицы, а итерироваться по дереву целевого кластера, с возможностью накопить состояние, обычно список путей к таблицам.

Между `SELECT` и `FROM` через запятую указываются имена столбцов из источника или `*` для выбора всех столбцов.

Примеры

```
SELECT key FROM my_table;
```

```
SELECT * FROM
  (SELECT value FROM my_table);
```

```
$table_name = "my_table";
SELECT * FROM $table_name;
```

FROM AS_TABLE

Обращение к именованным выражениям как к таблицам с помощью функции `AS_TABLE`.

`AS_TABLE($variable)` позволяет использовать значение `$variable` в качестве источника данных для запроса. При этом переменная `$variable` должна иметь тип `List<Struct<...>>`.

Пример

```
$data = AsList(
    AsStruct(1u AS Key, "v1" AS Value),
    AsStruct(2u AS Key, "v2" AS Value),
    AsStruct(3u AS Key, "v3" AS Value));

SELECT Key, Value FROM AS_TABLE($data);
```

При совместном использовании с другими операторами изменения данных, такими как UPSERT INTO или INSERT INTO, необходимо или указывать модифицируемые колонки и в источнике, и в приемнике:

```
$data = AsList(
    AsStruct(1u AS Key, "v1" AS Value),
    AsStruct(2u AS Key, "v2" AS Value),
    AsStruct(3u AS Key, "v3" AS Value));

INSERT INTO `my_table` (Key, Value) SELECT Key, Value FROM AS_TABLE($data);
```

Или не указывать их вовсе:

```
$data = AsList(
    AsStruct(1u AS Key, "v1" AS Value),
    AsStruct(2u AS Key, "v2" AS Value),
    AsStruct(3u AS Key, "v3" AS Value));

INSERT INTO `my_table` SELECT * FROM AS_TABLE($data);
```

FROM ... SELECT ...

Перевернутая форма записи, в которой сначала указывается источник данных, а затем — операция.

Примеры

```
FROM my_table SELECT key, value;
```

```
FROM a_table AS a
JOIN b_table AS b
USING (key)
SELECT *;
```

Чтение данных из топика

Можно выполнять чтение данных из топика обычным `SELECT` без создания потокового запроса.

> **Важно**
>
> Этот способ предназначен только для отладки и проверки данных в топике. Для production процессов создавайте потоковые запросы через CREATE STREAMING QUERY.

> **Примечание**
>
> В примерах:
>
> - `ext_source` — заранее созданный `external data source`;
> - `input_topic` — локальный или внешний топик.
>
> Подробнее — локальные и внешние топики.

Примеры

Чтение текущих данных

```
SELECT
    Data
FROM
    ext_source.input_topic -- или локальный топик input_topic
WITH (
    FORMAT = raw,
    SCHEMA = (
        Data String
    )
)
```

Чтение с ожиданием новых данных

```
SELECT
    Data
FROM
    input_topic -- или внешний топик ext_source.input_topic
WITH (
    FORMAT = raw,
    SCHEMA = (
        Data String
    ),
    STREAMING = "TRUE"
)
LIMIT 1
```

Подробнее — чтение из топика.

См. также

- Чтение из топика
- Потоковые запросы
- CREATE STREAMING QUERY

FLATTEN

FLATTEN BY

Преобразует строки исходной таблицы с помощью вертикального разворачивания контейнеров переменной длины (списков или словарей).

Например:

- Исходная таблица:

[a, b, c]	1
[d]	2
[]	3

- Таблица после вызова `FLATTEN BY` к левому столбцу:

a	1
b	1
c	1
d	2

В таблицах YDB не поддерживаются контейнерные типы, поэтому функция FLATTEN BY может применяться только над переменными табличных типов, создаваемых в рамках YQL запроса.

Пример

```
$sample = AsList(
    AsStruct(AsList('a','b','c') AS value, CAST(1 AS Uint32) AS id),
    AsStruct(AsList('d') AS value, CAST(2 AS Uint32) AS id),
    AsStruct(AsList() AS value, CAST(3 AS Uint32) AS id)
);

SELECT value, id FROM as_table($sample) FLATTEN BY (value);
```

Такое преобразование может быть удобным в следующих случаях:

- Когда по ячейкам из столбца-контейнера необходимо вывести статистику (например, через `GROUP BY`).
- Когда в ячейках столбца-контейнера хранятся идентификаторы из другой таблицы, которую нужно присоединить с помощью `JOIN`.

Синтаксис

- `FLATTEN BY` указывается после `FROM`, но перед `GROUP BY`, если `GROUP BY` присутствует в запросе.
- Тип столбца-результата зависит от типа исходного столбца:

Тип контейнера	Тип результата	Комментарий
`List<X>`	`X`	Тип ячейки списка
`Dict<X,Y>`	`Tuple<X,Y>`	Кортеж из двух элементов с парами «ключ—значение»
`Optional<X>`	`X`	Результат практически эквивалентен конструкции `WHERE foo IS NOT NULL`, но тип колонки `foo` будет изменен на `X`

- По умолчанию столбец с результатом заменяет исходный. Используйте `FLATTEN BY foo AS bar` для сохранения исходного контейнера. В результате исходный контейнер останется доступным в `foo`, а построенный — в `bar`.
- Чтобы построить декартово произведение нескольких столбцов-контейнеров, используйте конструкцию `FLATTEN BY (a, b, c)`. Скобки обязательны, чтобы избежать конфликтов в грамматике.
- В `FLATTEN BY` можно использовать только имена столбцов из входной таблицы. Чтобы применить `FLATTEN BY` к результату вычисления, используйте подзапрос.
- В `FLATTEN BY` можно использовать не только столбцы, но и произвольные именованные выражения (в отличие от столбцов `AS` обязательно). Из-за грамматических неоднозначностей выражения после `FLATTEN BY` должны быть заключены в скобки: `... FLATTEN BY (ListSkip(col, 1) AS col) ...`
- Если в исходном столбце были вложенные контейнеры, например `List<Dict<X,Y>>`, `FLATTEN BY` развернет только внешний уровень. Чтобы полностью развернуть вложенные контейнеры, используйте подзапрос.

Примечание

`FLATTEN BY` интерпретирует опциональные типы данных как списки длины 0 или 1. Строки таблицы с `NULL` пропускаются, и тип столбца меняется на аналогичный неопциональный.

`FLATTEN BY` делает только одно преобразование за раз, поэтому на опциональных контейнерах, например, `Optional<List<String>>` следует использовать `FLATTEN LIST BY` или `FLATTEN OPTIONAL BY`.

Уточнение типа контейнера

Чтобы уточнить тип контейнера, по которому необходимо произвести преобразование, можно использовать:

- `FLATTEN LIST BY`

Для `Optional<List<T>>` операция `FLATTEN LIST BY` будет разворачивать список, интерпретируя `NULL`-значение как пустой список.

- `FLATTEN DICT BY`

  Для `Optional<Dict<T>>` операция `FLATTEN DICT BY` будет разворачивать словарь, интерпретируя `NULL`-значение как пустой словарь.

- `FLATTEN OPTIONAL BY`

  Чтобы фильтровать `NULL`-значения без размножения, необходимо уточнить операцию до `FLATTEN OPTIONAL BY`.

Примеры

```
SELECT
  t.item.0 AS key,
  t.item.1 AS value,
  t.dict_column AS original_dict,
  t.other_column AS other
FROM my_table AS t
FLATTEN DICT BY dict_column AS item;
```

```
SELECT * FROM (
    SELECT
        AsList(1, 2, 3) AS a,
        AsList("x", "y", "z") AS b
) FLATTEN LIST BY (a, b);
```

```
SELECT * FROM (
    SELECT
        "1;2;3" AS a,
        AsList("x", "y", "z") AS b
) FLATTEN LIST BY (String::SplitToList(a, ";") as a, b);
```

Аналоги FLATTEN BY для других СУБД

- PostgreSQL: `unnest`;
- Hive: `LATERAL VIEW`;
- MongoDB: `unwind`;
- Google BigQuery: `FLATTEN`;
- ClickHouse: `ARRAY JOIN / arrayJoin`;

FLATTEN COLUMNS

Преобразует таблицу, в которой все столбцы должны являться структурами, в таблицу со столбцами, соответствующими каждому элементу каждой структуры из исходных столбцов.

Имена исходных столбцов-структур не используются и не возвращаются в результате. Имена элементов структур не должны повторяться в исходных столбцах.

Пример

```
SELECT x, y, z
FROM (
  SELECT
    AsStruct(
        1 AS x,
        "foo" AS y),
    AsStruct(
        false AS z)
) FLATTEN COLUMNS;
```

GROUP BY

Группирует результаты `SELECT` по значениям указанных столбцов или выражений. Вместе с `GROUP BY` часто применяются агрегатные функции (`COUNT` , `MAX` , `MIN` , `SUM` , `AVG`) для выполнения вычислений в каждой группе.

Если `GROUP BY` присутствует в запросе, то при выборке столбцов (между `SELECT ... FROM`) допустимы следующие конструкции:

1. Столбцы, по которым производится группировка (присутствующие в аргументе `GROUP BY`).
2. Агрегатные функции (см. следующий раздел). Столбцы, по которым **не** идёт группировка, можно включать только в качестве аргументов агрегатной функции.
3. Функции, выдающие начальное и конечное время текущего окна (`HOP_START` и `HOP_END`) (для `GROUP BY HOP`).
4. Произвольные вычисления, комбинирующие пункты 1-3.

Имеется возможность выполнять группировку по результату вычисления произвольного выражения от исходных столбцов. В этом случае для получения доступа к результату этого выражения рекомендуется присваивать ему имя с помощью `AS` , см. второй пример.

Синтаксис

```
SELECT                              -- В SELECT можно использовать:
    column1,                        -- ключевые колонки, заданные в GROUP BY
    key_n,                          -- именованные выражения, заданные в GROUP BY
    column1 + key_n,                -- произвольные неагрегатные функции от них
    Aggr_Func1( column2 ),          -- агрегатные функции, содержащие в аргументах любые колонки,
    Aggr_Func2( key_n + column2 ), --   включая именованные выражения, заданные в GROUP BY
    ...
FROM table
GROUP BY
    column1, column2, ...,
    <expr> AS key_n           -- При группировке по выражению ему может быть задано имя через AS,
                              -- которое может быть использовано в SELECT
```

Запрос вида `SELECT * FROM table GROUP BY k1, k2, ...` вернет все колонки, перечисленные в GROUP BY, то есть эквивалентен запросу `SELECT DISTINCT k1, k2, ... FROM table` .

Звездочка может также применяться в качестве аргумента агрегатной функции `COUNT` . `COUNT(*)` означает "число строк в группе".

Примечание

Агрегатные функции не учитывают `NULL` в своих аргументах, за исключением функции `COUNT` .

Также в YQL доступен механизм фабрик агрегатных функций, реализованный с помощью функций `AGGREGATION_FACTORY` и `AGGREGATE_BY` .

Примеры

```
SELECT key, COUNT(*) FROM my_table
GROUP BY key;
```

```
SELECT double_key, COUNT(*) FROM my_table
GROUP BY key + key AS double_key;
```

```
SELECT
   double_key,                          -- OK: ключевая колонка
   COUNT(*) AS group_size,              -- OK: COUNT(*)
   SUM(key + subkey) AS sum1,           -- OK: агрегатная функция
   CAST(SUM(1 + 2) AS String) AS sum2,  -- OK: агрегатная функция с константным аргументом
   SUM(SUM(1) + key) AS sum3,           -- ОШИБКА: вложенные агрегации не допускаются
   key AS k1,                           -- ОШИБКА: использование неключевой колонки key без агрегации
   key * 2 AS dk1,                      -- ОШИБКА в YQL: использование неключевой колонки key без агрегации
FROM my_table
GROUP BY
  key * 2 AS double_key,
  subkey as sk,
```

Важно

Возможность указывать имя для колонки или выражения в `GROUP BY .. AS foo` является расширением YQL. Такое имя становится видимым в `WHERE` несмотря на то, что фильтрация по `WHERE` выполняется раньше группировки. В частности, если в таблице `T` есть две колонки `foo` и `bar` , то в запросе `SELECT foo FROM T WHERE foo > 0 GROUP BY bar AS foo` фильтрация фактически произойдет по колонке `bar` из исходной таблицы.

GROUP BY ... SessionWindow()

В YQL поддерживаются группировки по сессиям. К обычным выражениям в `GROUP BY` можно добавить специальную функцию `SessionWindow` :

```
SELECT
  user,
  session_start,
  SessionStart() AS same_session_start, -- то же что и session_start
  COUNT(*) AS session_size,
  SUM(value) AS sum_over_session,
FROM my_table
GROUP BY user, SessionWindow(<time_expr>, <timeout_expr>) AS session_start
```

При этом происходит следующее:

1. Входная таблица партиционируется по ключам группировки, указанным в `GROUP BY`, без учета SessionWindow (в данном случае по `user`). Если кроме SessionWindow в `GROUP BY` ничего нет, то входная таблица попадает в одну партицию
2. Каждая партиция делится на непересекающие подмножества строк (сессии). Для этого партиция сортируется по возрастанию значения выражения `time_expr`. Границы сессий проводятся между соседними элементами партиции, разница значений `time_expr` для которых превышает `timeout_expr`
3. Полученные таким образом сессии и являются финальными партициями, на которых вычисляются агрегатные функции.

Ключевая колонка SessionWindow() (в примере `session_start`) имеет значение "минимальный `time_expr` в сессии". Кроме того, при наличии SessionWindow() в `GROUP BY` может использоваться специальная агрегатная функция SessionStart.

Поддерживается также расширенный вариант SessionWindow с четырьмя аргументами:

`SessionWindow(<order_expr>, <init_lambda>, <update_lambda>, <calculate_lambda>)`

Здесь:

- `<order_expr>` – выражение по которому сортируется исходная партиция
- `<init_lambda>` – лямбда-функция для инициализации состояния расчета сессий. Имеет сигнатуру `(TableRow())->State`. Вызывается один раз на первом (по порядку сортировки) элементе исходной партиции
- `<update_lambda>` – лямбда-функция для обновления состояния расчета сессий и определения границ сессий. Имеет сигнатуру `(TableRow(), State)->Tuple<Bool, State>`. Вызывается на каждом элементе исходной партиции, кроме первого. Новое значения состояния вычисляется на основе текущей строки таблицы и предыдущего состояния. Если первый элемент возвращенного кортежа имеет значение `True`, то с *текущей* строки начнется новая сессия. Ключ новой сессии получается путем применения `<calculate_lambda>` ко второму элементу кортежа.
- `<calculate_lambda>` – лямбда-функция для вычисления ключа сессии ("значения" SessionWindow(), которое также доступно через SessionStart()). Функция имеет сигнатуру `(TableRow(), State)->SessionKey`. Вызывается на первом элемента партиции (после `<init_lambda>`) и на тех элементах, для которых `<update_lambda>` вернула `True` в качестве первого элемента кортежа. Стоит отметить, что для начала новой сессии необходимо, чтобы `<calculate_lambda>` вернула значение, которое отличается от предыдущего ключа сессии. При этом сессии с одинаковыми ключами не объединяются. Например, если `<calculate_lambda>` последовательно возвращает `0, 1, 0, 1`, то это будут четыре различные сессии.

С помощью расширенного варианта SessionWindow можно решить, например, такую задачу: разделить партицию на сессии как в варианте SessionWindow с двумя аргументами, но с ограничением максимальной длины сессии некоторой константой:

Пример

```
$max_len = 1000; -- максимальная длина сессии
$timeout = 100; -- таймаут (timeout_expr в упрощенном варианте SessionWindow)

$init = ($row) -> (AsTuple($row.ts, $row.ts)); -- состояние сессии - тапл из 1) значения временной колонки ts
на первой строчке сессии и 2) на текущей строчке
$update = ($row, $state) -> {
  $is_end_session = $row.ts - $state.0 > $max_len OR $row.ts - $state.1 > $timeout;
  $new_state = AsTuple(IF($is_end_session, $row.ts, $state.0), $row.ts);
  return AsTuple($is_end_session, $new_state);
};
$calculate = ($row, $state) -> ($row.ts);
SELECT
  user,
  session_start,
  SessionStart() AS same_session_start, -- то же что и session_start
  COUNT(*) AS session_size,
  SUM(value) AS sum_over_session,
FROM my_table
GROUP BY user, SessionWindow(ts, $init, $update, $calculate) AS session_start
```

`SessionWindow` может использоваться в `GROUP BY` только один раз.

ROLLUP, CUBE и GROUPING SETS

Результаты вычисления агрегатной функции в виде промежуточных итогов для групп и общих итогов для отдельных столбцов или всей таблицы.

Синтаксис

```
SELECT
    c1, c2,                         -- столбцы, по которым производится группировка

AGGREGATE_FUNCTION(c3) AS outcome_c  -- агрегатная функция (SUM, AVG, MIN, MAX, COUNT)

FROM table_name
```

```
GROUP BY
    GROUP_BY_EXTENSION(c1, c2)       -- расширение GROUP BY: ROLLUP, CUBE или GROUPING SETS
```

- `ROLLUP` — группирует значения столбцов в порядке их перечисления в аргументах (строго слева направо), формирует промежуточные итоги для каждой группы и общий итог.
- `CUBE` — группирует значения для всех возможных комбинаций столбцов, формирует промежуточные итоги для каждой группы и общий итог.
- `GROUPING SETS` — задает группы для промежуточных итогов.

`ROLLUP`, `CUBE` и `GROUPING SETS` можно комбинировать через запятую.

GROUPING

В промежуточном итоге значения столбцов, которые не участвуют в вычислениях, заменяются на `NULL`. В общем итоге на `NULL` заменяются значения всех столбцов. `GROUPING` — функция, которая позволяет отличить исходные значения `NULL` от `NULL`, которые были добавлены при формировании общих и промежуточных итогов.

`GROUPING` возвращает битовую маску:

- `0` — `NULL` для исходного пустого значения.
- `1` — `NULL`, добавленный для промежуточного или общего итога.

Пример

```
SELECT
    column1,
    column2,
    column3,

    CASE GROUPING(
        column1,
        column2,
        column3,
    )
        WHEN 1  THEN "Subtotal: column1 and column2"
        WHEN 3  THEN "Subtotal: column1"
        WHEN 4  THEN "Subtotal: column2 and column3"
        WHEN 6  THEN "Subtotal: column3"
        WHEN 7  THEN "Grand total"
        ELSE         "Individual group"
    END AS subtotal,

    COUNT(*) AS rows_count

FROM my_table

GROUP BY
    ROLLUP(
        column1,
        column2,
        column3
    ),
    GROUPING SETS(
        (column2, column3),
        (column3)
        -- если добавить сюда ещё (column2), то в сумме
        -- эти ROLLUP и GROUPING SETS дали бы результат,
        -- аналогичный CUBE
    )
;
```

DISTINCT

Применение агрегатных функций только к уникальным значениям столбца.

> **Примечание**
>
> Применение `DISTINCT` к вычислимым значениям на данный момент не реализовано. С этой целью можно использовать подзапрос или выражение `GROUP BY ... AS ...`.

Пример

```
SELECT
  key,
  COUNT(DISTINCT value) AS count -- топ-3 ключей по количеству уникальных значений
FROM my_table
GROUP BY key
ORDER BY count DESC
LIMIT 3;
```

Также ключевое слово `DISTINCT` может использоваться для выборки уникальных строк через `SELECT DISTINCT`.

COMPACT

Наличие SQL хинта `COMPACT` непосредственно после ключевого слова `GROUP` позволяет более эффективно выполнять агрегацию в тех случаях, когда автору запроса заранее известно, что ни по одному из ключей агрегации не найдется

большого количества данных (порядка гигабайта или миллиона строк). Если это предположение не оправдается, то операция может упасть с ошибкой Out of Memory или начать работать значительно медленнее по сравнению с обычным GROUP BY.

В отличие от обычного GROUP BY, отключается стадия Map-side combiner и дополнительные Reduce для каждого поля с DISTINCT агрегацией.

Пример

```
SELECT
  key,
  COUNT(DISTINCT value) AS count -- топ-3 ключей по количеству уникальных значений
FROM my_table
GROUP /*+ COMPACT() */ BY key
ORDER BY count DESC
LIMIT 3;
```

GROUP BY ... HOP

`HOP` группирует данные по перекрывающимся временным окнам (hopping windows). Поддерживается как в аналитических запросах к таблицам, так и в потоковых запросах к топикам.

```
HOP(time_extractor, hop, interval, delay)
```

Где:

- `time_extractor` — SQL выражение типа `Timestamp`, определяющее время события. Из каждой входной строки вычисляется временная метка, по которой определяется принадлежность к окнам.
- `hop` — шаг между началами соседних окон в формате ISO 8601, например `"PT10S"` (10 секунд).
- `interval` — размер (длительность) каждого окна в формате ISO 8601, например `"PT30S"` (30 секунд).
- `delay` — задержка закрытия окна после его завершения в формате ISO 8601. Используется только в потоковых запросах (при работе с таблицами игнорируется). Для потоковых запросов рекомендуется использовать HoppingWindow с watermarks вместо `delay`.

Также доступны агрегатные функции `HOP_START()` и `HOP_END()`, которые возвращают временную метку начала и конца текущего окна типа `Timestamp` соответственно.

Описание

Разберём алгоритм на примере.

```
GROUP BY HOP(CAST(ts AS Timestamp), "PT10S", "PT30S", "PT20S")
```

В этом примере `CAST(ts AS Timestamp)` извлекает время события из столбца `ts`. Параметр `hop` равен 10 секундам, `interval` равен 30 секундам, `delay` равен 20 секундам.

Окна строятся по следующему правилу:

- Начала окон выравниваются по моментам, кратным `hop` (10 секунд), начиная с 0: 0, 10, 20 и так далее.
- Длительность каждого окна равна `interval` (30 секунд). Получаются окна: `[0; 30)`, `[10; 40)`, `[20; 50)` и так далее.
- Событие попадает во все окна, временной диапазон которых включает его время. Например, событие с временем 25 секунд попадает в окна `[0; 30)`, `[10; 40)` и `[20; 50)`.
- Окно считается завершённым при получении события с временной меткой не меньше, чем конец этого окна + `delay` (20 секунд). Например, окно `[10; 40)` закрывается при получении события с меткой 60 и более.

Аналитический `HOP` поверх таблицы

При работе с таблицами данные группируются по ключам `GROUP BY` (без учёта `HOP`), образуя группы строк (далее группы). Внутри каждой группы:

1. Строки сортируются по возрастанию `time_extractor`.
2. Каждая строка назначается в одно или несколько перекрывающихся окон.
3. На каждом окне вычисляются заданные агрегатные функции.

Параметр `delay` при аналитической обработке таблицы **не используется**: данные уже доступны целиком, порядок обхода задаётся сортировкой по `time_extractor`, а завершение окна определяется алгоритмом полного прохода по группе (см. правило закрытия в описании выше).

Потоковый `HOP` поверх топика

При работе с топиками данные группируются по ключам `GROUP BY` (без учёта `HOP`), образуя группы. Внутри каждой группы:

1. События обрабатываются в порядке, близком к возрастанию `time_extractor`. Допускаются небольшие отклонения от строгого порядка.
2. Каждое событие назначается в одно или несколько перекрывающихся окон.
3. На каждом окне вычисляются заданные агрегатные функции.

В потоковых запросах события могут приходить не в строгом хронологическом порядке. Параметр `delay` задаёт время ожидания после формального завершения окна: окно закрывается не сразу, а через `delay` секунд, чтобы дать задержавшимся событиям время на поступление. События, поступившие после закрытия окна, игнорируются.

Ограничения

`time_extractor` — это SQL выражение, зависящее только от входных значений столбцов, должно иметь тип `Timestamp`.

Для задания `hop`, `interval` и `delay` используется строковое выражение, соответствующее стандарту ISO 8601, например, `PT10S` — 10 секунд, `PT1M` — 1 минута. Это формат, который используется для конструирования встроенного типа `Interval` из строки.

Значения параметров `interval` и `delay` должны делиться на значение параметра `hop`. Это требование обеспечивает выравнивание границ окон: каждое окно начинается в момент, кратный `hop`, и заканчивается ровно через `interval`, что гарантирует равномерное покрытие временной оси без пропусков. Параметры `hop` и `interval` должны быть положительными.

Примеры

```
SELECT
    sensor_id,
    HOP_END() AS window_end,
    AVG(temperature) AS avg_temp,
    COUNT(*) AS event_count
FROM sensor_data
GROUP BY
    sensor_id,
    HOP(CAST(event_time AS Timestamp), "PT10S", "PT1M", "PT30S");
```

GROUP BY ... HoppingWindow

`HoppingWindow` группирует события по перекрывающимся временным окнам (hopping windows), аналогично GROUP BY HOP. Поддерживается как в аналитических запросах к таблицам, так и в потоковых запросах к топикам. Основное отличие от `HOP`: в потоковых запросах `HoppingWindow` использует механизм watermarks для определения момента закрытия окна вместо фиксированного параметра `delay`.

```
HoppingWindow(time_extractor, hop, interval)
```

Где:

- `time_extractor` — SQL выражение типа `Timestamp`, определяющее время события. Должно зависеть только от входных столбцов.
- `hop` — шаг (период сдвига) между началами соседних окон в формате ISO 8601, например `"PT10S"` (10 секунд).
- `interval` — размер (длительность) каждого окна в формате ISO 8601, например `"PT1M"` (1 минута). Значение `interval` должно делиться на `hop`, так как окна выравниваются по кратным интервалам шага.

Также доступны функции `HOP_START()` и `HOP_END()`, возвращающие временные метки начала и конца текущего окна.

Алгоритм построения окон совпадает с GROUP BY HOP: окна начинаются в моменты, кратные `hop`, и имеют длительность `interval`. Событие попадает во все окна, временной диапазон которых включает его время.

Аналитический `HoppingWindow` поверх таблицы

При работе с таблицами `HoppingWindow` выполняет группировку по временным окнам аналогично HOP, но без параметра `delay`, который при аналитическом использовании всегда игнорировался (данные в таблице уже отсортированы).

1. Входная таблица партиционируется по ключам группировки, указанным в `GROUP BY`, без учёта `HoppingWindow`.
2. Каждая партиция сортируется по возрастанию `time_extractor`.
3. Каждая партиция делится на перекрывающиеся подмножества событий (окна).
4. На каждом подмножестве вычисляются заданные агрегатные функции.

Потоковый `HoppingWindow` поверх топика

При работе с топиками `HoppingWindow` использует watermarks для определения момента закрытия окна. Окно закрывается, когда значение watermark не меньше конца окна. Это обеспечивает более точные результаты агрегации по сравнению с `HOP`, где окно закрывается по фиксированному `delay`.

1. Входной топик партиционируется по ключам группировки, указанным в `GROUP BY`, без учёта `HoppingWindow`.
2. В каждой партиции окно продвигается независимо от других.
3. События обрабатываются в порядке, близком к возрастанию `time_extractor`. Допускаются небольшие перестановки порядка входного потока.
4. Каждая партиция делится на перекрывающиеся подмножества событий (окна).
5. Окно закрывается при получении watermark, значение которого не меньше конца окна. После закрытия результат агрегации выдаётся.
6. События, поступившие после закрытия окна, не учитываются в результатах.

Для корректной работы `HoppingWindow` в потоковом режиме необходимо настроить watermark в секции WITH источника. Подробнее: Настройка.

Пример

Ниже — потоковое чтение из топика: в `SELECT` удобно выводить конец окна через `HOP_END()`. Для **таблиц** в аналитическом запросе чаще используют `HOP_START()` или `HOP_END()` в зависимости от того, какую границу окна нужно показать в результате; смысл окон при этом тот же, отличается только выбираемая метка.

```
SELECT
    key,
    HOP_END() AS window_end,
    COUNT(*) AS event_count
FROM
    my_topic
WITH (
    FORMAT = json_each_row,
    SCHEMA = (
        key String,
        event_time String
    ),
    WATERMARK = __ydb_write_time - Interval("PT5S")
```

```
)
GROUP BY
    key,
    HoppingWindow(__ydb_write_time, "PT10S", "PT1M");
```

HAVING

Фильтрация выборки `SELECT` по результатам вычисления агрегатных функций. Синтаксис аналогичен конструкции `WHERE`.

Пример

```
SELECT
    key
FROM my_table
GROUP BY key
HAVING COUNT(value) > 100;
```

JOIN

Позволяет объединить несколько источников данных (подзапросов или таблиц) по равенству значений указанных столбцов или выражений (ключей `JOIN`).

Синтаксис

```
SELECT ...    FROM table_1
-- первый шаг объединения:
  <Join_Type> JOIN table_2 <Join_Condition>
   -- исходная выборка -- записи в таблице table_1
   -- присоединяемая выборка -- записи в таблице table_2
-- следующий шаг объединения:
  <Join_Type> JOIN table_n <Join_Condition>
   -- исходная выборка -- результат объединения на предыдущем шаге
   -- присоединяемая выборка -- записи в таблице table_n
-- могут быть следующие шаги объединения
...
WHERE  ...
```

На каждом шаге объединения по заданным правилам определяются соответствия между строками исходной и присоединяемой выборок данных, и формируется новая выборка, в которую попадают все сочетания подошедших под условия объединения строк.

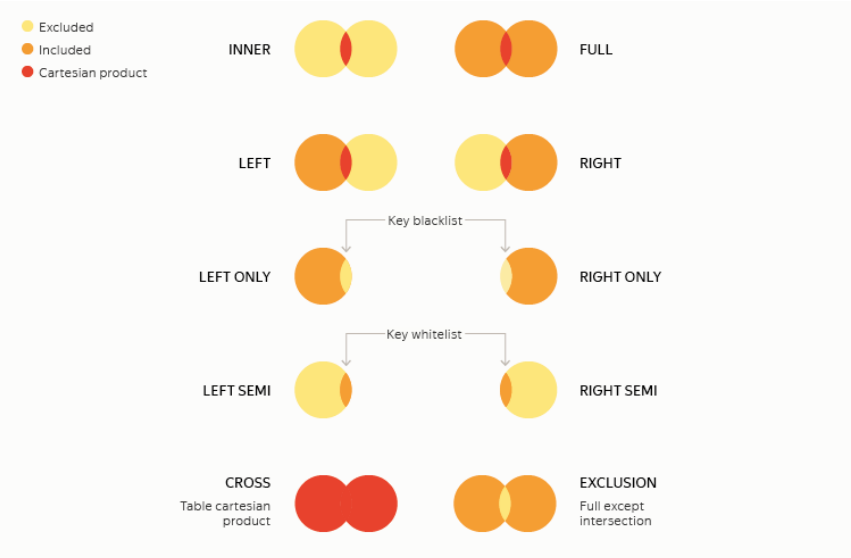
Внимание

Так как колонки в YQL идентифицируются по именам, и в выборке не может быть двух колонок с одинаковыми именами, `SELECT * FROM ... JOIN ...` не может быть исполнен при наличии колонок с одинаковыми именами в объединяемых таблицах.

Типы объединения (Join_Type)

- `INNER` (по умолчанию) — Строки объединяемых выборок, для которых не найдено соответствие ни с одной строкой с другой стороны, не попадут в результат.
- `LEFT` - При отсутствии значения в присоединяемой выборке включает строку в результат со значениями колонок из исходной выборки, оставляя пустыми (`NULL`) колонки присоединяемой выборки
- `RIGHT` - При отсутствии значения в исходной выборке включает строку в результат со значениями колонок из присоединяемой выборки, оставляя пустыми (`NULL`) колонки исходной выборки
- `FULL` = `LEFT` + `RIGHT`
- `LEFT/RIGHT SEMI` — одна сторона выступает как белый список (whitelist) ключей, её значения недоступны. В результат включаются столбцы только из одной таблицы, декартового произведения не возникает;
- `LEFT/RIGHT ONLY` — вычитание множеств по ключам (blacklist). Практически эквивалентно добавлению условия `IS NULL` на ключ противоположной стороны в обычном `LEFT/RIGHT`, но, как и в `SEMI`, нет доступа к значениям;
- `CROSS` — декартово произведение двух таблиц целиком без указания ключевых колонок, секция с `ON/USING` явно не пишется;
- `EXCLUSION` — обе стороны минус пересечение.

Используйте `LEFT SEMI JOIN` вместо коррелированного подзапроса `EXISTS`, а `LEFT ONLY JOIN` — вместо коррелированного `NOT EXISTS`. Подробнее см. в разделе Коррелированные подзапросы, EXISTS и NOT EXISTS.



Примечание

`NULL` является особым значением, которое ничему не равно. Таким образом, `NULL` с двух сторон не считаются равными друг другу. Это избавляет от неоднозначности в некоторых типах `JOIN`, а также от гигантского декартового произведения, которое часто возникает в противном случае.

Условия объединения (Join_Condition)

Для `CROSS JOIN` условие объединения не указывается. В результат попадет декартово произведение исходной и присоединяемой выборок, то есть сочетание всех со всеми. Количество строк в результирующей выборке будет произведением количества строк исходной и присоединяемой выборок.

Для любых других типов объединения необходимо указать условие одним из двух способов:

1. `USING (column_name)`. Используется при наличии в исходной и присоединяемой выборках одноименной колонки, равенство значений в которой является условием объединения.
2. `ON (equality_conditions)`. Позволяет задать условие равенства значений колонок или выражений над колонками исходной и присоединяемой выборок, или несколько таких условий, объединенных по `and`.

Примеры

```
SELECT    a.value as a_value, b.value as b_value
FROM      a_table AS a
FULL JOIN b_table AS b USING (key);
```

```
SELECT    a.value as a_value, b.value as b_value
FROM      a_table AS a
FULL JOIN b_table AS b ON a.key = b.key;
```

```
SELECT     a.value as a_value, b.value as b_value, c.column2
FROM       a_table AS a
CROSS JOIN b_table AS b
LEFT  JOIN c_table AS c ON c.ref = a.key and c.column1 = b.value;
```

Для исключения необходимости в полном сканировании правой присоединяемой таблицы может использоваться вторичный индекс над колонками, входящими в условие соединения. Обращение ко вторичному индексу должно быть указано в явном виде, в формате `JOIN table_name VIEW index_name AS table_alias`.

Например, создание индекса для использования в условии соединения:

```
ALTER TABLE b_table ADD INDEX b_index_ref GLOBAL ON(ref);
```

Использование созданного индекса:

```
SELECT    a.value as a_value, b.value as b_value
FROM      a_table AS a
INNER JOIN b_table VIEW b_index_ref AS b ON a.ref = b.ref;
```

OVER, PARTITION BY и WINDOW

Механизм оконных функций, появившийся в стандарте SQL:2003 и расширенный в стандарте SQL:2011, позволяет выполнять вычисления над набором строк таблицы, который некоторым образом соотносится с текущей строкой.

В отличие от агрегатных функций, при этом не происходит группировка нескольких строк в одну: после применения оконных функций число строк в результирующей таблице всегда совпадает с числом строк в исходной.

При наличии в запросе одновременно агрегатных и оконных функций сначала производится группировка и вычисляются значения агрегатных функций. Вычисленные значения агрегатных функций могут использоваться в качестве аргументов оконных (но не наоборот). Порядок, в котором вычисляются оконные функции относительно других элементов запроса, описан в разеделе SELECT.

Синтаксис

Общий синтаксис вызова оконной функции имеет вид

```
function_name([expression [, expression ...]]) OVER (window_definition)
```

или

```
function_name([expression [, expression ...]]) OVER window_name
```

Здесь `window_name` (*имя окна*) – произвольный идентификатор, уникальный в рамках запроса, `expression` – произвольное выражение не содержащее вызова оконных функций.

В запросе каждому имени окна должно быть сопоставлено *определение окна* (`window_definition`):

```
SELECT
    F0(...) OVER (window_definition_0),
    F1(...) OVER w1,
    F2(...) OVER w2,
    ...
FROM my_table
WINDOW
    w1 AS (window_definition_1),
    ...
    w2 AS (window_definition_2)
;
```

Здесь `window_definition` записывается в виде

```
[ PARTITION BY (expression AS column_identifier | column_identifier) [, ...] ]
[ ORDER BY expression [ASC | DESC] ]
[ frame_definition ]
```

Необязательное *определение рамки* (`frame_definition`) может быть задано одним из двух способов:

- `ROWS frame_begin`
- `ROWS BETWEEN frame_begin AND frame_end`

Начало рамки (`frame_begin`) и *конец рамки* (`frame_end`) задаются одним из следующих способов:

- `UNBOUNDED PRECEDING`
- `offset PRECEDING`
- `CURRENT ROW`
- `offset FOLLOWING`
- `UNBOUNDED FOLLOWING`

Здесь *смещение рамки* (`offset`) – неотрицательный числовой литерал. Если конец рамки не задан, то подразумевается `CURRENT ROW`.

Все выражения внутри определения окна не должны содержать вызовов оконных функций.

Алгоритм вычисления

Разбиение

Указание `PARTITION BY` группирует строки исходной таблицы в *разделы*, которые затем обрабатываются независимо друг от друга.
Если `PARTITION BY` не указан, то все строки исходной таблицы попадают в один раздел. Указание `ORDER BY` определяет порядок строк в разделе.
В `PARTITION BY`, как и в GROUP BY можно использовать алиасы и SessionWindow.

При отсутствии `ORDER BY` порядок строк в разделе не определен.

Рамка

Определение рамки `frame_definition` задает множество строк раздела, попадающих в *рамку окна* связанную с текущей строкой.

В режиме `ROWS` (в YQL пока поддерживается только он) в рамку окна попадают строки с указанными смещениями относительно текущей строки раздела. Например, для `ROWS BETWEEN 3 PRECEDING AND 5 FOLLOWING` в рамку окна попадут три строки перед текущей, текущая строка и пять строк после текущей строки.

Множество строк в рамке окна может меняться в зависимости от того, какая строка является текущей. Например, для первой строки раздела в рамку окна `ROWS BETWEEN 3 PRECEDING AND 1 PRECEDING` не попадет ни одной строки.

Указание `UNBOUNDED PRECEDING` в качестве начала рамки означает "от первой строки раздела", `UNBOUNDED FOLLOWING` в качестве конца рамки – "до последней строки раздела", `CURRENT ROW` – "от/до текущей строки".

Если `определение_рамки` не указано, то в множество строк попадающих в рамку окна определяется наличием `ORDER BY` в `определении_окна`.
А именно, при наличии `ORDER BY` неявно подразумевается `ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`, а при отсутствии – `ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`.

Далее, в зависимости от конкретной оконной функции производится ее вычисление либо на множестве строк раздела, либо на множестве строк рамки окна.

Список доступных оконных функций

Примеры

```
SELECT
    COUNT(*) OVER w AS rows_count_in_window,
    some_other_value -- доступ к текущей строке
FROM `my_table`
WINDOW w AS (
    PARTITION BY partition_key_column
    ORDER BY int_column
);
```

```
SELECT
    LAG(my_column, 2) OVER w AS row_before_previous_one
FROM `my_table`
WINDOW w AS (
    PARTITION BY partition_key_column
);
```

```
SELECT
    -- AVG (как и все агрегатные функции, используемые в качестве оконных)
    -- вычисляется на рамке окна
    AVG(some_value) OVER w AS avg_of_prev_current_next,
    some_other_value -- доступ к текущей строке
FROM my_table
WINDOW w AS (
    PARTITION BY partition_key_column
    ORDER BY int_column
    ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING
);
```

```
SELECT
    -- LAG не зависит от положения рамки окна
    LAG(my_column, 2) OVER w AS row_before_previous_one
FROM my_table
WINDOW w AS (
    PARTITION BY partition_key_column
    ORDER BY my_column
);
```

Особенности реализации

- Функции, вычисляемые на рамке окна `ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING` либо `ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`, реализованы эффективно (не требуют дополнительной памяти и вычисляются на разделе за O(размер раздела)).
- Для рамки окна `ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING` есть возможность выбрать стратегию выполнения в оперативной памяти, указав хинт `COMPACT` после ключевого слова `PARTITION`.

  Например: `PARTITION /*+ COMPACT() */ BY key` или `PARTITION /*+ COMPACT() */ BY ()` (в случае если `PARTITION BY` изначально отсутствовал).

  При наличии хинта `COMPACT` потребуется дополнительная память в размере O(размер раздела), но при этом не возникнет дополнительной `JOIN` операции.
- Если рамка окна не начинается с `UNBOUNDED PRECEDING`, то для вычисления оконных функций на таком окне потребуется дополнительная память в размере O(максимальное расстояние от границ окна до текущей строки), а время вычисления будет равно O(число_строк_в_разделе * размер_окна).
- Для рамки окна, начинающейся с `UNBOUNDED PRECEDING` и заканчивающейся на `N`, где `N` не равен `CURRENT ROW` или `UNBOUNDED FOLLOWING`, потребуется дополнительная память в размере O(N), а время вычисления будет O(N * число_строк_в_разделе).
- Функции `LEAD(expr, N)` и `LAG(expr, N)` всегда потребуют O(N) памяти.

Учитывая вышесказанное, запрос с `ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING` по возможности стоит переделать в `ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`, поменяв порядок сортировки в `ORDER BY` на обратный.

DISTINCT

Выбор уникальных строк.

> **Примечание**
>
> Применение `DISTINCT` к вычислимым значениям на данный момент не реализовано. С этой целью можно использовать подзапрос или выражение `GROUP BY ... AS ...`.

Пример

```
SELECT DISTINCT value -- только уникальные значения из таблицы
FROM my_table;
```

Также ключевое слово `DISTINCT` может использоваться для применения агрегатных функций только к уникальным значениям. Подробнее см. в документации по GROUP BY.

UNIQUE DISTINCT hints

Непосредственно после `SELECT` возможно добавить SQL хинты `unique` или `distinct` которые заявляют, что эта проекция порождает данные, содержащие уникальные значения в указанном наборе колонок строковой или колоночной таблицы. Это может, использоваться для оптимизации следующих подзапросов, выполняющихся над этой проекцией, или для записи в мета-атрибуты таблицы при `INSERT` (временно не поддерживается для колоночных таблиц).

- Колонки указываются в значениях хинта через пробел.
- Если набор колонок не задан, значит уникальность распространяется на полный набор колонок этой проекции.
- `unique` - означает уникальные либо `null` значения. По стандарту SQL каждый null уникален: NULL = NULL -> NULL
- `distinct` - означает полностью уникальные значение включая `null`: NULL IS DISTINCT FROM NULL -> FALSE
- Можно указать несколько наборов колонок в нескольких хинтах у одной проекции.
- Если хинт содержит колонку, которой нет в проекции, он будет проигнорирован.

Примеры

```
SELECT /*+ unique() */ * FROM Input;
SELECT /*+ distinct() */ * FROM Input;

SELECT /*+ distinct(key subkey) */ * FROM Input;
SELECT /*+ unique(key) distinct(subkey value) */ * FROM Input;

-- Missed column - ignore hint.
SELECT /*+ unique(subkey value) */ key, value FROM Input;
```

Объединение результатов подзапросов (UNION)

UNION

Объединение результатов нескольких подзапросов с удалением дубликатов.
Поведение идентично последовательному исполнению `UNION ALL` и `SELECT DISTINCT *`.
См. UNION ALL для информации о деталях поведения.

Примеры

```
SELECT key FROM T1
UNION
SELECT key FROM T2 -- возвращает таблицу различных ключей, лежащих хотя бы в одной из исходных таблиц
```

UNION ALL

Конкатенация результатов нескольких `SELECT` (или подзапросов).

Поддерживаются два режима выполнения `UNION ALL` – по именам колонок (режим по умолчанию) и по позициям колонок (соответствует стандарту ANSI SQL и включается через соответствующую PRAGMA).

В режиме "по именам" результирующая схема данных выводится по следующим правилам:

- в результирующую таблицу включаются все колонки, которые встречались хоть в одной из входных таблиц;
- если колонка присутствовала не во всех входных таблицах, то ей автоматически присваивается опциональный тип данных (допускающий значение `NULL`);
- если колонка в разных входных таблицах имела разные типы, то выводится общий тип (наиболее широкий);
- если колонка в разных входных таблицах имела разнородный тип, например строку и число, то это считается ошибкой.

Порядок выходных колонок в этом режиме выводится как наибольший общий префикс порядка входов, после чего следуют все остальные колонки в алфавитном порядке.
Если наибольший общий префикс пуст (в том числе и из-за отсутствия порядка на одном из входов), то порядок выхода не определен.

В режиме "по позициям" результирующая схема данных выводится по следующим правилам:

- число колонок во всех входах должно быть одинаковым
- порядок колонок во всех входах должен быть определен
- имена результирующих колонок совпадают с именами колонок первой таблицы
- тип результирующих колонок выводится как общий (наиболее широкий) тип из типов входных колонок стоящих на одинаковых позициях

Порядок выходных колонок в этом режиме совпадает с порядком колонок первого входа.

Примеры

```
SELECT 1 AS x
UNION ALL
SELECT 2 AS y
UNION ALL
SELECT 3 AS z;
```

В результате выполнения данного запроса в режиме по-умолчанию будет сформирована выборка с тремя колонками x, y, и z.
При включенной `PRAGMA PositionalUnionAll;` в выборке будет одна колонка x.

```
PRAGMA PositionalUnionAll;

SELECT 1 AS x, 2 as y
UNION ALL
SELECT * FROM AS_TABLE([<|x:3, y:4|>]); -- ошибка: порядок колонок в AS_TABLE не определен
```

```
SELECT * FROM T1
UNION ALL
(SELECT * FROM T2 ORDER BY key LIMIT 100); -- при отсутствии скобок ORDER BY/LIMIT применится к результату вс
его UNION ALL
```

VIEW (INDEX)

Чтобы сделать запрос `SELECT` по вторичному индексу строковой таблицы, используйте конструкцию:

```
SELECT *
    FROM TableName VIEW IndexName
    WHERE …
```

> **Важно**
>
> Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Примеры

Выбрать все поля из строковой таблицы `series` по индексу `views_index` с условием `views >= someValue`:

```
SELECT series_id, title, info, release_date, views, uploaded_user_id
    FROM series VIEW views_index
    WHERE views >= someValue
```

Сделать `JOIN` строковых таблиц `series` и `users` с заданным полем `userName` по индексам `users_index` и `name_index` соответственно:

```
SELECT t1.series_id, t1.title
    FROM series VIEW users_index AS t1
    INNER JOIN users VIEW name_index AS t2
    ON t1.uploaded_user_id == t2.user_id
    WHERE t2.name == userName;
```

VIEW (Векторный индекс)

Для выполнения запроса `SELECT` с использованием векторного индекса в строчно-ориентированной таблице используйте следующий синтаксис:

```
PRAGMA ydb.KMeansTreeSearchTopSize = "10";

SELECT ...
    FROM TableName VIEW IndexName
    WHERE ...
    ORDER BY Knn::DistanceFunction(...)
    LIMIT ...
```

```
PRAGMA ydb.KMeansTreeSearchTopSize = "10";

SELECT ...
    FROM TableName VIEW IndexName
    WHERE ...
    ORDER BY Knn::SimilarityFunction(...) DESC
    LIMIT ...
```

Принципы работы и настройки векторного индекса подробно описаны в отдельной статье Векторные индексы.

Примечание

Векторный индекс поддерживает функцию расстояния или сходства расширения Knn, выбранную при создании индекса.

Векторный индекс не будет автоматически выбран оптимизатором, поэтому его нужно указывать явно с помощью выражения `VIEW IndexName`.

Если не использовать выражение `VIEW`, запрос выполнит полное сканирование таблицы с попарным сравнением векторов. Рекомендуется проверять оптимальность написанного запроса, используя анализ плана выполнения запроса. В частности, следует следить за отсутствием полного сканирования (full scan) основной таблицы.

Важно

При накоплении значительного объёма изменений в таблицах с векторным индексом полнота или производительность поиска может ухудшиться. Подробности смотрите в разделе Обновление векторных индексов.

KMeansTreeSearchTopSize

Векторный поиск по индексу основан на приближённом алгоритме (ANN, Approximate Nearest Neighbors). Это значит, что результат поиска по векторному индексу может отличаться от результата поиска при полном сканировании таблицы.

Полнота поиска по индексу может быть отрегулирована параметром: `PRAGMA ydb.KMeansTreeSearchTopSize`.

Данный параметр задаёт максимальное число сканируемых кластеров, ближайших к запрашиваемому вектору, на каждом уровне дерева поиска.

Задавайте значение явно при настройке баланса между полнотой поиска и стоимостью запроса.

Значение по умолчанию равно 4 для индекса с перекрывающимися кластерами (`overlap_clusters > 1`) и 10 для индекса без перекрытия. При значении 1 на каждом уровне сканируется только один ближайший кластер: это уменьшает стоимость запроса за счёт снижения полноты. Увеличение значения позволяет исследовать больше кластеров и может повысить полноту, но требует больше чтений и сравнений векторов. Например:

```
PRAGMA ydb.KMeansTreeSearchTopSize="10";

SELECT *
    FROM TableName VIEW IndexName
    ORDER BY Knn::CosineDistance(embedding, $target)
    LIMIT 10
```

Фильтрация по префиксу векторного индекса

Векторные индексы с фильтрацией поддерживают условия равенства, `IN` и `OR` по колонкам префикса. Для индекса на `(user, embedding)` условие `WHERE user IN ("john", "jane")` выбирает несколько категорий. Поддерживается и начальная часть составного префикса: индекс на `(user, article_id, embedding)` допускает `WHERE user = "john"`.

Итоговые `ORDER BY` и `LIMIT` применяются к объединённым кандидатам. Число исследуемых кластеров масштабируется с количеством подходящих групп префикса.

Примеры

- Выбор всех полей из таблицы `series` с использованием векторного индекса `views_index`, созданного для `embedding` с мерой близости "косинусное расстояние":

  ```
  PRAGMA ydb.KMeansTreeSearchTopSize="10";
  SELECT series_id, title, info, release_date, views, uploaded_user_id, Knn::CosineSimilarity(embedding,
  $target) as similarity
      FROM series VIEW views_index
      ORDER BY similarity DESC
      LIMIT 10
  ```

- Выбор всех полей из таблицы `series` с использованием векторного индекса с фильтрацией `views_filtered_index`, созданного для `embedding` с мерой близости "косинусное расстояние" и с ускорением фильтрации по колонке `release_date`:

```
PRAGMA ydb.KMeansTreeSearchTopSize="10";
SELECT series_id, title, info, release_date, views, uploaded_user_id, Knn::CosineSimilarity(embedding,
$target) as similarity
    FROM series VIEW views_filtered_index
    WHERE release_date = "2025-03-31"
    ORDER BY similarity DESC
    LIMIT 10
```

VIEW (Полнотекстовый индекс)

Для выполнения запроса `SELECT` с использованием полнотекстового индекса в строковой таблице используйте выражение `VIEW`:

```
SELECT ...
FROM TableName VIEW IndexName
WHERE FulltextMatch(TextColumn, "query")
ORDER BY ...
```

Примечание

Полнотекстовый индекс не будет автоматически выбран оптимизатором, поэтому его нужно указывать явно с помощью `VIEW IndexName`.

Функции полнотекстового поиска (`FulltextMatch`, `FulltextScore`) требуют `VIEW`. Если `VIEW` не используется, запрос завершится с ошибкой.

Несколько предикатов `FulltextMatch` с настройками по умолчанию можно объединять через `AND`. В одном чтении нельзя смешивать `FulltextMatch` и `FulltextScore` или использовать несколько различных предикатов `FulltextScore`. `FulltextMatch` / `FulltextScore` нельзя использовать под `OR` или `NOT`.
Для доступа к индексу по релевантности требуется ограничение `FulltextScore(...) > 0` в `WHERE`.

Описание функций см. в разделе Полнотекстовый поиск, включая
FulltextMatch и
FulltextScore.

FulltextMatch

FulltextMatch(text, query) фильтрует строки по совпадению текста с полнотекстовым запросом:

```
SELECT id, title
FROM articles VIEW ft_idx
WHERE FulltextMatch(body, "машинное обучение")
LIMIT 20;
```

Только первые два аргумента могут быть позиционными. Дополнительные параметры нужно передавать как **именованные аргументы**:

- `Mode` (String): вид запроса:
  - `Keywords` (по умолчанию) — текст запроса разбивается на отдельные термы; логика их объединения определяется `DefaultOperator`
  - `Wildcard` — поиск с подстановочными символами: `%` заменяет любую подстроку, `_` — один символ (аналогично `LIKE`); требует N-граммного индекса
- `DefaultOperator` (String): оператор объединения термов в режиме `Keywords`:
  - `And` (по умолчанию) — все термы запроса должны присутствовать в тексте
  - `Or` — термы делятся на **обязательные** (с префиксом `+`, должны присутствовать в каждом результате) и **необязательные** (без префикса, учитываются в `MinimumShouldMatch`). Без префикса `+` все термы считаются необязательными, и достаточно совпадения хотя бы одного (см. пример)
- `MinimumShouldMatch` (String): минимальное число **необязательных** термов, которые должны совпасть при `DefaultOperator = "Or"` — задаётся как абсолютное число (например, `"3"`) или процент от числа необязательных термов (например, `"50%"`). Обязательные (`+`) термы в счёт не входят

Режим `Wildcard` и шаблоны `%` / `_` (требуются N-граммы)

Если индекс создан с фильтрацией N-грамм, можно использовать шаблоны с `%` и `_` (по аналогии с `LIKE`).
Чтобы явно указать такой режим, передайте именованный аргумент `"Wildcard" AS Mode`:

```
SELECT id, title
FROM articles VIEW ft_idx
WHERE FulltextMatch(body, "маш% обу%ние", "Wildcard" AS Mode)
LIMIT 20;
```

LIKE / ILIKE (используют полнотекстовый индекс)

Для полнотекстовых индексов с N-граммами предикаты `LIKE` / `ILIKE` используют индекс для отбора кандидатов, затем проверяют шаблон по исходному тексту. `LIKE` учитывает регистр, а `ILIKE` выполняет регистронезависимую проверку:

```
SELECT id, title
FROM articles VIEW ft_idx
WHERE body ILIKE "%обуч%ние%"
LIMIT 20;
```

Для полнотекстового индекса с фильтрацией в `WHERE` требуется предикат равенства для каждой колонки префикса.

FulltextScore

FulltextScore(text, query) возвращает оценку релевантности (BM25) и может использоваться для ранжирования.
Ранжирование требует индекса типа fulltext_relevance.

```
SELECT id, title, FulltextScore(body, "машинное обучение") AS relevance
FROM articles VIEW ft_idx
```

```
WHERE FulltextScore(body, "машинное обучение") > 0
ORDER BY relevance DESC
LIMIT 10;
```

Дополнительные параметры

Дополнительные параметры нужно передавать как **именованные аргументы**:

- `DefaultOperator` (String): оператор объединения термов — `And` (по умолчанию, все термы должны присутствовать) или `Or` (термы делятся на **обязательные** с префиксом `+` и **необязательные** без префикса; каждый обязательный должен присутствовать, а из необязательных должны совпасть не менее `MinimumShouldMatch`)
- `MinimumShouldMatch` (String): при `DefaultOperator = "Or"` — минимальное число **необязательных** термов, которые должны совпасть; задаётся как абсолютное число (например, `"2"`) или процент от числа необязательных термов (например, `"50%"`). Обязательные (`+`) термы в счёт не входят
- `K1` (Double или Float): параметр насыщения частоты терма в BM25 — определяет, насколько сильно влияет повторное появление терма на итоговую оценку; по умолчанию `1.2`
- `B` (Double или Float): параметр нормализации длины документа в BM25 — `0.0` отключает нормализацию, `1.0` полностью нормализует по длине документа; по умолчанию `0.75`

Пример:

```
SELECT id, FulltextScore(body, "машинное обучение", "Or" AS DefaultOperator, "50%" AS MinimumShouldMatch) AS
relevance
FROM articles VIEW ft_idx
WHERE FulltextScore(body, "машинное обучение", "Or" AS DefaultOperator, "50%" AS MinimumShouldMatch) > 0
ORDER BY relevance DESC;
```

Примечание

Только первые два аргумента `FulltextMatch` / `FulltextScore` могут быть позиционными. Для дополнительных параметров используйте именованные аргументы.

Выражение `FulltextScore(...)` повторяется в `SELECT` и `WHERE` целиком — YQL, как и стандартный SQL, вычисляет `WHERE` раньше `SELECT`, поэтому псевдонимы из `SELECT` недоступны в `WHERE`. Оба вхождения должны быть идентичны.

VIEW (JSON-индекс)

Для выполнения запроса `SELECT` над строковой таблицей с явным использованием JSON-индекса используйте выражение `VIEW`:

```
SELECT ...
FROM documents VIEW json_idx
WHERE <предикат на основе JSON_EXISTS / JSON_VALUE>
ORDER BY ...
```

В примере запроса выше `documents` — это имя таблицы, содержащей колонку типа `Json` или `JsonDocument`, а `json_idx` — имя созданного над этой колонкой JSON-индекса.

Если предикат не поддерживается для выполнения через JSON-индекс, запрос с явным указанием выражения `VIEW` завершается ошибкой на этапе компиляции. Без выражения `VIEW` оптимизатор не сможет выбрать этот индекс для такого предиката, и запрос будет выполнен другим способом (например, выбором другого индекса, либо полным сканированием основной таблицы).

Примечание

JSON-индекс может быть выбран оптимизатором автоматически при условии соответствия предиката требованиям для использования индекса. Для отладки и гарантированного использования индекса указывайте его явно с помощью `VIEW IndexName`.

Описание поддерживаемых предикатов см. в разделе Поддерживаемые предикаты.

Автоматический выбор индекса

Если предикат `WHERE` содержит вызовы `JSON_EXISTS` / `JSON_VALUE` по индексированной JSON-колонке, оптимизатор может задействовать JSON-индекс без явного указания `VIEW`. Автоматический выбор подчиняется следующим правилам:

- JSON-индекс рассматривается оптимизатором с наименьшим приоритетом — это запасной вариант, который выбирается только после того, как остальные способы доступа исключены.
- JSON-индекс не выбирается, если запрос уже может быть обслужен первичным ключом или другим, более специфичным вторичным индексом.
- Явное указание `VIEW` переопределяет решение оптимизатора и принудительно включает использование индекса.
- Все индексируемые подвыражения в одном запросе должны ссылаться на одну и ту же индексированную JSON-колонку. Если объединить через `AND` / `OR` предикаты по разным индексированным колонкам, автоматический выбор не применяется.

Примечание

Автоматический выбор JSON-индекса работает на основе системы правил (rule-based): он использует структуру запроса и метаданные схемы, без статистики данных. Логика выбора может измениться в будущих версиях YDB, поэтому для гарантированного использования индекса указывайте его явно через `VIEW`.

JSON_EXISTS

JSON_EXISTS(doc, jsonpath) проверяет существование пути или значения внутри JSON-документа:

```
SELECT id, payload
FROM documents VIEW json_idx
WHERE JSON_EXISTS(payload, '$.user.id');
```

JSON-индекс поддерживает практически весь синтаксис JsonPath, за исключением вложенных предикатов и булевых выражений на уровне контекстного объекта (`$`) в `JSON_EXISTS`. Более сложный пример поддерживаемого выражения:

```
SELECT id, payload
FROM documents VIEW json_idx
WHERE JSON_EXISTS(payload, '$.user ? (@.id > 100)');
```

JSON_VALUE

JSON_VALUE(doc, jsonpath RETURNING) извлекает скалярное значение по пути JsonPath и возвращает его в заданном типе. Для использования JSON-индекса секция `RETURNING <type>` обязательна:

```
SELECT id, payload
FROM documents VIEW json_idx
WHERE JSON_VALUE(payload, '$.user.name' RETURNING Utf8) = "Charlie"u;
```

При проверке условий равенства либо вхождения значения в заданный список (`IN`) поиск в индексе осуществляется по токену «путь + значение», что обеспечивает наибольшую селективность. При других сравнениях (`!=` , `<` , `>=` , `BETWEEN` и др.) используется только токен пути, а итоговая точность сравнения обеспечивается пост-фильтром.

Параметры

Поддерживаются три способа передачи параметров запроса:

1. Прямое сравнение результата `JSON_VALUE` с параметром:

   ```
   DECLARE $id AS Int64;
   SELECT * FROM documents VIEW json_idx
   ```

```
WHERE JSON_VALUE(payload, '$.owner_id' RETURNING Int64) = $id;
```

2. Проверка наличия результата в списке значений:

```
DECLARE $tags AS List<Utf8>;
SELECT * FROM documents VIEW json_idx
WHERE JSON_VALUE(payload, '$.tag' RETURNING Utf8) IN $tags;
```

3. Передача параметра в JsonPath через секцию `PASSING`:

```
DECLARE $v AS Int64;
SELECT * FROM documents VIEW json_idx
WHERE JSON_EXISTS(payload, '$.k ? (@ == $v)' PASSING $v AS v);
```

Комбинации AND и OR

`JSON_EXISTS` и `JSON_VALUE` над одной JSON-колонкой могут объединяться в одном `WHERE` операторами `AND` и `OR`:

```
SELECT id, payload
FROM documents VIEW json_idx
WHERE JSON_VALUE(payload, '$.user.name' RETURNING Utf8) = "Charlie"u
  AND JSON_VALUE(payload, '$.user.id' RETURNING Int64) BETWEEN 100 AND 200;
```

Примечание

Выполнение выборки через JSON-индекс может использовать только подмножество выражений на основе `JSON_EXISTS` и `JSON_VALUE` (см. Поддерживаемые предикаты). Условия, не подпадающие под эти правила (например, отрицания, сравнения с другой колонкой, сравнение двух `JSON_VALUE` из разных колонок), не индексируются; в `AND` они переходят в пост-фильтр, в `OR` приводят к отказу от использования индекса по всей группе условий, объединённой через `OR`.

Гибридный поиск (HybridRank)

Гибридный поиск объединяет полнотекстовое и векторное ранжирование в едином результате. Гибридный запрос — это `SELECT` по основной таблице, у которого ключ `ORDER BY` представляет собой единственный вызов `HybridRank`:

```
PRAGMA ydb.KMeansTreeSearchTopSize = "10";

$queryText = "машинное обучение";
$queryVector = Knn::ToBinaryStringFloat([0.1, 0.2, 0.3, 0.4]);

SELECT id, title
FROM documents
ORDER BY HybridRank(
    FullTextScore(text, $queryText),
    Knn::CosineDistance(embedding, $queryVector))
LIMIT 10;
```

> **Примечание**
>
> Гибридный запрос читает данные из основной таблицы и **не** использует `VIEW IndexName`: индекс каждой ветви определяется по соответствующему аргументу `HybridRank`.
>
> Каждой ветви `FullTextScore` требуется готовый индекс fulltext_relevance. Каждой ветви `Knn` требуется готовый индекс vector_kmeans_tree с совместимой метрикой. Оба типа индексов могут иметь префиксы; см. выбор индексов и фильтрация. Полнота векторной ветви управляется параметром KMeansTreeSearchTopSize.

HybridRank

Общая форма функции:

```
HybridRank(
    score1, score2 [, ... scoreN]        -- 2+ оценивающих выражения: FullTextScore(...) или Knn::<Distance|Similarity>(...)
    [, "rrf" | "linear"  AS Mode]        -- способ объединения (по умолчанию "rrf")
    [, (w1, w2, ...)     AS Weights]     -- веса ветвей (по умолчанию 1.0)
    [, k                 AS K]           -- константа RRF (по умолчанию 60.0)
    [, true | false      AS Normalize]   -- только для режима "linear": min-max нормализация (по умолчанию true)
    [, (idx1, idx2, ...) AS Indexes]     -- имена индексов по ветвям (по умолчанию: автоопределение)
    [, (lim1, lim2, ...) AS Limits]      -- размеры пулов кандидатов по ветвям (по умолчанию: LIMIT * 10)
    [, ($x) -> {...}     AS RankLambda]  -- пользовательское объединение по рангам ветвей (заменяет встроенный Mode)
    [, ($x) -> {...}     AS ScoreLambda] -- пользовательское объединение по исходным оценкам ветвей (заменяет встроенный Mode)
)
```

`HybridRank` может использоваться только как единственный ключ сортировки в секции `ORDER BY`. Он принимает **два или более оценивающих выражения** (позиционно), за которыми следуют необязательные **именованные аргументы**.

Каждое оценивающее выражение — это одна *ветвь* объединения, классифицируемая по своей форме:

- выражение FullTextScore(text, query) — **полнотекстовая** ветвь, разрешается через индекс `fulltext_relevance` по колонке `text`;
- Knn-расстояние или сходство по колонке (например, `Knn::CosineDistance(embedding, $queryVector)` или `Knn::CosineSimilarity(embedding, $queryVector)`) — **векторная** ветвь, разрешается через индекс `vector_kmeans_tree` по этой колонке.

Порядок аргументов не имеет значения, а ветвей может быть больше двух (например, полнотекстовая ветвь и две векторные). Каждая ветвь извлекает собственный пул кандидатов. Объединение рассматривает все эти пулы и возвращает каждый документ один раз, в том числе документы, найденные только одной ветвью. Значение `NULL` в оцениваемой колонке исключает документ из соответствующей ветви, но другая ветвь всё ещё может его найти.

Используйте непосредственный вызов `FullTextScore` как оценивающий аргумент. Выражения вида `2 * FullTextScore(...)` отклоняются; для изменения вклада используйте `Weights` или `ScoreLambda`. Вызов поддерживает именованные параметры `DefaultOperator`, `MinimumShouldMatch`, `K1` и `B`, описанные в FullTextScore. `K1` и `B` принимают литералы и параметры типа `Float` или `Double`; `DefaultOperator` и `MinimumShouldMatch` принимают строковые литералы и параметры.

Поддерживаются векторные функции `Knn::CosineDistance`, `Knn::CosineSimilarity`, `Knn::ManhattanDistance`, `Knn::EuclideanDistance` и `Knn::InnerProductSimilarity`.

Режимы объединения

Через аргумент `Mode` доступны два способа объединения:

- **`rrf`** (по умолчанию) — Reciprocal Rank Fusion. Каждая ветвь вносит вклад `weight / (K + rank)`, где `rank` — позиция документа внутри этой ветви. Так как RRF использует ранги, а не исходные оценки, несопоставимые величины оценок ветвей не имеют значения.
- **`linear`** вычисляет взвешенную сумму оценок ветвей, в которой большее значение всегда означает лучшее совпадение. При `Normalize = true` (по умолчанию) нормализация использует минимум и максимум внутри пула кандидатов каждой ветви. Для релевантности и сходства используется `(score - min) / (max - min)`, для расстояний используется `(max - score) / (max - min)`. Ветвь с одинаковыми оценками всех кандидатов вносит `0`. При `Normalize = false` релевантность и сходство сохраняют исходные значения, а расстояния меняют знак перед умножением на вес.

В обоих режимах документ, отсутствующий в пуле кандидатов ветви, просто не получает вклада этой ветви.

Итоговый результат отсортирован по убыванию объединённой оценки. Взаимный порядок документов с одинаковой итоговой оценкой не гарантируется. Вес `0` обнуляет вклад ветви в оценку, но её кандидаты могут присутствовать в результате.

Пользовательская лямбда объединения

Вместо встроенного `Mode` (`rrf` / `linear`) можно передать собственную лямбду объединения, которая вычисляет итоговую оценку из значений ветвей. Доступны два взаимоисключающих вида лямбд:

- `RankLambda` — лямбда получает ранги документа по ветвям в виде `Dict<Int64, Int64>` (индекс ветви → 1-based ранг внутри этой ветви). Если документ отсутствует в ветви, соответствующей записи нет, поэтому `$ranks[i]` равно `NULL`.
- `ScoreLambda` — лямбда получает исходные оценки документа по ветвям в виде `Dict<Int64, Double>` (индекс ветви → значение оценки ветви: релевантность полнотекстового поиска или векторное расстояние/сходство). Если документ отсутствует в ветви, соответствующей записи нет, поэтому `$scores[i]` равно `NULL`.

Лямбда принимает один аргумент (словарь) и должна возвращать числовое значение (рекомендуется `Double`); чем больше значение, тем выше позиция в ранжировании. Лямбда заменяет встроенное объединение, поэтому её нельзя комбинировать с `Mode`, `Weights`, `K` или `Normalize` — перенесите все веса и константы в тело лямбды.

```
SELECT id, title
FROM documents
ORDER BY HybridRank(
    FullTextScore(text, $queryText),
    Knn::CosineDistance(embedding, $queryVector),
    ($ranks) -> {
        RETURN 1.0 / (60 + COALESCE($ranks[0], 100000))
             + 1.0 / (60 + COALESCE($ranks[1], 100000));
    } AS RankLambda)
LIMIT 10;
```

Именованные аргументы

Все необязательные параметры передаются как именованные аргументы. За исключением лямбд объединения, их значения должны быть литералами. Аргументы `Weights`, `Limits` и `Indexes` — это кортежи, позиционно соответствующие оценивающим выражениям: в них должно быть ровно по одному элементу на ветвь. Веса и константа RRF должны быть конечными; для RRF также требуется ненулевое значение `K + rank`.

Аргумент	Тип	Описание
`Mode`	String	Способ объединения: `"rrf"` (по умолчанию) или `"linear"`. Регистр не учитывается (`"RRF"` / `"Linear"` также допустимы).
`Weights`	Кортеж чисел	Вес каждой ветви, по одному на оценивающий аргумент. По умолчанию `1.0` для каждой ветви. Вес `0` обнуляет вклад в оценку, сохраняя кандидатов ветви.
`K`	Double	Константа RRF (только режим `rrf`). По умолчанию `60.0`. Большие значения сглаживают влияние верхних рангов.
`Normalize`	Bool	Только режим `linear`. При `true` (по умолчанию) оценки ветвей нормализуются методом min-max, а расстояния инвертируются. При `false` используются исходные оценки релевантности и сходства, а исходные расстояния меняют знак.
`Indexes`	Кортеж строк	Явное имя индекса для каждой ветви вместо автоматического выбора. Требуется при неоднозначности автоматического выбора.
`Limits`	Кортеж положительных целых	Явный размер пула кандидатов для каждой ветви. По умолчанию каждая ветвь извлекает до `LIMIT * HybridSearchFactor` кандидатов (фактор по умолчанию `10`).
`RankLambda`	Лямбда	Пользовательское объединение по рангам ветвей. Лямбда принимает `Dict<Int64, Int64>` (индекс ветви → 1-based ранг; для отсутствующих ветвей записи нет) и возвращает числовую оценку. Заменяет встроенный `Mode`; несовместим с `Mode`, `Weights`, `K` и `Normalize`.
`ScoreLambda`	Лямбда	Пользовательское объединение по исходным оценкам ветвей. Лямбда принимает `Dict<Int64, Double>` (индекс ветви → исходная оценка; для отсутствующих ветвей записи нет) и возвращает числовую оценку. Заменяет встроенный `Mode`; несовместим с `Mode`, `Weights`, `K` и `Normalize`.

Выбор индексов и фильтрация

Автоматический выбор рассматривает готовые индексы, последняя индексируемая колонка которых совпадает с колонкой оценивающего выражения. Префиксный индекс подходит, только если `WHERE` фиксирует значение каждой колонки префикса предикатами равенства, соединёнными через `AND`.

- Для полнотекстовой ветви должен остаться ровно один подходящий индекс релевантности. Поддерживаются исходный и компактный форматы хранения.
- Для векторной ветви метрика также должна соответствовать функции `Knn`. Из совместимых индексов выбирается индекс с самым длинным полностью заданным префиксом. Индексы с незаданным или частично заданным префиксом пропускаются. Если у нескольких индексов одинаковая максимальная длина префикса, выбор неоднозначен.

Для устранения неоднозначности используйте `AS Indexes`. При явном указании имени индекса по-прежнему проверяются колонка, метрика, готовность и префикс.

Например, для индексов по `(category, text)` и `(category, embedding)` выберите одну категорию перед ранжированием:

```
DECLARE $category AS Utf8;
DECLARE $queryText AS Utf8;
DECLARE $queryVector AS String;
```

```
SELECT id, title
FROM documents
WHERE category = $category
ORDER BY HybridRank(
    FullTextScore(text, $queryText),
    Knn::CosineDistance(embedding, $queryVector),
    ("ft_by_category", "vec_by_category") AS Indexes,
    (100, 100) AS Limits)
LIMIT 10;
```

Значениями префикса могут быть литералы, параметры или скалярные поля неопциональных параметров-структур. Колонки таблицы, допускающие `NULL`, могут входить в префикс при сравнении с неопциональными значениями. Требуется задать каждую колонку префикса, в том числе входящую в первичный ключ. В текущей реализации начальная часть префикса, равенство внутри `OR` и поле опционального параметра-структуры не удовлетворяют этому требованию.

Предикаты префикса ограничивают соответствующую индексную ветвь. Остальные предикаты `WHERE` применяются после объединения кандидатов и чтения строк основной таблицы, перед итоговым `LIMIT`. Они могут сократить результат ниже запрошенного размера; при необходимости увеличьте пулы кандидатов. Саму колонку, оцениваемую `FullTextScore`, нельзя фиксировать предикатом равенства в `WHERE`, поскольку все оставшиеся документы имели бы одинаковое текстовое значение для этой ветви.

Пулы кандидатов и пагинация

Задайте `PRAGMA ydb.HybridSearchFactor = "20";`, чтобы извлекать из каждой ветви до двадцати итоговых лимитов кандидатов, или используйте `AS Limits` для явного задания каждого пула. Эти настройки определяют количество объединяемых кандидатов; KMeansTreeSearchTopSize отдельно управляет обходом векторного индекса.

Изменение итогового `LIMIT` меняет и пулы кандидатов по умолчанию, поэтому может изменить ранжирование. Для пагинации используйте фиксированные явные пулы, достаточные для запрошенной страницы. `OFFSET` применяется к объединённому ранжированию. Векторный поиск приближённый, а порядок при равных оценках не определён, поэтому стабильный полный порядок между отдельными запросами не гарантируется.

Примеры

Взвешенный RRF — смещение ранжирования в сторону векторной ветви:

```
$queryText = "машинное обучение";
$queryVector = Knn::ToBinaryStringFloat([0.1, 0.2, 0.3, 0.4]);

SELECT id, title
FROM documents
ORDER BY HybridRank(
    FullTextScore(text, $queryText),
    Knn::CosineDistance(embedding, $queryVector),
    (1, 2) AS Weights)
LIMIT 10;
```

RRF по трём ветвям — комбинация текстовой релевантности с двумя векторными ветвями по разным колонкам эмбеддингов:

```
$queryText = "машинное обучение";
$titleVector = Knn::ToBinaryStringFloat([0.1, 0.2, 0.3, 0.4]);
$bodyVector = Knn::ToBinaryStringFloat([0.7, 0.8, 0.9, 1.0]);

SELECT id, title
FROM documents
ORDER BY HybridRank(
    FullTextScore(text, $queryText),
    Knn::CosineDistance(title_embedding, $titleVector),
    Knn::CosineDistance(body_embedding, $bodyVector),
    (1, 2, 3) AS Weights)
LIMIT 10;
```

Линейное объединение нормализованных оценок:

```
$queryText = "машинное обучение";
$queryVector = Knn::ToBinaryStringFloat([0.1, 0.2, 0.3, 0.4]);

SELECT id, title
FROM documents
ORDER BY HybridRank(
    FullTextScore(text, $queryText),
    Knn::CosineDistance(embedding, $queryVector),
    "linear" AS Mode)
LIMIT 10;
```

Явные имена индексов и размеры пулов кандидатов. Так как размеры пулов по ветвям заданы через `Limits`, собственный `LIMIT` запроса больше не обязан быть литералом и может быть параметром:

```
DECLARE $limit AS Uint64;

$queryText = "машинное обучение";
$queryVector = Knn::ToBinaryStringFloat([0.1, 0.2, 0.3, 0.4]);

SELECT id, title
FROM documents
ORDER BY HybridRank(
    FullTextScore(text, $queryText),
    Knn::CosineDistance(embedding, $queryVector),
```

```
    ("ft_idx", "vec_idx") AS Indexes,
    (100, 200) AS Limits)
LIMIT $limit;
```

Пользовательский `RankLambda` — RRF вручную, с весом векторной ветви в 100 раз больше текстовой:

```
$queryText = "машинное обучение";
$queryVector = Knn::ToBinaryStringFloat([0.1, 0.2, 0.3, 0.4]);

SELECT id, title
FROM documents
ORDER BY HybridRank(
    FullTextScore(text, $queryText),
    Knn::CosineDistance(embedding, $queryVector),
    ($ranks) -> {
        RETURN   1.0 / (60 + COALESCE($ranks[0], 100000))
             + 100.0 / (60 + COALESCE($ranks[1], 100000));
    } AS RankLambda)
LIMIT 10;
```

Пользовательский `ScoreLambda` — комбинация исходной релевантности полнотекстового поиска и векторного расстояния со знаком минус (ближе = больше оценка):

```
$queryText = "машинное обучение";
$queryVector = Knn::ToBinaryStringFloat([0.1, 0.2, 0.3, 0.4]);

SELECT id, title
FROM documents
ORDER BY HybridRank(
    FullTextScore(text, $queryText),
    Knn::CosineDistance(embedding, $queryVector),
    ($scores) -> {
        RETURN COALESCE($scores[0], 0.0) - COALESCE($scores[1], 1000000.0);
    } AS ScoreLambda)
LIMIT 10;
```

Ограничения

- `HybridRank(...)` должен быть единственным ключом `ORDER BY`; его нельзя отрицать, вкладывать в большее выражение или комбинировать с другими ключами сортировки.
- Текущая реализация поддерживает только таблицы с одноколоночным первичным ключом.
- Требуется не менее двух оценивающих аргументов — одна ветвь не является гибридным запросом.
- `LIMIT` должен быть литералом (он задаёт размеры пулов кандидатов по ветвям). Чтобы использовать параметризованный `LIMIT`, передайте явный `AS Limits`.
- Пользовательская лямбда объединения (`RankLambda` или `ScoreLambda`) заменяет встроенное объединение и не может комбинироваться с `Mode`, `Weights`, `K` или `Normalize`. Можно указать не более одного из `RankLambda` или `ScoreLambda`.
- Префиксные индексы требуют равенств для всех колонок префикса; текущая реализация гибридного запроса не поддерживает частичные префиксы.

WITH

Задается после источника данных во `FROM` и используется для указания дополнительных подсказок использования таблиц. Подсказки нельзя задать для подзапросов и именованных выражений.

Поддерживаются следующие значения:

- `INFER_SCHEMA` — задает флаг вывода схемы таблицы. Поведение аналогично заданию прагмы yt.InferSchema, только для конкретного источника данных. Можно задать число строк для выведения (число от 1 до 1000).
- `FORCE_INFER_SCHEMA` — задает флаг вывода схемы таблицы. Поведение аналогично заданию прагмы yt.ForceInferSchema, только для конкретного источника данных. Можно задать число строк для выведения (число от 1 до 1000).
- `DIRECT_READ` — подавляет работу некоторых оптимизаторов и заставляет использовать содержимое таблицы как есть. Поведение аналогично заданию отладочной прагмы DirectRead, только для конкретного источника данных.
- `INLINE` — указание на то, что содержимое таблицы небольшое и нужно использовать его представление в памяти для обработки запроса. Реальный объем таблицы при этом не контролируется, и если он большой, то запрос может упасть по превышению памяти.
- `UNORDERED` — подавляет использование исходной сортировки таблицы.
- `XLOCK` — указание на то, что нужно брать эксклюзивный лок на таблицу. Полезен, когда чтение таблицы происходит на стадии обработки метапрограммы запроса, а затем ее содержимое обновляется в основном запросе. Позволяет избежать потери данных, если между исполнением фазы метапрограммы и основной частью запроса внешний процесс успел изменить таблицу.
- `SCHEMA` type — указание на то, что следует использовать указанную схему таблицы целиком, игнорируя схему в метаданных.
- `COLUMNS` type — указание на то, что следует использовать указанные типы для колонок, чьи имена совпадают с именами колонок таблицы в метаданных, а также какие колонки дополнительно присутствуют в таблице.
- `IGNORETYPEV3` , `IGNORE_TYPE_V3` — задает флаг игнорирования type_v3 типов в таблице. Поведение аналогично заданию прагмы yt.IgnoreTypeV3, только для конкретного источника данных.

При работе с внешними файловыми источниками данных можно дополнительно указывать ряд параметров:

- `FORMAT` - формат хранимых данных в файловых хранилищах в федеративных запросах. Допустимые значения: `csv_with_names` , `tsv_with_names` , `json_list` , `json_each_row` , `json_as_string` , `parquet` , `raw` .
- `COMPRESSION` - формат сжатия файлов в файловых хранилищах в федеративных запросах. Допустимые значения: gzip, zstd, lz4, brotli, bzip2, xz.
- `PARTITIONED_BY` - список колонок партиционирования данных в файловых хранилищах в федеративных запросах. Содержит список колонок в порядке их размещения в файловом хранилище.
- `projection.enabled` - флаг включения расширенного партиционирования данных. Допустимые значения: `true` , `false` .
- `projection.<field_name>.type` - тип поля расширенного партиционирования данных. Допустимые значения: `integer` , `enum` , `date` .
- `projection.<field_name>.<options>` - расширенные свойства поля расширенного партиционирования данных.

При чтении из топика в потоковых запросах можно указывать параметры watermarks:

- `WATERMARK` — выражение для вычисления watermark. Сейчас поддерживается только время записи в топик с константной задержкой. Формат: `__ydb_write_time - Interval("<delay>")` , где `<delay>` задаётся в формате ISO 8601.
- `WATERMARK_GRANULARITY` — периодичность генерации watermarks. Чем она меньше, тем больше потребление CPU запросом и тем меньше задержка ответа. Имеет смысл только для потоковых запросов. Задаётся в формате ISO 8601. Значение по умолчанию — 1 секунда.
- `WATERMARK_IDLE_TIMEOUT` — период, после которого простаивающая партиция будет исключена из вычисления объединённого watermark. Имеет смысл только для потоковых запросов. Задаётся в формате ISO 8601. Значение по умолчанию — 5 секунд.

При задании подсказок `SCHEMA` и `COLUMNS` в качестве значения типа type должен быть задан тип структуры.

Примеры

```
SELECT key FROM my_table WITH INFER_SCHEMA;
SELECT key FROM my_table WITH FORCE_INFER_SCHEMA="42";
```

```
$s = (SELECT COUNT(*) FROM my_table WITH XLOCK);

INSERT INTO my_table WITH TRUNCATE
SELECT EvaluateExpr($s) AS a;
```

```
SELECT key, value FROM my_table WITH SCHEMA Struct<key:String, value:Int32>;
```

```
SELECT key, value FROM my_table WITH COLUMNS Struct<value:Int32?>;
```

```
SELECT key, value FROM EACH($my_tables) WITH SCHEMA Struct<key:String, value:List<Int32>>;
```

```
SELECT
    *
FROM
    my_topic
WITH (
    FORMAT = json_each_row,
    SCHEMA = (
```

```
        ts String
    ),
    WATERMARK = __ydb_write_time - Interval("PT5S"),
    WATERMARK_GRANULARITY = "PT1S",
    WATERMARK_IDLE_TIMEOUT = "PT5S"
);
```

WITHOUT

Исключение столбцов из результата запроса `SELECT *`.

Примеры

```
SELECT * WITHOUT foo, bar FROM my_table;
```

```
PRAGMA simplecolumns;
SELECT * WITHOUT t.foo FROM my_table AS t
CROSS JOIN (SELECT 1 AS foo) AS v;
```

WHERE

Фильтрация строк в результате выполнения `SELECT` по условию в колоночной или строковой таблице.

Пример

```
SELECT key FROM my_table
WHERE value > 0;
```

Важно

Коррелированные подзапросы в `WHERE` не поддерживаются: подзапрос не может обращаться к колонкам или псевдонимам таблиц из внешнего запроса. Для отбора строк, у которых есть совпадающие строки в другой таблице, используйте `LEFT SEMI JOIN`. Для отбора строк без совпадений используйте `LEFT ONLY JOIN`.

ORDER BY

Сортировка результата `SELECT` по разделенному запятыми перечню критериев сортировки. В качестве критерия может выступать значение столбца, или выражение над столбцами. Не поддерживается указание порядкового номера колонки выборки (`ORDER BY N`, где `N` - номер).

Направление сортировки может быть указано после каждого критерия:

- `ASC` — по возрастанию. Применяется по умолчанию.
- `DESC` — по убыванию.

Несколько критериев сортировки будут применены слева направо.

Пример

```
SELECT key, string_column
FROM my_table
ORDER BY key DESC, LENGTH(string_column) ASC;
```

Ключевое слово `ORDER BY` также может использоваться в механизме оконных функций.

ASSUME ORDER BY

Проверка сортированности результата `SELECT` по значению в указанном столбце или нескольких столбцах. Результат такого `SELECT`-а будет считаться сортированным, но без выполнения фактической сортировки. Проверка сортированности осуществляется на этапе исполнения запроса.

Как и для `ORDER BY`, поддерживается задание порядка сортировки с помощью ключевых слов `ASC` (по возрастанию) и `DESC` (по убыванию). Выражения в `ASSUME ORDER BY` не поддерживается.

Примеры

```
SELECT key || "suffix" as key, -CAST(subkey as Int32) as subkey
FROM my_table
ASSUME ORDER BY key, subkey DESC;
```

LIMIT и OFFSET

`LIMIT` ограничивает вывод указанным количеством строк. Если значение лимита равно `NULL`, или `LIMIT` не указан, то вывод не ограничен.

`OFFSET` указывает отступ от начала (в строках). Если значение отступа равно `NULL`, или `OFFSET` не указан, то используется значение ноль.

Примеры

```
SELECT key FROM my_table
LIMIT 7;
```

```
SELECT key FROM my_table
LIMIT 7 OFFSET 3;
```

```
SELECT key FROM my_table
LIMIT 3, 7; -- эквивалентно предыдущему примеру
```

```
SELECT key FROM my_table
LIMIT NULL OFFSET NULL; -- эквивалентно SELECT key FROM my_table
```

SHOW CREATE

`SHOW CREATE` возвращает запрос, возможно состоящий из нескольких DDL-выражений, необходимых для воссоздания структуры выбранного объекта: таблицы или представления.

Синтаксис

```
SHOW CREATE [TABLE|VIEW] <name>;
```

Параметры

- `TABLE|VIEW` - тип объекта: `TABLE` для таблицы или `VIEW` для представления.
- `<name>` - имя объекта, также может быть указан абсолютный путь до объекта.

Результат выполнения

Команда возвращает **ровно одну строку** с тремя колонками:

Path	PathType	CreateQuery
Путь	Table/View	DDL-выражения для создания объекта

- **Path** — путь к объекту (например, `MyTable` или `MyView`).
- **PathType** — тип объекта: `Table` или `View`.
- **CreateQuery** — полный набор DDL-выражений, необходимых для создания объекта:
  - Для таблиц: основной оператор CREATE TABLE (с путем относительно базы), а также дополнительные команды, необходимые для описания текущего состояния и настроек:
    - ALTER TABLE ... ALTER INDEX— для задания настроек партицирования вторичных индексов.
    - ALTER TABLE ... ADD CHANGEFEED— для добавления потока изменений.
    - `ALTER SEQUENCE` — для восстановления состояния `Sequence` у колонок типа Serial.
  - Для представлений: определение посредством команды CREATE VIEW, а также, если необходимо, выражения, которые были зафиксированы представлением из контекста создания, например, PRAGMA TablePathPrefix.

Примеры

Строковые таблицы

```
SHOW CREATE TABLE my_table;
```

Path	PathType	CreateQuery
`my_table`	`Table`	`CREATE TABLE...` - см. ниже

```
CREATE TABLE `my_table` (
    `Key1` Uint32 NOT NULL,
    `Key2` Utf8 NOT NULL,
    `Key3` Serial4 NOT NULL,
    `Value1` Utf8 FAMILY `my_family`,
    `Value2` Bool,
    `Value3` String,
    INDEX `my_index` GLOBAL SYNC ON (`Key2`, `Value1`, `Value2`),
    FAMILY `my_family` (COMPRESSION = 'lz4'),
    PRIMARY KEY (`Key1`, `Key2`, `Key3`)
)
WITH (
    AUTO_PARTITIONING_BY_SIZE = ENABLED,
    AUTO_PARTITIONING_PARTITION_SIZE_MB = 1000
);

ALTER TABLE `my_table`
    ADD CHANGEFEED `feed_3` WITH (MODE = 'KEYS_ONLY', FORMAT = 'JSON', RETENTION_PERIOD = INTERVAL('PT30M'),
TOPIC_MIN_ACTIVE_PARTITIONS = 3)
;

ALTER SEQUENCE `/Root/my_table/_serial_column_Key3` START WITH 101 INCREMENT BY 404 RESTART;

ALTER TABLE `my_table`
    ALTER INDEX `my_index` SET (AUTO_PARTITIONING_BY_LOAD = ENABLED, AUTO_PARTITIONING_MIN_PARTITIONS_COUNT =
1000, AUTO_PARTITIONING_MAX_PARTITIONS_COUNT = 5000)
;
```

Колоночные таблицы

```
SHOW CREATE TABLE my_table;
```

Path	PathType	CreateQuery
`my_table`	`Table`	`CREATE TABLE...` - см. ниже

```
CREATE TABLE `my_table` (
    `Key1` Uint64 NOT NULL,
    `Key2` Utf8 NOT NULL,
```

```
    `Key3` Int32 NOT NULL,
    `Value1` Utf8,
    `Value2` Int16,
    `Value3` String,
    PRIMARY KEY (`Key1`, `Key2`, `Key3`)
)
PARTITION BY HASH (`Key1`, `Key2`)
WITH (
    STORE = COLUMN,
    AUTO_PARTITIONING_MIN_PARTITIONS_COUNT = 100,
    TTL =
        INTERVAL('PT10S') TO EXTERNAL DATA SOURCE `/Root/tier1`,
        INTERVAL('PT1H') DELETE
    ON Key1 AS SECONDS
);
```

Представления

```
SHOW CREATE VIEW my_view;
```

Path	PathType	CreateQuery
`my_view`	`View`	`PRAGMA TablePathPrefix...` - см. ниже

```
PRAGMA TablePathPrefix = '/Root/DirA/DirB/DirC';

CREATE VIEW `my_view` WITH (security_invoker = TRUE) AS
SELECT
    *
FROM
    test_table
;
```

TRUNCATE TABLE

`TRUNCATE TABLE` удаляет все пользовательские данные из указанной таблицы и ее индексов.

Синтаксис

```
TRUNCATE TABLE <table_name>;
```

Ограничения

- Поддерживается только для строковых таблиц.
- Во время выполнения операции таблица блокируется на чтение и запись.
- Операцию нельзя прервать или отменить после начала выполнения.
- Операцию нельзя выполнить, если у таблицы есть:
  - асинхронный вторичный индекс,
  - поток изменений,
  - асинхронная репликация.

Примеры

Удаляет все данные из таблицы с полным путем `/Root/test/my_table`.

```
TRUNCATE TABLE `/Root/test/my_table`;
```

Удаляет все данные из таблицы `my_table` в текущей базе данных.

```
TRUNCATE TABLE `my_table`;
```

См. также

- CREATE TABLE
- ALTER TABLE
- DROP TABLE

UPDATE

Важно

В настоящее время одновременное использование колоночных и строковых таблиц поддерживается в транзакциях, в которых данные только читаются, но не изменяются. Поддержка транзакций с возможностью модификации данных при одновременном использовании строковых и колоночных таблиц находится в разработке.

Если попытаться выполнить операцию записи в транзакции, в которой задействованы и колоночные, и строковые таблицы, транзакция завершится с ошибкой:

```
Write transactions that use both row-oriented and column-oriented tables are disabled at current t
ime
```

Изменяет данные в таблице. После ключевого слова `SET` через запятую указываются столбцы, значение которых необходимо заменить, и сами новые значения. Список обновляемых строк задается с помощью условия `WHERE`. Если условие `WHERE` отсутствует, изменения будут применены ко всем строкам таблицы.

`UPDATE` не может менять значение колонок, входящих в состав первичного ключа.

Пример

```
UPDATE my_table
SET Value1 = YQL::ToString(Value2 + 1), Value2 = Value2 - 1
WHERE Key1 > 1;
```

UPDATE ON

Используется для обновления данных на основе результатов подзапроса. Набор колонок, возвращаемых подзапросом, должен быть подмножеством колонок обновляемой таблицы. В составе возвращаемых подзапросом колонок обязательно должны присутствовать все колонки первичного ключа таблицы. Типы данных возвращаемых подзапросом колонок должны совпадать с типами данных соответствующих колонок таблицы.

Для поиска обновляемых записей используется значение первичного ключа. В каждой найденной записи при выполнении оператора изменяются значения неключевых колонок, указанных в подзапросе. Значения колонок таблицы, которые отсутствуют в возвращаемых колонках подзапроса, остаются неизменными.

Пример

```
$to_update = (
    SELECT Key, SubKey, "Updated" AS Value FROM my_table
    WHERE Key = 1
);

UPDATE my_table ON
SELECT * FROM $to_update;
```

UPDATE ... RETURNING

Используется для обновления строк и одновременного возврата их новых значений. Это позволяет получить информацию об обновленных записях за один запрос, избавляя от необходимости выполнять дополнительный SELECT.

Примеры

- Возврат всех значений обновленных строк

```
UPDATE orders
SET status = 'shipped'
WHERE order_date < '2023-01-01'
RETURNING *;
```

- Возврат конкретных столбцов

```
UPDATE products
SET price = price * 0.9
WHERE category = 'Electronics'
RETURNING product_id, name, price AS new_price;
```

См. также

- BATCH UPDATE

UPSERT INTO

Важно

В настоящее время одновременное использование колоночных и строковых таблиц поддерживается в транзакциях, в которых данные только читаются, но не изменяются. Поддержка транзакций с возможностью модификации данных при одновременном использовании строковых и колоночных таблиц находится в разработке.

Если попытаться выполнить операцию записи в транзакции, в которой задействованы и колоночные, и строковые таблицы, транзакция завершится с ошибкой:

```
Write transactions that use both row-oriented and column-oriented tables are disabled at current t
ime
```

UPSERT (расшифровывается как UPDATE or INSERT) обновляет или добавляет множество строк в таблице на основании сравнения по первичному ключу. Отсутствующие строки добавляются. В присутствующих строках обновляются значения заданных столбцов, значения остальных столбцов остаются неизменными.

`UPSERT` и `REPLACE` являются операциями модификации данных, которые не требует их предварительного чтения, за счет чего работают быстрее и дешевле других операций.

Сопоставление столбцов при использовании `UPSERT INTO ... SELECT` производится по именам. Используйте `AS` для получения колонки с нужным именем в `SELECT`.

Примеры

```
UPSERT INTO my_table
SELECT pk_column, data_column1, col24 as data_column3 FROM other_table
```

```
UPSERT INTO my_table ( pk_column1, pk_column2, data_column2, data_column5 )
VALUES ( 1, 10, 'Some text', Date('2021-10-07')),
       ( 2, 10, 'Some text', Date('2021-10-08'))
```

UPSERT INTO ... RETURNING

Используется для вставки или обновления строк и одновременного возврата значений из них. Это позволяет получить информацию об измененной записи за один запрос, избавляя от необходимости выполнять дополнительный SELECT.

Примеры

- Возврат всех значений измененной строки

```
UPSERT INTO orders (order_id, status, amount)
VALUES (1001, 'shipped', 500)
RETURNING *;
```

- Возврат конкретных столбцов

```
UPSERT INTO users (user_id, name, email)
VALUES (42, 'John Doe', 'john@example.com')
RETURNING user_id, email;
```

UPSERT OBJECT (TYPE SECRET)

Внимание

Данная команда устарела и будет удалена в будущих версиях YDB. Рекомендуемый синтаксис работы с секретами описан в разделе Секреты.

Для изменения содержимого секрета используется следующий SQL-запрос:

```
UPSERT OBJECT `secret_name` (TYPE SECRET) WITH value = `secret_value`;
```

Где:

- `secret_name` - имя секрета.
- `secret_value` - содержимое секрета.

Пример

Следующий запрос устанавливает новое значение секрета с именем `MySecretName` в значение `MySecretData`.

```
UPSERT OBJECT `MySecretName` (TYPE SECRET) WITH value = `MySecretData`;
```

Базовый синтаксис VALUES в YQL

VALUES как оператор верхнего уровня

Позволяет сформировать таблицу из указанных значений. Например, данное выражение формирует таблицу из k колонок и n строк:

```
VALUES (expr_11, expr_12, ..., expr_1k),
       (expr_21, expr_22, ..., expr_2k),
       ....
       (expr_n1, expr_n2, ..., expr_nk);
```

Это выражение полностью эквивалентно следующему:

```
SELECT expr_11, expr_12, ..., expr_1k UNION ALL
SELECT expr_21, expr_22, ..., expr_2k UNION ALL
....
SELECT expr_n1, expr_n2, ..., expr_nk;
```

Пример

```
VALUES (1,2), (3,4);
```

VALUES после FROM

`VALUES` может использоваться и в подзапросе после FROM. В частности, эти два запроса эквивалентны:

```
VALUES (1,2), (3,4);
SELECT * FROM (VALUES (1,2), (3,4));
```

Во всех примерах выше имена колонок назначаются YQL и имеют вид `column0 ... columnN`. Для того, чтобы назначить произвольные имена колонок, можно воспользоваться следующей конструкцией:

```
SELECT * FROM (VALUES (1,2), (3,4)) as t(x,y);
```

В данном случае колонки получат имена `x`, `y`.

Встроенные функции YQL

- Базовые
- Агрегатные
- Оконные
- Для работы со списками
- Для работы со словарями
- Для работы со структурами
- Для работы с типами
- Для работы с генерацией кода
- Для работы с JSON
- Библиотеки C++

Базовые встроенные функции

Ниже описаны функции общего назначения, а для специализированных функций есть отдельные статьи: агрегатные, оконные, а также для работы со списками, словарями, структурами, типами данных и генерацией кода.

COALESCE

Перебирает аргументы слева направо и возвращает первый найденный непустой аргумент. Чтобы результат получился гарантированно непустым (не optional типа), самый правый аргумент должен быть такого типа (зачастую используют литерал). При одном аргументе возвращает его без изменений.

Сигнатура

```
COALESCE(T?, ..., T)->T
COALESCE(T?, ..., T?)->T?
```

Позволяет передавать потенциально пустые значения в функции, которые не умеют обрабатывать их самостоятельно.

Доступен краткий формат записи в виде оператора `??`. Можно использовать алиас `NVL`.

Примеры

```
SELECT COALESCE(
  maybe_empty_column,
  "it's empty!"
) FROM my_table;
```

```
SELECT
  maybe_empty_column ?? "it's empty!"
FROM my_table;
```

```
SELECT NVL(
  maybe_empty_column,
  "it's empty!"
) FROM my_table;
```

Все три примера выше эквивалентны.

LENGTH

Возвращает длину строки в байтах. Также эта функция доступна под именем `LEN`.

Сигнатура

```
LENGTH(T)->Uint32
LENGTH(T?)->Uint32?
```

Примеры

```
SELECT LENGTH("foo");
```

```
SELECT LEN("bar");
```

Примечание

Для вычисления длины строки в unicode символах можно воспользоваться функцией Unicode::GetLength.

Для получения числа элементов в списке нужно использовать функцию ListLength.

SUBSTRING

Возвращает подстроку.

Сигнатура

```
Substring(String[, Uint32? [, Uint32?]])->String
Substring(String?[, Uint32? [, Uint32?]])->String?
```

Обязательные аргументы:

- Исходная строка;
- Позиция — отступ от начала строки в байтах (целое число) или `NULL`, означающий «от начала».

Опциональные аргументы:

- Длина подстроки — количество байт, начиная с указанной позиции (целое число, или `NULL` по умолчанию, означающий «до конца исходной строки»).

Индексация с нуля. Если указанные позиция и длина выходят за пределы строки, возвращает пустую строку.
Если входная строка является опциональной, то таким же является и результат.

Примеры

```
SELECT SUBSTRING("abcdefg", 3, 1); -- d
```

```
SELECT SUBSTRING("abcdefg", 3); -- defg
```

```
SELECT SUBSTRING("abcdefg", NULL, 3); -- abc
```

FIND

Поиск позиции подстроки в строке.

Сигнатура

```
Find(String, String[, Uint32?])->Uint32?
Find(String?, String[, Uint32?])->Uint32?
Find(Utf8, Utf8[, Uint32?])->Uint32?
Find(Utf8?, Utf8[, Uint32?])->Uint32?
```

Обязательные аргументы:

- Исходная строка;
- Искомая подстрока.

Опциональные аргументы:

- Позиция — в байтах, с которой начинать поиск (целое число, или `NULL` по умолчанию, означающий «от начала исходной строки»).

Возвращает первую найденную позицию подстроки или `NULL`, означающий что искомая подстрока с указанной позиции не найдена.

Примеры

```
SELECT FIND("abcdefg_abcdefg", "abc"); -- 0
```

```
SELECT FIND("abcdefg_abcdefg", "abc", 1); -- 8
```

```
SELECT FIND("abcdefg_abcdefg", "abc", 9); -- null
```

RFIND

Обратный поиск позиции подстроки в строке, от конца к началу.

Сигнатура

```
RFind(String, String[, Uint32?])->Uint32?
RFind(String?, String[, Uint32?])->Uint32?
RFind(Utf8, Utf8[, Uint32?])->Uint32?
RFind(Utf8?, Utf8[, Uint32?])->Uint32?
```

Обязательные аргументы:

- Исходная строка;
- Искомая подстрока.

Опциональные аргументы:

- Позиция — в байтах, с которой начинать поиск (целое число, или `NULL` по умолчанию, означающий «от конца исходной строки»).

Возвращает первую найденную позицию подстроки или `NULL`, означающий, что искомая подстрока с указанной позиции не найдена.

Примеры

```
SELECT RFIND("abcdefg_abcdefg", "bcd"); -- 9
```

```
SELECT RFIND("abcdefg_abcdefg", "bcd", 8); -- 1
```

```
SELECT RFIND("abcdefg_abcdefg", "bcd", 0); -- null
```

StartsWith, EndsWith

Проверка наличия префикса или суффикса в строке.

Сигнатуры

```
StartsWith(T str, U prefix)->Bool[?]

EndsWith(T str, U suffix)->Bool[?]
```

Обязательные аргументы:

- Исходная строка;
- Искомая подстрока.

Аргументы должны иметь тип `String` / `Utf8` (или опциональный `String` / `Utf8`) либо строковый PostgreSQL тип (`PgText` / `PgBytea` / `PgVarchar`).
Результатом функции является опциональный Bool, за исключением случая, когда оба аргумента неопциональные – в этом случае возвращается Bool.

Примеры

```
SELECT StartsWith("abc_efg", "abc") AND EndsWith("abc_efg", "efg"); -- true
```

```
SELECT StartsWith("abc_efg", "efg") OR EndsWith("abc_efg", "abc"); -- false
```

```
SELECT StartsWith("abcd", NULL); -- null
```

```
SELECT EndsWith(NULL, Utf8("")); -- null
```

```
SELECT StartsWith("abc_efg"u, "abc"p) AND EndsWith("abc_efg", "efg"pv); -- true
```

IF

Проверяет условие `IF(condition_expression, then_expression, else_expression)`.

Является упрощенной альтернативой для CASE WHEN ... THEN ... ELSE ... END.

Сигнатура

```
IF(Bool, T, T)->T
IF(Bool, T)->T?
```

Аргумент `else_expression` можно не указывать. В этом случае, если условие ложно (`condition_expression` вернул `false`), будет возвращено пустое значение с типом, соответствующим `then_expression` и допускающим значение `NULL`. Таким образом, у результата получится optional тип данных.

Примеры

```
SELECT
  IF(foo > 0, bar, baz) AS bar_or_baz,
  IF(foo > 0, foo) AS only_positive_foo
FROM my_table;
```

NANVL

Заменяет значения `NaN` (not a number) в выражениях типа `Float`, `Double` или Optional.

Сигнатура

```
NANVL(Float, Float)->Float
NANVL(Double, Double)->Double
```

Аргументы:

1. Выражение, в котором нужно произвести замену.
2. Значение, на которое нужно заменить `NaN`.

Если один из агрументов `Double`, то в выдаче `Double`, иначе `Float`. Если один из аргументов `Optional`, то и в выдаче `Optional`.

Примеры

```
SELECT
  NANVL(double_column, 0.0)
FROM my_table;
```

Random...

Генерирует псевдослучайное число:

- `Random()` — число с плавающей точкой (Double) от 0 до 1;
- `RandomNumber()` — целое число из всего диапазона Uint64;

- `RandomUuid()` — Uuid version 4.

Сигнатуры

```
Random(T1[, T2, ...])->Double
RandomNumber(T1[, T2, ...])->Uint64
RandomUuid(T1[, T2, ...])->Uuid
```

При генерации случайных чисел аргументы не используются и нужны исключительно для управления моментом вызова. В каждый момент вызова возвращается новое случайное число. Поэтому:

- Повторный вызов Random в рамках **одного запроса** при идентичном наборе аргументов не гарантирует получения одинаковых наборов случайных чисел. Значения будут равны, если вызовы Random попадут в одну фазу исполнения.
- Вызовы Random с одним и тем же набором аргументов в **разных запросах** вернут разные наборы случайных чисел.

> **Важно**
>
> Если Random используется в именованных выражениях, то его однократное вычисление не гарантируется. В зависимости от оптимизаторов и среды исполнения он может посчитаться как один раз, так и многократно. Для гарантированного однократного подсчета необходимо в этом случае материализовать именованное выражение в таблицу.

Сценарии использования:

- `SELECT RANDOM(1);` — получить одно случайное значение на весь запрос и несколько раз его использовать (чтобы получить несколько, можно передать разные константы любого типа);
- `SELECT RANDOM(1) FROM table;` — одно и то же случайное число на каждую строку таблицы;
- `SELECT RANDOM(1), RANDOM(2) FROM table;` — по два случайных числа на каждую строку таблицы, все числа в каждой из колонок одинаковые;
- `SELECT RANDOM(some_column) FROM table;` — разные случайные числа на каждую строку таблицы;
- `SELECT RANDOM(some_column), RANDOM(some_column) FROM table;` — разные случайные числа на каждую строку таблицы, но в рамках одной строки — два одинаковых числа;
- `SELECT RANDOM(some_column), RANDOM(some_column + 1) FROM table;` или `SELECT RANDOM(some_column), RANDOM(other_column) FROM table;` — две колонки, и все с разными числами.

Примеры

```
SELECT
    Random(key) -- [0, 1)
FROM my_table;
```

```
SELECT
    RandomNumber(key) -- [0, Max<Uint64>)
FROM my_table;
```

```
SELECT
    RandomUuid(key) -- Uuid version 4
FROM my_table;
```

```
SELECT
    RANDOM(column) AS rand1,
    RANDOM(column) AS rand2, -- same as rand1
    RANDOM(column, 1) AS randAnd1, -- different from rand1/2
    RANDOM(column, 2) AS randAnd2 -- different from randAnd1
FROM my_table;
```

Udf

Строит `Callable` по заданному названию функции и опциональным `external user types`, `RunConfig` и `TypeConfig`.

- `Udf(Foo::Bar)` — Функция `Foo::Bar` без дополнительных параметров.
- `Udf(Foo::Bar)(1, 2, 'abc')` — Вызов udf `Foo::Bar`.
- `Udf(Foo::Bar, Int32, @@{"device":"AHCI"}@@ as TypeConfig")(1, 2, 'abc')` — Вызов udf `Foo::Bar` с дополнительным типом `Int32` и указанным `TypeConfig`.
- `Udf(Foo::Bar, "1e9+7" as RunConfig")(1, 'extended' As Precision)` — Вызов udf `Foo::Bar` с указанным `RunConfig` и именованными параметрами.

Сигнатуры

```
Udf(Callable[, T1, T2, ..., T_N][, V1 as TypeConfig][,V2 as RunConfig]])->Callable
```

Где `T1`, `T2`, и т. д. -- дополнительные (`external`) пользовательские типы.

Примеры

```
$IsoParser = Udf(DateTime2::ParseIso8601);
SELECT $IsoParser("2022-01-01");
```

```
SELECT Udf(Unicode::IsUtf)("2022-01-01")
```

```
$config = @@{
    "name":"MessageFoo",
    "meta": "..."
}@@;
SELECT Udf(Protobuf::TryParse, $config As TypeConfig)("")
```

CurrentUtc...

`CurrentUtcDate()`, `CurrentUtcDatetime()` и `CurrentUtcTimestamp()` - получение текущей даты и/или времени в UTC. Тип данных результата указан в конце названия функции.

Сигнатуры

```
CurrentUtcDate(...)->Date
CurrentUtcDatetime(...)->Datetime
CurrentUtcTimestamp(...)->Timestamp
```

Аргументы опциональны и работают по тому же принципу, что и у RANDOM.

Примеры

```
SELECT CurrentUtcDate();
```

```
SELECT CurrentUtcTimestamp(TableRow()) FROM my_table;
```

CurrentTz...

`CurrentTzDate()`, `CurrentTzDatetime()` и `CurrentTzTimestamp()` - получение текущей даты и/или времени в указанной в первом аргументе IANA временной зоне. Тип данных результата указан в конце названия функции.

Сигнатуры

```
CurrentTzDate(String, ...)->TzDate
CurrentTzDatetime(String, ...)->TzDatetime
CurrentTzTimestamp(String, ...)->TzTimestamp
```

Последующие аргументы опциональны и работают по тому же принципу, что и у RANDOM.

Примеры

```
SELECT CurrentTzDate("Europe/Moscow");
```

```
SELECT CurrentTzTimestamp("Europe/Moscow", TableRow()) FROM my_table;
```

AddTimezone

Добавление информации о временной зоне к дате/времени, заданных в UTC. При выводе в результате `SELECT` или после `CAST` в `String` будут применены правила временной зоны по вычислению смещения времени.

Сигнатура

```
AddTimezone(Date, String)->TzDate
AddTimezone(Date?, String)->TzDate?
AddTimezone(Datetime, String)->TzDatetime
AddTimezone(Datetime?, String)->TzDatetime?
AddTimezone(Timestamp, String)->TzTimestamp
AddTimezone(Timestamp?, String)->TzTimestamp?
```

Аргументы:

1. Дата - тип `Date` / `Datetime` / `Timestamp`;
2. IANA имя временной зоны.

Тип результата - `TzDate` / `TzDatetime` / `TzTimestamp`, в зависимости от типа данных входа.

Примеры

```
SELECT AddTimezone(Datetime("2018-02-01T12:00:00Z"), "Europe/Moscow");
```

RemoveTimezone

Удаление информации о временной зоне и перевод в дату/время, заданные в UTC.

Сигнатура

```
RemoveTimezone(TzDate)->Date
RemoveTimezone(TzDate?)->Date?
```

```
RemoveTimezone(TzDatetime)->Datetime
RemoveTimezone(TzDatetime?)->Datetime?
RemoveTimezone(TzTimestamp)->Timestamp
RemoveTimezone(TzTimestamp?)->Timestamp?
```

Аргументы:

1. Дата - тип `TzDate` / `TzDatetime` / `TzTimestamp`.

Тип результата - `Date` / `Datetime` / `Timestamp`, в зависимости от типа данных входа.

Примеры

```
SELECT RemoveTimezone(TzDatetime("2018-02-01T12:00:00,Europe/Moscow"));
```

Version

`Version()` возвращает строку, описывающую текущую версию узла, обрабатывающего запрос. В некоторых случаях, например, во время постепенного обновлений кластера, она может возвращать разные строки в зависимости от того, какой узел обрабатывает запрос. Функция не принимает никаких аргументов.

Примеры

```
SELECT Version();
```

MAX_OF, MIN_OF, GREATEST и LEAST

Возвращает минимальный или максимальный среди N аргументов. Эти функции позволяют не использовать стандартную для SQL конструкцию `CASE WHEN a < b THEN a ELSE b END`, которая была бы особенно громоздкой для N больше двух.

Сигнатуры

```
MIN_OF(T[,T,...])->T
MAX_OF(T[,T,...])->T
```

Типы аргументов должны быть приводимы друг к другу и могут допускать значение `NULL`.

`GREATEST` является синонимом к `MAX_OF`, а `LEAST` — к `MIN_OF`.

Примеры

```
SELECT MIN_OF(1, 2, 3);
```

AsTuple, AsStruct, AsList, AsDict, AsSet, AsListStrict, AsDictStrict и AsSetStrict

Создает контейнеры соответствующих типов. Также доступна операторная запись литералов контейнеров.

Особенности:

- Элементы контейнеров передаются через аргументы, таким образом число элементов результирующего контейнера равно числу переданных аргументов, кроме случая, когда повторяются ключи словаря.
- В `AsTuple` и `AsStruct` могут быть вызваны без аргументов, а также аргументы могут иметь разные типы.
- Имена полей в `AsStruct` задаются через `AsStruct(field_value AS field_name)`.
- Для создания списка требуется хотя бы один аргумент, если нужно вывести типы элементов. Для создания пустого списка с заданным типом элементов используется функция ListCreate. Можно создать пустой список как вызов `AsList()` без аргументов, в этом случае это выражение будет иметь тип `EmptyList`.
- Для создания словаря требуется хотя бы один аргумент, если нужно вывести типы элементов. Для создания пустого словаря с заданным типом элементов используется функция DictCreate. Можно создать пустой словарь как вызов `AsDict()` без аргументов, в этом случае это выражение будет иметь тип `EmptyDict`.
- Для создания множества требуется хотя бы один аргумент, если нужно вывести типы элементов. Для создания пустого множества с заданным типом элементов используется функция SetCreate. Можно создать пустое множество как вызов `AsSet()` без аргументов, в этом случае это выражение будет иметь тип `EmptyDict`.
- `AsList` выводит общий тип элементов списка. При несовместимых типах генерируется ошибка типизации.
- `AsDict` выводит раздельно общие типы ключей и значений. При несовместимых типах генерируется ошибка типизации.
- `AsSet` выводит общие типы ключей. При несовместимых типах генерируется ошибка типизации.
- `AsListStrict`, `AsDictStrict`, `AsSetStrict` требуют одинакового типа для аргументов.
- В `AsDict` и `AsDictStrict` в качестве аргументов ожидаются `Tuple` из двух элементов: ключ и значение, соответственно. Если ключи повторяются, в словаре останется только значение для первого ключа.
- В `AsSet` и `AsSetStrict` в качестве аргументов ожидаются ключи.

Примеры

```
SELECT
  AsTuple(1, 2, "3") AS `tuple`,
  AsStruct(
    1 AS a,
    2 AS b,
    "3" AS c
  ) AS `struct`,
  AsList(1, 2, 3) AS `list`,
  AsDict(
    AsTuple("a", 1),
    AsTuple("b", 2),
    AsTuple("c", 3)
```

```
  ) AS `dict`,
  AsSet(1, 2, 3) AS `set`
```

Литералы контейнеров

Для некоторых контейнеров возможна операторная форма записи их литеральных значений:

- Кортеж — `(value1, value2...)`;
- Структура — `<|name1: value1, name2: value2...|>`;
- Список — `[value1, value2,...]`;
- Словарь — `{key1: value1, key2: value2...}`;
- Множество — `{key1, key2...}`.

Во всех случаях допускается незначащая хвостовая запятая. Для кортежа с одним элементом эта запятая является обязательной - `(value1,)`.
Для имен полей в литерале структуры допускается использовать выражение, которое можно посчитать в evaluation time, например, строковые литералы, а также идентификаторы (в том числе в backticks).

Для списка внутри используется функция AsList, словаря - AsDict, множества - AsSet, кортежа - AsTuple, структуры - AsStruct.

Примеры

```
$name = "computed " || "member name";
SELECT
  (1, 2, "3") AS `tuple`,
  <|
    `complex member name`: 2.3,
    b: 2,
    $name: "3",
    "inline " || "computed member name": false
  |> AS `struct`,
  [1, 2, 3] AS `list`,
  {
    "a": 1,
    "b": 2,
    "c": 3,
  } AS `dict`,
  {1, 2, 3} AS `set`
```

Variant

`Variant()` создает значение варианта над кортежем или структурой.

Сигнатура

```
Variant(T, String, Type<Variant<...>>)->Variant<...>
```

Аргументы:

- Значение
- Строка с именем поля или индексом кортежа
- Тип варианта

Пример

```
$var_type = Variant<foo: Int32, bar: Bool>;

SELECT
   Variant(6, "foo", $var_type) as Variant1Value,
   Variant(false, "bar", $var_type) as Variant2Value;
```

AsVariant

`AsVariant()` создает значение варианта над структурой с одним полем. Это значение может быть неявно преобразовано к любому варианту над структурой, в которой совпадает для этого имени поля тип данных и могут быть дополнительные поля с другими именами.

Сигнатура

```
AsVariant(T, String)->Variant
```

Аргументы:

- Значение
- Строка с именем поля

Пример

```
SELECT
   AsVariant(6, "foo") as VariantValue
```

Visit, VisitOrDefault

Обрабатывает возможные значения варианта, представленного структурой или кортежем, с использованием предоставленных функций-обработчиков для каждого из его полей/элементов.

Сигнатура

```
Visit(Variant<key1: K1, key2: K2, ...>, K1->R AS key1, K2->R AS key2, ...)->R
Visit(Variant<K1, K2, ...>, K1->R, K2->R, ...)->R

VisitOrDefault(Variant<K1, K2, ...>{Flags:AutoMap}, R, [K1->R, [K2->R, ...]])->R
VisitOrDefault(Variant<key1: K1, key2: K2, ...>{Flags:AutoMap}, R, [K1->R AS key1, [K2->R AS key2, ...]])->R
```

Аргументы

- Для варианта над структурой функция принимает сам вариант в качестве позиционного аргумента и по одному именованному аргументу-обработчику для каждого поля этой структуры.
- Для варианта над кортежем функция принимает сам вариант и по одному обработчику на каждый элемент кортежа в качестве позиционных аргументов.
- Модификация `VisitOrDefault` принимает дополнительный позиционный аргумент (на втором месте), представляющий значение по умолчанию, и позволяет не указывать некоторые обработчики.

Пример

```
$vartype = Variant<num: Int32, flag: Bool, str: String>;
$handle_num = ($x) -> { return 2 * $x; };
$handle_flag = ($x) -> { return If($x, 200, 10); };
$handle_str = ($x) -> { return Unwrap(CAST(LENGTH($x) AS Int32)); };

$visitor = ($var) -> { return Visit($var, $handle_num AS num, $handle_flag AS flag, $handle_str AS str); };
SELECT
    $visitor(Variant(5, "num", $vartype)),                -- 10
    $visitor(Just(Variant(True, "flag", $vartype))),      -- Just(200)
    $visitor(Just(Variant("somestr", "str", $vartype))),  -- Just(7)
    $visitor(Nothing(OptionalType($vartype))),            -- Nothing(Optional<Int32>)
    $visitor(NULL)                                        -- NULL
;
```

VariantItem

Возвращает значение гомогенного варианта (т.е. содержащего поля/элементы одного типа).

Сигнатура

```
VariantItem(Variant<key1: K, key2: K, ...>{Flags:AutoMap})->K
VariantItem(Variant<K, K, ...>{Flags:AutoMap})->K
```

Пример

```
$vartype1 = Variant<num1: Int32, num2: Int32, num3: Int32>;
SELECT
    VariantItem(Variant(7, "num2", $vartype1)),          -- 7
    VariantItem(Just(Variant(5, "num1", $vartype1))),    -- Just(5)
    VariantItem(Nothing(OptionalType($vartype1))),       -- Nothing(Optional<Int32>)
    VariantItem(NULL)                                    -- NULL
;
```

Way

Возвращает активное поле (активный индекс) варианта поверх структуры (кортежа).

Сигнатура

```
Way(Variant<key1: K1, key2: K2, ...>{Flags:AutoMap})->Utf8
Way(Variant<K1, K2, ...>{Flags:AutoMap})->Uint32
```

Пример

```
$vr = Variant(1, "0", Variant<Int32, String>);
$vrs = Variant(1, "a", Variant<a:Int32, b:String>);


SELECT Way($vr);  -- 0
SELECT Way($vrs); -- "a"
```

DynamicVariant

Создает экземпляр гомогенного варианта (т.е. содержащего поля/элементы одного типа), причем индекс или поле варианта можно задавать динамически. При несуществующем индексе или имени поля будет возвращен `NULL`.
Обратная функция - VariantItem.

Сигнатура

```
DynamicVariant(item:T,index:Uint32?,Variant<T, T, ...>)->Optional<Variant<T, T, ...>>
DynamicVariant(item:T,index:Utf8?,Variant<key1: T, key2: T, ...>)->Optional<Variant<key1: T, key2: T, ...>>
```

Пример

```
$dt = Int32;
$tvt = Variant<$dt,$dt>;
SELECT ListMap([(10,0u),(20,2u),(30,NULL)],($x)->(DynamicVariant($x.0,$x.1,$tvt))); -- [0: 10,NULL,NULL]

$dt = Int32;
$svt = Variant<x:$dt,y:$dt>;
SELECT ListMap([(10,'x'u),(20,'z'u),(30,NULL)],($x)->(DynamicVariant($x.0,$x.1,$svt))); -- [x: 10,NULL,NULL]
```

Enum

`Enum()` создает значение перечисления.

Сигнатура

```
Enum(String, Type<Enum<...>>)->Enum<...>
```

Аргументы:

- Строка с именем поля
- Тип перечисления

Пример

```
$enum_type = Enum<Foo, Bar>;
SELECT
   Enum("Foo", $enum_type) as Enum1Value,
   Enum("Bar", $enum_type) as Enum2Value;
```

AsEnum

`AsEnum()` создает значение перечисления с одним элементом. Это значение может быть неявно преобразовано к любому перечислению, содержащему такое имя.

Сигнатура

```
AsEnum(String)->Enum<'tag'>
```

Аргументы:

- Строка с именем элемента перечисления

Пример

```
SELECT
   AsEnum("Foo");
```

AsTagged, Untag

Оборачивает значение в Tagged тип данных с указанной меткой с сохранением физического типа данных. `Untag` — обратная операция.

Сигнатура

```
AsTagged(T, tagName:String)->Tagged<T,tagName>
AsTagged(T?, tagName:String)->Tagged<T,tagName>?

Untag(Tagged<T, tagName>)->T
Untag(Tagged<T, tagName>?)->T?
```

Обязательные аргументы:

1. Значение произвольного типа;
2. Имя метки.

Возвращает копию значения из первого аргумента с указанной меткой в типе данных.

Примеры сценариев использования:

- Возвращение на клиент для отображения в веб-интерфейсе медиа-файлов из base64-encoded строк.
- Дополнительные уточнения на уровне типов возвращаемых колонок.

TableRow, JoinTableRow

Получение всей строки таблицы целиком в виде структуры. Аргументов нет. `JoinTableRow` в случае `JOIN`-ов всегда возвращает структуру с префиксами таблиц.

Сигнатура

```
TableRow()->Struct
```

Пример

```
SELECT TableRow() FROM my_table;
```

Ensure...

Проверка пользовательских условий:

- `Ensure()` — проверка верности предиката во время выполнения запроса.
- `EnsureType()` — проверка точного соответствия типа выражения указанному.
- `EnsureConvertibleTo()` — мягкая проверка соответствия типа выражения, работающая по тем же правилам, что и неявное приведение типов.

Если проверка не прошла успешно, то весь запрос завершается с ошибкой.

Сигнатуры

```
Ensure(T, Bool, String)->T
EnsureType(T, Type<T>, String)->T
EnsureConvertibleTo(T, Type<T>, String)->T
```

Аргументы:

1. Выражение, которое станет результатом вызова функции в случае успеха проверки. Оно же подвергается проверке на тип данных в соответствующих функциях.
2. В Ensure — булевый предикат, который проверяется на `true`. В остальных функциях — тип данных, который может быть получен через предназначенные для этого функции, либо строковый литерал с текстовым описанием типа.
3. Опциональная строка с комментарием к ошибке, которая попадет в общее сообщение об ошибке при завершении запроса. Для проверок типов не может использовать сами данные, так как они выполняются на этапе валидации запроса, а для Ensure — может быть произвольным выражением.

Примеры

```
SELECT Ensure(
    value,
    value < 100,
    "value out or range"
) AS value FROM my_table;
```

```
SELECT EnsureType(
    value,
    TypeOf(other_value),
    "expected value and other_value to be of same type"
) AS value FROM my_table;
```

```
SELECT EnsureConvertibleTo(
    value,
    Double?,
    "expected value to be numeric"
) AS value FROM my_table;
```

AssumeStrict

Сигнатура

```
AssumeStrict(T)->T
```

Функция `AssumeStrict` возвращает свой аргумент. Использование этой функции – способ сказать оптимизатору YQL, что выражение в аргументе является *строгим*, т.е. свободным от ошибок времени выполнения.
Большинство встроенных функций и операторов YQL являются строгими, но есть исключения – например Unwrap и Ensure.
Кроме того, нестрогим выражением считается вызов UDF.

Если есть уверенность, что при вычислении выражения ошибок времени выполнения на самом деле не возникает, то имеет смысл использовать `AssumeStrict`.

Пример

```
SELECT * FROM T1 AS a JOIN T2 AS b USING(key)
WHERE AssumeStrict(Unwrap(CAST(a.key AS Int32))) == 1;
```

В данном примере мы считаем что все значения текстовой колонки `a.key` в таблице `T1` являются валидными числами, поэтому Unwrap не приводит к ошибке.
При наличии `AssumeStrict` оптимизатор сможет выполнить сначала фильтрацию, а потом JOIN.
Без `AssumeStrict` такая оптимизация не выполняется – оптимизатор обязан учитывать ситуацию, при которой в колонке `a.key` есть нечисловые значения, которые отфильтровываются `JOIN`ом.

Likely

Сигнатура

```
Likely(Bool)->Bool
```

```
Likely(Bool?)->Bool?
```

Функция `Likely` возвращает свой аргумент. Функция является подсказкой оптимизатору и говорит о том, что в большинстве случаев ее аргумент будет иметь значение `True`.
Например, наличие такой функции в `WHERE` означает что фильтр является слабоселективным.

Пример

```
SELECT * FROM T1 AS a JOIN T2 AS b USING(key)
WHERE Likely(a.amount > 0)  -- почти всегда верно
```

При наличии `Likely` оптимизатор не будет стараться выполнить фильтрацию перед `JOIN`.

EvaluateExpr, EvaluateAtom

Возможность выполнить выражение до начала основного расчета и подставить его результат в запрос как литерал (константу). Во многих контекстах, где в стандартном SQL ожидалась бы только константа (например, в именах таблиц, количестве строк в LIMIT и т.п.) этот функционал активируется неявным образом автоматически.

EvaluateExpr может использоваться в тех местах, где грамматикой уже ожидается выражение. Например, с его помощью можно:

- округлить текущее время до дней, недель или месяцев и подставить в запрос, что затем позволит корректно работать кешированию запросов, хотя обычно использование функций для получения текущего времени его полностью отключает;
- сделать тяжелое вычисление с небольшим результатом один раз на запрос вместо одного раза на каждую джобу.

Единственный аргумент у обоих функций — само выражение для вычисления и подстановки.

Ограничения:

- выражение не должно приводить к запуску MapReduce операций;
- данный функционал полностью заблокирован в YQL over YDB.

Примеры

```
$now = CurrentUtcDate();
SELECT EvaluateExpr(
    DateTime::MakeDate(DateTime::StartOfWeek($now)
    )
);
```

Литералы простых типов

Для простых типов могут быть созданы литералы на основании строковых литералов.

Синтаксис

`<Простой тип>( <строка>[, <дополнительные атрибуты>] )`

В отличие от `CAST("myString" AS MyType)`:

- Проверка на приводимость литерала к требуемому типу происходит на этапе валидации;
- Результат не является optional.

Аргумент `<строка>` должен быть строковым литералом. Выражения, включая переменные, колонки, параметры запроса и результаты функций, не поддерживаются. Для преобразования значения во время выполнения запроса используйте `CAST(выражение AS <Простой тип>)`. Например, вычисляемое значение `String` следует преобразовывать в `Uuid` через `CAST(выражение AS Uuid)`, а не через `Uuid(выражение)`.

```
$uuid = Uuid("550e8400-e29b-41d4-a716-446655440000");
$uuidAsString = CAST($uuid AS String);

SELECT CAST($uuidAsString AS Uuid) = $uuid; -- true
```

Для типов данных `Date`, `Datetime`, `Timestamp` и `Interval` поддерживаются литералы только в формате, соответствующем ISO 8601. У `Interval` есть следующие отличия от стандарта:

- поддерживается отрицательный знак для сдвигов в прошлое;
- микросекунды могут быть записаны как дробная часть секунд;
- единицы измерения больше недель не доступны;
- не поддерживаются варианты с началом/концом интервала, а также повторами.

Для типов данных `TzDate`, `TzDatetime`, `TzTimestamp` литералы также задаются в формате, соответствующем ISO 8601, но вместо опционального суффикса Z через запятую указывается IANA имя временной зоны, например, GMT или Europe/Moscow.

Для параметрического типа данных Decimal дополнительно указывается два аргумента:

- общее число десятичных знаков (до 35, включительно);
- число десятичных знаков после запятой (из общего числа, то есть строго не больше предыдущего аргумента).

Примеры

```
SELECT
  Bool("true"),
  Uint8("0"),
  Int32("-1"),
  Uint32("2"),
  Int64("-3"),
```

```
  Uint64("4"),
  Float("-5"),
  Double("6"),
  Decimal("1.23", 5, 2), -- до 5 десятичных знаков, из которых 2 после запятой
  DyNumber("1"),
  String("foo"),
  Utf8("привет"),
  Yson("<a=1>[3;%false]"),
  Json(@@{"a":1,"b":null}@@),
  Date("2017-11-27"),
  Datetime("2017-11-27T13:24:00Z"),
  Timestamp("2017-11-27T13:24:00.123456Z"),
  Interval("P1W2DT2H3M4.567890S"),
  TzDate("2017-11-27,Europe/Moscow"),
  TzDatetime("2017-11-27T13:24:00,America/Los_Angeles"),
  TzTimestamp("2017-11-27T13:24:00.123456,GMT"),
  Uuid("f9d5cc3f-f1dc-4d9c-b97e-766e57ca4ccb");
```

ToBytes и FromBytes

Конвертация простых типов данных в строку со своим бинарным представлением и обратно. Числа представляются в little endian.

Сигнатуры

```
ToBytes(T)->String
ToBytes(T?)->String?

FromBytes(String, Type<T>)->T?
FromBytes(String?, Type<T>)->T?
```

Примеры

```
SELECT
    ToBytes(123), -- "{\u0000\u0000\u0000"
    String::HexEncode(ToBytes(123)) -- "7B000000"
    FromBytes(
        "\xd2\x02\x96\x49\x00\x00\x00\x00",
        Uint64
    ); -- 1234567890ul
```

ByteAt

Получение значение байта в строке по индексу от её начала. В случае некорректного индекса возвращается `NULL`.

Сигнатура

```
ByteAt(String, Uint32)->Uint8
ByteAt(String?, Uint32)->Uint8?

ByteAt(Utf8, Uint32)->Uint8
ByteAt(Utf8?, Uint32)->Uint8?
```

Аргументы:

1. Строка: `String` или `Utf8`;
2. Индекс: `Uint32`.

Примеры

```
SELECT
    ByteAt("foo", 0), -- 102
    ByteAt("foo", 1), -- 111
    ByteAt("foo", 9); -- NULL
```

...Bit

`TestBit()`, `ClearBit()`, `SetBit()` и `FlipBit()` - проверить, сбросить, установить или инвертировать бит в беззнаковом числе по указанному порядковому номеру бита.

Сигнатуры

```
TestBit(T, Uint8)->Bool
TestBit(T?, Uint8)->Bool?

ClearBit(T, Uint8)->T
ClearBit(T?, Uint8)->T?

SetBit(T, Uint8)->T
SetBit(T?, Uint8)->T?

FlipBit(T, Uint8)->T
FlipBit(T?, Uint8)->T?
```

Аргументы:

1. Беззнаковое число, над которым выполнять требуемую операцию. TestBit также реализован и для строк.
2. Номер бита.

TestBit возвращает `true/false`. Остальные функции возвращают копию своего первого аргумента с проведенным соответствующим преобразованием.

Примеры

```
SELECT
    TestBit(1u, 0), -- true
    SetBit(8u, 0); -- 9
```

Abs

Абсолютное значение числа.

Сигнатура

```
Abs(T)->T
Abs(T?)->T?
```

Примеры

```
SELECT Abs(-123); -- 123
```

Just

`Just()` - Изменить тип данных значения на optional от текущего типа данных (то есть `T` превращается в `T?`).

Сигнатура

```
Just(T)->T?
```

Примеры

```
SELECT
  Just("my_string"); --  String?
```

Unwrap

`Unwrap()` - Преобразование значения optional типа данных в соответствующий не-optional тип с ошибкой времени выполнений, если в данных оказался `NULL`. Таким образом, `T?` превращается в `T`.

Если значение не является optional, то функция возвращает свой первый аргумент без изменений.

Сигнатура

```
Unwrap(T?)->T
Unwrap(T?, Utf8)->T
Unwrap(T?, String)->T
```

Аргументы:

1. Значение для преобразования;
2. Опциональная строка с комментарием для текста ошибки.

Обратная операция — Just.

Примеры

```
$value = Just("value");

SELECT Unwrap($value, "Unexpected NULL for $value");
```

Nothing

`Nothing()` - Создать пустое значение указанного Optional типа данных.

Сигнатура

```
Nothing(Type<T?>)->T?
```

Примеры

```
SELECT
  Nothing(String?); -- пустое значение (NULL) с типом String?
```

Подробнее о ParseType и других функциях для работы с типами данных.

Callable

Создать вызываемое значение с заданной сигнатурой из лямбда-функции. Обычно используется для того, чтобы размещать вызываемые значения в контейнерах.

Сигнатура

```
Callable(Type<Callable<(...)->T>>, lambda)->Callable<(...)->T>
```

Аргументы:

1. Тип;
2. Лямбда-функция.

Примеры

```
$lambda = ($x) -> {
    RETURN CAST($x as String)
};

$callables = AsTuple(
    Callable(Callable<(Int32)->String>, $lambda),
    Callable(Callable<(Bool)->String>, $lambda),
);

SELECT $callables.0(10), $callables.1(true);
```

Pickle, Unpickle

`Pickle()` и `StablePickle()` сериализуют произвольный объект в последовательность байт, если это возможно. Типовыми несериализуемыми объектами являются Callable и Resource. Формат сериализации не версионируется, допускается использовать в пределах одного запроса. Для типа Dict функция StablePickle предварительно сортирует ключи, а для Pickle порядок элементов словаря в сериализованном представлении не определен.

`Unpickle()` — обратная операция (десериализация), где первым аргументом передается тип данных результата, а вторым — строка с результатом `Pickle()` или `StablePickle()`.

Сигнатуры

```
Pickle(T)->String
StablePickle(T)->String
Unpickle(Type<T>, String)->T
```

Примеры

```
SELECT *
FROM my_table
WHERE Digest::MurMurHash32(
        Pickle(TableRow())
    ) % 10 == 0;

$buf = Pickle(123);
SELECT Unpickle(Int32, $buf);
```

StaticMap

Преобразует структуру или кортеж, применяя лямбду к каждому элементу.

Сигнатура

```
StaticMap(Struct<...>, lambda)->Struct<...>
StaticMap(Tuple<...>, lambda)->Tuple<...>
```

Аргументы:

- Структура или кортеж;
- Лямбда для обработки элементов.

Результат: структура или кортеж с аналогичным первому аргументу количеством и именованием элементов, а типы данных элементов определяются результатами лямбды.

Примеры

```
SELECT *
FROM (
    SELECT
        StaticMap(TableRow(), ($item) -> {
            return CAST($item AS String);
        })
    FROM my_table
) FLATTEN COLUMNS; -- преобразование всех колонок в строки
```

StaticZip

Поэлементно "склеивает" структуры или кортежи. Все аргументы (один и более) должны быть либо структурами с одинаковым набором полей, либо кортежами одинаковой длины.
Результататом будет соответственно структура или кортеж.
Каждый элемент результата – кортеж с соответствующими элементами из аргументов.

Сигнатура

```
StaticZip(Struct, Struct)->Struct
StaticZip(Tuple, Tuple)->Tuple
```

Примеры

```
$one = <|k1:1, k2:2.0|>;
$two = <|k1:3.0, k2:4|>;

-- поэлементное сложение двух структур
SELECT StaticMap(StaticZip($one, $two), ($tuple)->($tuple.0 + $tuple.1)) AS sum;
```

StaticFold, StaticFold1

```
StaticFold(obj:Struct/Tuple, initVal, updateLambda)
StaticFold1(obj:Struct/Tuple, initLambda, updateLambda)
```

Статическая левоассоциативная свертка структуры или кортежа.
Для кортежей свертка производится в порядке от меньшего индекса к большему, для структур порядок не гарантируется.

- `obj` - объект, элементы которого нужно свернуть
- `initVal` - *(для StaticFold)* исходное состояние свертки
- `initLambda` - *(для StaticFold1)* функция для получения исходного состояния по первому элементу
- `updateLambda` - функция обновления состояния (принимает в аргументах следующий элемент объекта и предыдущее состояние)

`StaticFold(<|key_1:$el_1, key_2:$el_2, ..., key_n:$el_n|>, $init, $f)` преобразуется в свертку:

```
$f($el_n, ...$f($el_2, $f($init, el_1))...)
```

`StaticFold1(<|key_1:$el_1, key_2:$el_2, ..., key_n:$el_n|>, $f0, $f)`:

```
$f($el_n, ...$f($el_2, $f($f0($init), el_1))...)
```

`StaticFold1(<||>, $f0, $f)` вернет `NULL`.

Аналогично работает и с кортежами.

AggregationFactory

Создать фабрику для агрегатных функций для того чтобы разделить процесс описания того, как агрегировать данные, и то, к каким данным это применять.

Аргументы:

1. Строка в кавычках, являющаяся именем агрегатной функции, например "MIN".
2. Опциональные параметры агрегатной функции, которые не зависят от данных. Например, значение percentile в PERCENTILE.

Полученную фабрику можно использовать как второй параметр функции AGGREGATE_BY.
Если агрегатная функция работает на двух колонках вместо одной, как например, MIN_BY, то в AGGREGATE_BY первым аргументом передается `Tuple` из двух значений. Подробнее это указано при описании такой агрегатной функции.

Примеры

```
$factory = AggregationFactory("MIN");
SELECT
    AGGREGATE_BY(value, $factory) AS min_value -- применить MIN агрегацию к колонке value
FROM my_table;
```

AggregateTransformInput

`AggregateTransformInput()` преобразует фабрику для агрегатных функций, например, полученную через функцию AggregationFactory в другую фабрику, в которой перед началом выполнения агрегации производится указанное преобразование входных элементов.

Аргументы:

1. Фабрика для агрегатных функций;
2. Лямбда функция с одним аргументом, преобразующая входной элемент.

Примеры

```
$f = AggregationFactory("sum");
$g = AggregateTransformInput($f, ($x) -> (cast($x as Int32)));
$h = AggregateTransformInput($f, ($x) -> ($x * 2));
```

```
SELECT ListAggregate([1,2,3], $f); -- 6
SELECT ListAggregate(["1","2","3"], $g); -- 6
SELECT ListAggregate([1,2,3], $h); -- 12
```

AggregateTransformOutput

`AggregateTransformOutput()` преобразует фабрику для агрегатных функций, например, полученную через функцию AggregationFactory в другую фабрику, в которой после окончания выполнения агрегации производится указанное преобразование результата.

Аргументы:

1. Фабрика для агрегатных функций;
2. Лямбда функция с одним аргументом, преобразующая результат.

Примеры

```
$f = AggregationFactory("sum");
$g = AggregateTransformOutput($f, ($x) -> ($x * 2));
SELECT ListAggregate([1,2,3], $f); -- 6
SELECT ListAggregate([1,2,3], $g); -- 12
```

AggregateFlatten

Адаптирует фабрику для агрегатных функций, (например, полученную через функцию AggregationFactory) так, чтобы выполнять агрегацию над входными элементами - списками. Эта операция похожа на FLATTEN LIST BY - производится агрегация каждого элемента списка.

Аргументы:

1. Фабрика для агрегатных функций.

Примеры

```
$i = AggregationFactory("AGGREGATE_LIST_DISTINCT");
$j = AggregateFlatten($i);
SELECT AggregateBy(x, $j) from (
   SELECT [1,2] as x
   union all
   SELECT [2,3] as x
); -- [1, 2, 3]
```

Полнотекстовый поиск

Функции `FulltextMatch` и `FulltextScore` предназначены для выполнения полнотекстового поиска в YDB **через полнотекстовый индекс**.

> **Примечание**
>
> Эти функции используют полнотекстовый индекс только при выполнении запроса через `VIEW IndexName`. См. VIEW (Полнотекстовый индекс) и Полнотекстовые индексы.

FulltextMatch

`FulltextMatch(text, query)` фильтрует строки по совпадению текста с полнотекстовым запросом.

```
SELECT id, body
FROM articles VIEW ft_idx
WHERE FulltextMatch(body, "машинное обучение");
```

Только первые два аргумента могут быть позиционными. Дополнительные параметры нужно передавать как **именованные аргументы**.

Поддерживаемые именованные аргументы:

- `Mode` (String): вид запроса:
  - `Keywords` (по умолчанию) — текст запроса разбивается на отдельные термы; логика их объединения определяется `DefaultOperator`
  - `Wildcard` — поиск с подстановочными символами: `%` заменяет любую подстроку, `_` — один символ (аналогично `LIKE`); требует N-граммного индекса
- `DefaultOperator` (String): оператор объединения термов в режиме `Keywords`:
  - `And` (по умолчанию) — все термы запроса должны присутствовать в тексте
  - `Or` — термы делятся на **обязательные** (с префиксом `+`, должны присутствовать в каждом результате) и **необязательные** (без префикса, учитываются в `MinimumShouldMatch`). Без префикса `+` все термы считаются необязательными, и достаточно совпадения хотя бы одного
- `MinimumShouldMatch` (String): минимальное число **необязательных** термов, которые должны совпасть при `DefaultOperator = "Or"` — задаётся как абсолютное число (например, `"3"`) или процент от числа необязательных термов (например, `"50%"`). Обязательные (`+`) термы в счёт не входят

Значения аргументов `Mode` и `DefaultOperator` регистронезависимы.

Пример для Wildcard

Если индекс создан с фильтрацией N-грамм, можно использовать шаблоны с `%` и `_` (по аналогии с `LIKE`).

```
SELECT id, title
FROM articles VIEW ft_idx
WHERE FulltextMatch(body, "маш% обу%ние", "Wildcard" AS Mode)
LIMIT 20;
```

Шаблон должен содержать фрагмент для поиска не короче `filter_ngram_min_length`; `%` или более короткий фрагмент нельзя использовать для поиска по индексу.

Пример для Keywords

Запрос из 8 слов, где требуется совпадение не менее половины слов:

```
SELECT id
FROM articles VIEW ft_idx
WHERE FulltextMatch(
  body,
  "машинное обучение нейронные сети глубокое обучение большие данные рекомендации поиск",
  "Keywords" AS Mode,
  "Or" AS DefaultOperator,
  "50%" AS MinimumShouldMatch
);
```

В этом примере `Mode` установлен в `Keywords` (запрос трактуется как набор термов), `DefaultOperator` установлен в `Or` (разрешается совпадение по любому подмножеству термов), а `MinimumShouldMatch` установлен в `50%` (требуется совпадение минимум 4 термов из 8). Это помогает отсеять документы, где встретилось лишь одно-два слова из длинного запроса.

Пример с обязательным термом (`+`)

Добавьте `+` перед термом в режиме `Keywords` с `DefaultOperator = "Or"`, чтобы сделать его обязательным для каждого результата. Этот синтаксис следует обязательным термам в Lucene.

```
SELECT id
FROM articles VIEW ft_idx
WHERE FulltextMatch(
  body,
  "+машинное обучение нейронные сети",
  "Or" AS DefaultOperator,
  "1" AS MinimumShouldMatch
);
```

Запрос вернёт документы, содержащие «машинное» (обязательный) и хотя бы один из «обучение», «нейронные», «сети» (необязательные). `MinimumShouldMatch` учитывает только необязательные термы. Если все термы начинаются с `+`, запрос работает как строгий AND.

`MinimumShouldMatch` учитывает различные необязательные термы после анализа. Процент округляется вниз, но требуется хотя бы одно совпадение. Допустимы отрицательные целые: `"-1"` означает все необязательные термы, кроме одного. Требуется `DefaultOperator="Or"`.

FulltextScore

`FulltextScore(text, query)` возвращает оценку релевантности на основе алгоритма BM25 и может использоваться для ранжирования результатов.

Требует индекс типа `fulltext_relevance`.

```
SELECT id, FulltextScore(body, "машинное обучение") AS relevance
FROM articles VIEW ft_idx
WHERE FulltextScore(body, "машинное обучение") > 0
ORDER BY relevance DESC
LIMIT 10;
```

Только первые два аргумента могут быть позиционными. Дополнительные параметры нужно передавать как **именованные аргументы**.

Поддерживаемые именованные аргументы:

- `DefaultOperator` (String): оператор объединения термов — `And` (по умолчанию, все термы должны присутствовать) или `Or` (термы делятся на **обязательные** с префиксом `+` и **необязательные** без префикса; каждый обязательный должен присутствовать, а из необязательных должны совпасть не менее `MinimumShouldMatch`)
- `MinimumShouldMatch` (String): при `DefaultOperator = "Or"` — минимальное число **необязательных** термов, которые должны совпасть; задаётся как абсолютное число (например, `"2"`) или процент от числа необязательных термов (например, `"50%"`). Обязательные (`+`) термы в счёт не входят
- `K1` (Double или Float): параметр насыщения частоты терма в BM25 — определяет, насколько сильно влияет повторное появление терма на итоговую оценку; по умолчанию `1.2`
- `B` (Double или Float): параметр нормализации длины документа в BM25 — `0.0` отключает нормализацию, `1.0` полностью нормализует по длине документа; по умолчанию `0.75`

Пример:

```
SELECT id,
       FulltextScore(body, "машинное обучение", "Or" AS DefaultOperator, "1" AS MinimumShouldMatch, 0.75 AS K
1, 1.2 AS B) AS relevance
FROM articles VIEW ft_idx
WHERE FulltextScore(body, "машинное обучение", "Or" AS DefaultOperator, "1" AS MinimumShouldMatch, 0.75 AS K
1, 1.2 AS B) > 0
ORDER BY relevance DESC;
```

Примечание

`MinimumShouldMatch` поддерживается только когда `DefaultOperator` установлен в `Or`. Когда `DefaultOperator` установлен в `And`, используйте `FulltextScore` без `MinimumShouldMatch`.

Агрегатные функции

COUNT

Сигнатура

```
COUNT(*)->Uint64
COUNT(T)->Uint64
COUNT(T?)->Uint64
```

Подсчет количества строк в строковой или колоночной таблице (если в качестве аргумента указана `*` или константа) или непустых значений в столбце таблицы (если в качестве аргумента указано имя столбца).

Как и другие агрегатные функции, может использоваться в сочетании с GROUP BY для получения статистики по частям таблицы, соответствующим значениям в столбцах, по которым идет группировка. А модификатор DISTINCT позволяет посчитать число уникальных значений.

Примеры

```
SELECT COUNT(*) FROM my_table;
```

```
SELECT key, COUNT(value) FROM my_table GROUP BY key;
```

```
SELECT COUNT(DISTINCT value) FROM my_table;
```

MIN и MAX

Сигнатура

```
MIN(T?)->T?
MIN(T)->T?
MAX(T?)->T?
MAX(T)->T?
```

Минимальное или максимальное значение.

В качестве аргумента допустимо произвольное вычислимое выражение с результатом, допускающим сравнение значений.

Примеры

```
SELECT MIN(value), MAX(value) FROM my_table;
```

SUM

Сигнатура

```
SUM(Unsigned?)->Uint64?
SUM(Signed?)->Int64?
SUM(Interval?)->Interval?
SUM(Decimal(N, M)?)->Decimal(35, M)?
```

Сумма чисел.

В качестве аргумента допустимо произвольное вычислимое выражение с числовым результатом или типом `Interval`.

Целые числа автоматически расширяются до 64 бит, чтобы уменьшить риск переполнения.

```
SELECT SUM(value) FROM my_table;
```

AVG

Сигнатура

```
AVG(Double?)->Double?
AVG(Interval?)->Interval?
AVG(Decimal(N, M)?)->Decimal(N, M)?
```

Арифметическое среднее.

В качестве аргумента допустимо произвольное вычислимое выражение с числовым результатом или типом `Interval`.

Целочисленные значения и интервалы времени автоматически приводятся к Double.

Примеры

```
SELECT AVG(value) FROM my_table;
```

COUNT_IF

Сигнатура

```
COUNT_IF(Bool?)->Uint64?
```

Количество строк, для которых указанное в качестве аргумента выражение истинно (результат вычисления выражения — true).

Значение `NULL` приравнивается к `false` (в случае, если тип аргумента `Bool?`).

Функция *не* выполняет неявного приведения типов к булевым для строк и чисел.

Примеры

```
SELECT
  COUNT_IF(value % 2 == 1) AS odd_count
FROM my_table;
```

Примечание

Если нужно посчитать число уникальных значений на строках, где выполняется условие, то в отличие от остальных агрегатных функций модификатор DISTINCT тут не поможет, так как в аргументах нет никаких значений. Для получения данного результата можно использовать запрос следующего вида:

```
SELECT
    COUNT(DISTINCT IF(value % 2 == 1, value))
FROM my_table;
```

SUM_IF и AVG_IF

Сигнатура

```
SUM_IF(Unsigned?, Bool?)->Uint64?
SUM_IF(Signed?, Bool?)->Int64?
SUM_IF(Interval?, Bool?)->Interval?

AVG_IF(Double?, Bool?)->Double?
```

Сумма или арифметическое среднее, но только для строк, удовлетворяющих условию, переданному вторым аргументом.

Таким образом, `SUM_IF(value, condition)` является чуть более короткой записью для `SUM(IF(condition, value))`, аналогично для `AVG`. Расширение типа данных аргумента работает так же аналогично одноименным функциям без суффикса.

Примеры

```
SELECT
    SUM_IF(value, value % 2 == 1) AS odd_sum,
    AVG_IF(value, value % 2 == 1) AS odd_avg,
FROM my_table;
```

При использовании фабрики агрегационной функции в качестве первого аргумента AGGREGATE_BY передается `Tuple` из значения и предиката.

```
$sum_if_factory = AggregationFactory("SUM_IF");
$avg_if_factory = AggregationFactory("AVG_IF");

SELECT
    AGGREGATE_BY(AsTuple(value, value % 2 == 1), $sum_if_factory) AS odd_sum,
    AGGREGATE_BY(AsTuple(value, value % 2 == 1), $avg_if_factory) AS odd_avg
FROM my_table;
```

SOME

Сигнатура

```
SOME(T?)->T?
SOME(T)->T?
```

Получить значение указанного в качестве аргумента выражения для одной из строк таблицы. Не дает никаких гарантий о том, какая именно строка будет использована.

Из-за отсутствия гарантий `SOME` вычислительно дешевле, чем часто использующиеся в подобных ситуациях MIN и MAX.

Примеры

```
SELECT
  SOME(value)
FROM my_table;
```

Внимание

При вызове агрегатной функции `SOME` несколько раз **не** гарантируется, что все значения результатов будут взяты с одной строки исходной таблицы. Для получения данной гарантии, нужно запаковать значения в какой-либо из контейнеров и передавать в `SOME` уже его. Например, для структуры это можно сделать с помощью AsStruct

CountDistinctEstimate, HyperLogLog и HLL

Сигнатура

```
CountDistinctEstimate(T)->Uint64?
HyperLogLog(T)->Uint64?
HLL(T)->Uint64?
```

Примерная оценка числа уникальных значений по алгоритму HyperLogLog. Логически делает то же самое, что и COUNT(DISTINCT ...), но работает значительно быстрее ценой некоторой погрешности.

Аргументы:

1. Значение для оценки;
2. Точность (от 4 до 18 включительно, по умолчанию 14).

Выбор точности позволяет разменивать дополнительное потребление вычислительных ресурсов и оперативной памяти на уменьшение погрешности.

На данный момент все три функции являются алиасами, но в будущем `CountDistinctEstimate` может начать использовать другой алгоритм.

Примеры

```
SELECT
  CountDistinctEstimate(my_column)
FROM my_table;
```

```
SELECT
  HyperLogLog(my_column, 4)
FROM my_table;
```

AGGREGATE_LIST

Сигнатура

```
AGGREGATE_LIST(T? [, limit:Uint64])->List<T>
AGGREGATE_LIST(T [, limit:Uint64])->List<T>
AGGREGATE_LIST_DISTINCT(T? [, limit:Uint64])->List<T>
AGGREGATE_LIST_DISTINCT(T [, limit:Uint64])->List<T>
```

Получить все значения столбца в виде списка. В сочетании с `DISTINCT` возвращает только уникальные значения. Опциональный второй параметр задает максимальное количество получаемых значений. Значение 0 равносильно отсутствию лимита.

Если заранее известно, что уникальных значений не много, то лучше воспользоваться агрегатной функцией `AGGREGATE_LIST_DISTINCT`, которая строит тот же результат в памяти (которой при большом числе уникальных значений может не хватить).

Порядок элементов в результирующем списке зависит от реализации и снаружи не задается. Чтобы получить упорядоченный список, необходимо отсортировать результат, например с помощью ListSort.

Чтобы получить список нескольких значений с одной строки, важно *НЕ* использовать функцию `AGGREGATE_LIST` несколько раз, а сложить все нужные значения в контейнер, например через AsList или AsTuple и передать этот контейнер в один вызов `AGGREGATE_LIST`.

Например, можно использовать в сочетании с `DISTINCT` и функцией String::JoinFromList (аналог `','.join(list)` из Python) для распечатки в строку всех значений, которые встретились в столбце после применения GROUP BY.

Примеры

```
SELECT
   AGGREGATE_LIST( region ),
   AGGREGATE_LIST( region, 5 ),
   AGGREGATE_LIST( DISTINCT region ),
   AGGREGATE_LIST_DISTINCT( region ),
   AGGREGATE_LIST_DISTINCT( region, 5 )
FROM users
```

```
-- Аналог GROUP_CONCAT из MySQL
SELECT
    String::JoinFromList(CAST(AGGREGATE_LIST(region, 2) AS List<String>), ",")
FROM users
```

Существует также короткая форма записи этих функций - `AGG_LIST` и `AGG_LIST_DISTINCT`.

Внимание

Выполняется **НЕ** ленивым образом, поэтому при использовании нужно быть уверенным, что список получится разумных размеров, примерно в пределах тысячи элементов. Чтобы подстраховаться, можно воспользоваться вторым опциональным числовым аргументом, который включает ограничение на число элементов в списке.

MAX_BY и MIN_BY

Сигнатура

```
MAX_BY(T1?, T2)->T1?
MAX_BY(T1, T2)->T1?
MAX_BY(T1, T2, limit:Uint64)->List<T1>?

MIN_BY(T1?, T2)->T1?
MIN_BY(T1, T2)->T1?
MIN_BY(T1, T2, limit:Uint64)->List<T1>?
```

Вернуть значение первого аргумента для строки таблицы, в которой второй аргумент оказался минимальным/максимальным.

Опционально можно указать третий аргумент N, который влияет на поведение в случае, если в таблице есть несколько строк с одинаковым минимальным или максимальным значением:

- Если N не указано — будет возвращено значение одной из строк, а остальные отбрасываются.
- Если N указано — будет возвращен список со всеми значениями, но не более N, все значения после достижения указанного числа отбрасываются.

При выборе значения N рекомендуется не превышать порядка сотен или тысяч, чтобы не возникало проблем с ограниченной доступной памятью на кластерах YDB.

Если для задачи обязательно нужны все значения, и их количество может измеряться десятками тысяч и больше, то вместо данных агрегационных функций следует использовать `JOIN` исходной таблицы с подзапросом, где по ней же сделан `GROUP BY + MIN/MAX` на интересующих вас колонках.

Важно

Если второй аргумент всегда `NULL`, то результатом агрегации будет `NULL`.

При использовании фабрики агрегационной функции в качестве первого аргумента AGGREGATE_BY передается `Tuple` из значения и ключа.

Примеры

```
SELECT
  MIN_BY(value, LENGTH(value)),
  MAX_BY(value, key, 100)
FROM my_table;
```

```
$min_by_factory = AggregationFactory("MIN_BY");
$max_by_factory = AggregationFactory("MAX_BY", 100);

SELECT
    AGGREGATE_BY(AsTuple(value, LENGTH(value)), $min_by_factory),
    AGGREGATE_BY(AsTuple(value, key), $max_by_factory)
FROM my_table;
```

TOP и BOTTOM

Сигнатура

```
TOP(T?, limit:Uint32)->List<T>
TOP(T, limit:Uint32)->List<T>
BOTTOM(T?, limit:Uint32)->List<T>
BOTTOM(T, limit:Uint32)->List<T>
```

Вернуть список максимальных/минимальных значений выражения. Первый аргумент - выражение, второй - ограничение на количество элементов.

Примеры

```
SELECT
    TOP(key, 3),
    BOTTOM(value, 3)
FROM my_table;
```

```
$top_factory = AggregationFactory("TOP", 3);
$bottom_factory = AggregationFactory("BOTTOM", 3);

SELECT
    AGGREGATE_BY(key, $top_factory),
```

```
    AGGREGATE_BY(value, $bottom_factory)
FROM my_table;
```

TOP_BY и BOTTOM_BY

Сигнатура

```
TOP_BY(T1, T2, limit:Uint32)->List<T1>
BOTTOM_BY(T1, T2, limit:Uint32)->List<T1>
```

Вернуть список значений первого аргумента для строк с максимальными/минимальными значениями второго аргумента. Третий аргумент - ограничение на количество элементов в списке.

При использовании фабрики агрегационной функции в качестве первого аргумента AGGREGATE_BY передается `Tuple` из значения и ключа. Ограничение на количество элементов в этом случае передаётся вторым аргументом при создании фабрики.

Примеры

```
SELECT
    TOP_BY(value, LENGTH(value), 3),
    BOTTOM_BY(value, key, 3)
FROM my_table;
```

```
$top_by_factory = AggregationFactory("TOP_BY", 3);
$bottom_by_factory = AggregationFactory("BOTTOM_BY", 3);

SELECT
    AGGREGATE_BY(AsTuple(value, LENGTH(value)), $top_by_factory),
    AGGREGATE_BY(AsTuple(value, key), $bottom_by_factory)
FROM my_table;
```

TOPFREQ и MODE

Сигнатура

```
TOPFREQ(T [, num:Uint32 [, bufSize:Uint32]])->List<Struct<Frequency:Uint64, Value:T>>
MODE(T [, num:Uint32 [, bufSize:Uint32]])->List<Struct<Frequency:Uint64, Value:T>>
```

Получение **приближенного** списка самых часто встречающихся значений колонки с оценкой их числа. Возвращают список структур с двумя полями:

- `Value` — найденное часто встречающееся значение;
- `Frequency` — оценка числа упоминаний в таблице.

Обязательный аргумент: само значение.

Опциональные аргументы:

1. Для `TOPFREQ` — желаемое число элементов в результате. `MODE` является алиасом к `TOPFREQ` с 1 в этом аргументе. У `TOPFREQ` по умолчанию тоже 1.
2. Число элементов в используемом буфере, что позволяет разменивать потребление памяти на точность. По умолчанию 100.

Примеры

```
SELECT
    MODE(my_column),
    TOPFREQ(my_column, 5, 1000)
FROM my_table;
```

STDDEV и VARIANCE

Сигнатура

```
STDDEV(Double?)->Double?
STDDEV_POPULATION(Double?)->Double?
POPULATION_STDDEV(Double?)->Double?
STDDEV_SAMPLE(Double?)->Double?
STDDEVSAMP(Double?)->Double?

VARIANCE(Double?)->Double?
VARIANCE_POPULATION(Double?)->Double?
POPULATION_VARIANCE(Double?)->Double?
VARPOP(Double?)->Double?
VARIANCE_SAMPLE(Double?)->Double?
```

Стандартное отклонение и дисперсия по колонке. Используется однопроходной параллельный алгоритм, результат которого может отличаться от полученного более распространенными методами, требующими двух проходов по данным.

По умолчанию вычисляются выборочная дисперсия и стандартное отклонение. Доступны несколько способов записи:

- с суффиксом/префиксом `POPULATION`, например: `VARIANCE_POPULATION`, `POPULATION_VARIANCE` — вычисляет дисперсию/стандартное отклонение для генеральной совокупности;

- с суффиксом `SAMPLE` или без суффикса, например `VARIANCE_SAMPLE`, `SAMPLE_VARIANCE`, `VARIANCE` — вычисляет выборочную дисперсию и стандартное отклонение.

Также определено несколько сокращенных алиасов, например `VARPOP` или `STDDEVSAMP`.

Если все переданные значения — `NULL`, возвращает `NULL`.

Примеры

```
SELECT
  STDDEV(numeric_column),
  VARIANCE(numeric_column)
FROM my_table;
```

CORRELATION и COVARIANCE

Сигнатура

```
CORRELATION(Double?, Double?)->Double?
COVARIANCE(Double?, Double?)->Double?
COVARIANCE_SAMPLE(Double?, Double?)->Double?
COVARIANCE_POPULATION(Double?, Double?)->Double?
```

Корреляция и ковариация двух колонок.

Также доступны сокращенные версии `CORR` или `COVAR`, а для ковариации - версии с суффиксом `SAMPLE` / `POPULATION` по аналогии с описанной выше VARIANCE.

В отличие от большинства других агрегатных функций не пропускают `NULL`, а считают его за 0.

При использовании фабрики агрегационной функции в качестве первого аргумента AGGREGATE_BY передается `Tuple` из двух значений.

Примеры

```
SELECT
  CORRELATION(numeric_column, another_numeric_column),
  COVARIANCE(numeric_column, another_numeric_column)
FROM my_table;
```

```
$corr_factory = AggregationFactory("CORRELATION");

SELECT
    AGGREGATE_BY(AsTuple(numeric_column, another_numeric_column), $corr_factory)
FROM my_table;
```

PERCENTILE и MEDIAN

Сигнатура

```
PERCENTILE(T, Double)->T
PERCENTILE(T, Tuple<Double, ...>)->Tuple<T, ...>
PERCENTILE(T, Struct<name1:Double, ...>)->Struct<name1:T, ...>
PERCENTILE(T, List<Double>)->List<T>

MEDIAN(T, [ Double ])->T
MEDIAN(T, [ Tuple<Double, ...> ])->Tuple<T, ...>
MEDIAN(T, [ Struct<name1:Double, ...> ])->Struct<name1:T, ...>
MEDIAN(T, [ List<Double> ])->List<T>
```

Подсчет процентилей по амортизированной версии алгоритма TDigest. `MEDIAN(x)` без второго аргумента — алиас для `PERCENTILE(x, 0.5)`.
`MEDIAN` с двумя аргументами полностью эквивалентен `PERCENTILE`.

В качестве первого аргумента `PERCENTILE` / `MEDIAN` принимает выражение типа `T`. В качестве типа `T` на данный момент поддерживаются типы `Interval` и `Double` (а также типы которые допускают неявное приведение к ним - например целочисленные типы).

В качестве второго аргумента можно использовать либо один `Double` (значение перцентиля), либо сразу несколько значений перцентиля в виде `Tuple` / `Struct` / `List`.

Значения прецентиля должны лежать в диапазоне от 0.0 до 1.0 включительно.

Примеры

```
SELECT
    MEDIAN(numeric_column),
    PERCENTILE(numeric_column, 0.99),
    PERCENTILE(CAST(string_column as Double), (0.01, 0.5, 0.99)),                  -- подсчет сразу трех пер
центилей
    PERCENTILE(numeric_column, AsStruct(0.01 as p01, 0.5 as median, 0.99 as p99)), -- используя структуру, зн
ачениям перцентиля можно дать удобные имена
    PERCENTILE(numeric_column, ListFromRange(0.00, 1.05, 0.05)),                   -- подсчет множества перце
нтилей (от 0.0 до 1.0 включительно с шагом 0.05)
FROM my_table;
```

HISTOGRAM

Сигнатура

```
HISTOGRAM(Double?)->HistogramStruct?
HISTOGRAM(Double?, weight:Double)->HistogramStruct?
HISTOGRAM(Double?, intervals:Uint32)->HistogramStruct?
HISTOGRAM(Double?, weight:Double, intervals:Uint32)->HistogramStruct?
```

В описании сигнатур под HistogramStruct подразумевается результат работы агрегатной функции, который является структурой определенного вида.

Построение примерной гистограммы по числовому выражению с автоматическим выбором корзин.

Вспомогательные функции

Базовые настройки

Ограничение на число корзин можно задать с помощью опционального аргумента, значение по умолчанию — 100. Следует иметь в виду, что дополнительная точность стоит дополнительных вычислительных ресурсов и может негативно сказываться на времени выполнения запроса, а в экстремальных случаях — и на его успешности.

Поддержка весов

Имеется возможность указать «вес» для каждого значения, участвующего в построении гистограммы. Для этого вторым аргументом в агрегатную функцию нужно передать выражение для вычисления веса. По умолчанию всегда используется вес `1.0`. Если используются нестандартные веса, ограничение на число корзин можно задать третьим аргументом.

В случае, если передано два аргумента, смысл второго аргумента определяется по его типу (целочисленный литерал — ограничение на число корзин, в противном случае — вес).

Если нужна точная гистограмма

1. Можно воспользоваться описанными ниже агрегатными функциями с фиксированными сетками корзин: LinearHistogram или LogarithmicHistogram.
2. Можно самостоятельно вычислить номер корзины для каждой строки и сделать по нему GROUP BY.

При использовании фабрики агрегационной функции в качестве первого аргумента AGGREGATE_BY передается `Tuple` из значения и веса.

Примеры

```
SELECT
    HISTOGRAM(numeric_column)
FROM my_table;
```

```
SELECT
    Histogram::Print(
        HISTOGRAM(numeric_column, 10),
        50
    )
FROM my_table;
```

```
$hist_factory = AggregationFactory("HISTOGRAM");

SELECT
    AGGREGATE_BY(AsTuple(numeric_column, 1.0), $hist_factory)
FROM my_table;
```

LinearHistogram, LogarithmicHistogram и LogHistogram

Построение гистограммы по явно указанной фиксированной шкале корзин.

Сигнатура

```
LinearHistogram(Double?)->HistogramStruct?
LinearHistogram(Double? [, binSize:Double [, min:Double [, max:Double]]])->HistogramStruct?

LogarithmicHistogram(Double?)->HistogramStruct?
LogarithmicHistogram(Double? [, logBase:Double [, min:Double [, max:Double]]])->HistogramStruct?
LogHistogram(Double?)->HistogramStruct?
LogHistogram(Double? [, logBase:Double [, min:Double [, max:Double]]])->HistogramStruct?
```

Аргументы:

1. Выражение, по значению которого строится гистограмма. Все последующие — опциональны.
2. Расстояние между корзинами для `LinearHistogram` или основание логарифма для `LogarithmicHistogram` / `LogHistogram` (это алиасы). В обоих случаях значение по умолчанию — 10.
3. Минимальное значение. По умолчанию минус бесконечность.
4. Максимальное значение. По умолчанию плюс бесконечность.

Формат результата полностью аналогичен адаптивным гистограммам, что позволяет использовать тот же набор вспомогательных функций.

Если разброс входных значений неконтролируемо велик, рекомендуется указывать минимальное и максимальное значение для предотвращения потенциальных падений из-за высокого потребления памяти.

Примеры

```
SELECT
    LogarithmicHistogram(numeric_column, 2)
FROM my_table;
```

CDF (cumulative distribution function)

К каждому виду функции Histogram можно приписать суффикс CDF для построения кумулятивной функции распределения. Конструкции

```
SELECT
    Histogram::ToCumulativeDistributionFunction(Histogram::Normalize(<вид_функции>Histogram(numeric_column)))
FROM my_table;
```

и

```
SELECT
    <вид_функции>HistogramCDF(numeric_column)
FROM my_table;
```

полностью эквивалентны.

BOOL_AND, BOOL_OR и BOOL_XOR

Сигнатура

```
BOOL_AND(Bool?)->Bool?
BOOL_OR(Bool?)->Bool?
BOOL_XOR(Bool?)->Bool?
```

Применение соответствующей логической операции (`AND` / `OR` / `XOR`) ко всем значениям булевой колонки или выражения.

В отличие от большинства агрегатных функций, эти функции **не пропускают** `NULL` значение при агрегации и действуют по правилу:

- `true AND null == null`
- `false OR null == null`

Для `BOOL_AND`:

- Для любого количества значений `true` и хотя бы одного `NULL` значения, результатом будет `NULL`.
- В случае хотя бы одного `false` значения, результатом будет `false`, независимо от наличия `NULL`.

Для `BOOL_OR`:

- Для любого количества значений `false` и хотя бы одного `NULL` значения, результатом будет `NULL`.
- В случае хотя бы одного `true` значения, результатом будет `true`, независимо от наличия `NULL`.

Для `BOOL_XOR`:

- В случае хотя бы одного `NULL` значения, результатом будет `NULL`.

Примеры описанного поведения приведены ниже.

Для агрегации с пропуском `NULL`-ов можно использовать функции `MIN` / `MAX` или `BIT_AND` / `BIT_OR` / `BIT_XOR`.

Примеры

```
$data = [
    <|nonNull: true, nonFalse: true, nonTrue: NULL, anyVal: true|>,
    <|nonNull: false, nonFalse: NULL, nonTrue: NULL, anyVal: NULL|>,
    <|nonNull: false, nonFalse: NULL, nonTrue: false, anyVal: false|>,
];

SELECT
    BOOL_AND(nonNull) as nonNullAnd,      -- false
    BOOL_AND(nonFalse) as nonFalseAnd,    -- NULL
    BOOL_AND(nonTrue) as nonTrueAnd,      -- false
    BOOL_AND(anyVal) as anyAnd,           -- false
    BOOL_OR(nonNull) as nonNullOr,        -- true
    BOOL_OR(nonFalse) as nonFalseOr,      -- true
    BOOL_OR(nonTrue) as nonTrueOr,        -- NULL
    BOOL_OR(anyVal) as anyOr,             -- true
    BOOL_XOR(nonNull) as nonNullXor,      -- true
    BOOL_XOR(nonFalse) as nonFalseXor,    -- NULL
    BOOL_XOR(nonTrue) as nonTrueXor,      -- NULL
    BOOL_XOR(anyVal) as anyXor,           -- NULL
FROM AS_TABLE($data);
```

BIT_AND, BIT_OR и BIT_XOR

Применение соответствующей битовой операции ко всем значениям числовой колонки или выражения.

Примеры

```
SELECT
    BIT_XOR(unsigned_numeric_value)
FROM my_table;
```

SessionStart

Без аргументов. Допускается только при наличии SessionWindow в
GROUP BY / PARTITION BY.
Возвращает значение ключевой колонки `SessionWindow`. В случае `SessionWindow` с двумя аргументами – минимальное значение первого аргумента внутри группы/раздела.
В случае расширенного варианта `SessionWindow` – значение второго элемента кортежа, возвращаемого `<calculate_lambda>`, при котором первый элемент кортежа равен `True`.

AGGREGATE_BY и MULTI_AGGREGATE_BY

Важно

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Применение фабрики агрегационной функции ко всем значениям колонки или выражения. Функция `MULTI_AGGREGATE_BY` требует, чтобы в значении колонки или выражения была структура, кортеж или список, и применяет фабрику поэлементно, размещая результат в контейнере той же формы. Если в разных значениях колонки или выражения содержатся списки разной длины, результирующий список будет иметь наименьшую из длин этих списков.

1. Колонка, `DISTINCT` колонка или выражение;
2. Фабрика.

Примеры

```
$count_factory = AggregationFactory("COUNT");

SELECT
    AGGREGATE_BY(DISTINCT column, $count_factory) as uniq_count
FROM my_table;

SELECT
    MULTI_AGGREGATE_BY(nums, AggregationFactory("count")) as count,
    MULTI_AGGREGATE_BY(nums, AggregationFactory("min")) as min,
    MULTI_AGGREGATE_BY(nums, AggregationFactory("max")) as max,
    MULTI_AGGREGATE_BY(nums, AggregationFactory("avg")) as avg,
    MULTI_AGGREGATE_BY(nums, AggregationFactory("percentile", 0.9)) as p90
FROM my_table;
```

Список оконных функций в YQL

Синтаксис вызова оконных функций подробно описан в отдельной статье.

Агрегатные функции

Все агрегатные функции также могут использоваться в роли оконных.
В этом случае на каждой строке оказывается результат агрегации, полученный на множестве строк из рамки окна.

Примеры

```
SELECT
    SUM(int_column) OVER w1 AS running_total,
    SUM(int_column) OVER w2 AS total,
FROM my_table
WINDOW
    w1 AS (ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW),
    w2 AS ();
```

ROW_NUMBER

Номер строки в рамках раздела. Без аргументов.

Сигнатура

```
ROW_NUMBER()->Uint64
```

Примеры

```
SELECT
    ROW_NUMBER() OVER w AS row_num
FROM my_table
WINDOW w AS (ORDER BY key);
```

LAG / LEAD

Доступ к значению из строки раздела, отстающей (`LAG`) или опережающей (`LEAD`) текущую на фиксированное число. В первом аргументе указывается выражение, к которому необходим доступ, а во втором — отступ в строках. Отступ можно не указывать, по умолчанию используется соседняя строка — предыдущая или следующая, соответственно, то есть подразумевается 1. В строках, для которых нет соседей с заданным расстоянием (например `LAG(expr, 3)` в первой и второй строках раздела), возвращается `NULL` .

Сигнатура

```
LEAD(T[,Int32])->T?
LAG(T[,Int32])->T?
```

Примеры

```
SELECT
   int_value - LAG(int_value) OVER w AS int_value_diff
FROM my_table
WINDOW w AS (ORDER BY key);
```

```
SELECT item, odd, LAG(item, 1) OVER w as lag1 FROM (
    SELECT item, item % 2 as odd FROM (
        SELECT AsList(1, 2, 3, 4, 5, 6, 7) as item
    )
    FLATTEN BY item
)
WINDOW w As (
    PARTITION BY odd
    ORDER BY item
);

/* Output:
item  odd  lag1
--------------------
2  0  NULL
4  0  2
6  0  4
1  1  NULL
3  1  1
5  1  3
7  1  5
*/
```

FIRST_VALUE / LAST_VALUE

Доступ к значениям из первой и последней (в порядке `ORDER BY` на окне) строк рамки окна. Единственный аргумент - выражение, к которому необходим доступ.

Опционально перед `OVER` может указываться дополнительный модификатор `IGNORE NULLS` , который меняет поведение функций на первое или последнее **не пустое** (то есть не `NULL`) значение среди строк рамки окна. Антоним этого

модификатора — `RESPECT NULLS` является поведением по умолчанию и может не указываться.

Сигнатура

```
FIRST_VALUE(T)->T?
LAST_VALUE(T)->T?
```

Примеры

```
SELECT
   FIRST_VALUE(my_column) OVER w
FROM my_table
WINDOW w AS (ORDER BY key);
```

```
SELECT
   LAST_VALUE(my_column) IGNORE NULLS OVER w
FROM my_table
WINDOW w AS (ORDER BY key);
```

NTH_VALUE

Доступ к значения из заданной строки (в порядке `ORDER BY` на окне) рамки окна. Аргументы - выражение, к которому необходим доступ и номер строки, начиная с 1.

Опционально перед `OVER` может указываться дополнительный модификатор `IGNORE NULLS`, который приводит к пропуску строк с NULL в значении первого аргумента. Антоним этого модификатора `RESPECT NULLS` является поведением по умолчанию и может не указываться.

Сигнатура

```
NTH_VALUE(T,N)->T?
```

Примеры

```
SELECT
   NTH_VALUE(my_column, 2) OVER w
FROM my_table
WINDOW w AS (ORDER BY key);
```

```
SELECT
   NTH_VALUE(my_column, 3) IGNORE NULLS OVER w
FROM my_table
WINDOW w AS (ORDER BY key);
```

RANK / DENSE_RANK / PERCENT_RANK

Пронумеровать группы соседних строк раздела с одинаковым значением выражения в аргументе. `DENSE_RANK` нумерует группы подряд, а `RANK` — пропускает `(N - 1)` значений, где `N` — число строк в предыдущей группе. `PERCENT_RANK` выдает относительный ранг текущей строки: `(RANK - 1)/(число строк в разделе - 1)`.

При отсутствии аргумента использует порядок, указанный в секции `ORDER BY` определения окна.
Если аргумент отсутствует и `ORDER BY` не указан, то все строки считаются равными друг другу.

Примечание

Возможность передавать аргумент в `RANK` / `DENSE_RANK` / `PERCENT_RANK` является нестандартным расширением YQL.

Сигнатура

```
RANK([T])->Uint64
DENSE_RANK([T])->Uint64
PERCENT_RANK([T])->Double
```

Примеры

```
SELECT
   RANK(my_column) OVER w
FROM my_table
WINDOW w AS (ORDER BY key);
```

```
SELECT
   DENSE_RANK() OVER w
FROM my_table
WINDOW w AS (ORDER BY my_column);
```

```
SELECT
   PERCENT_RANK() OVER w
FROM my_table
WINDOW w AS (ORDER BY my_column);
```

NTILE

Распределяет строки упорядоченного раздела в заданное количество групп. Группы нумеруются, начиная с единицы. Для каждой строки функция NTILE возвращает номер группы,которой принадлежит строка.

Сигнатура

```
NTILE(Uint64)->Uint64
```

Примеры

```
SELECT
    NTILE(10) OVER w AS group_num
FROM my_table
WINDOW w AS (ORDER BY key);
```

CUME_DIST

Возвращает относительную позицию (> 0 и <= 1) строки в рамках раздела. Без аргументов.

Сигнатура

```
CUME_DIST()->Double
```

Примеры

```
SELECT
    CUME_DIST() OVER w AS dist
FROM my_table
WINDOW w AS (ORDER BY key);
```

SessionState()

Нестандартная оконная функция `SessionState()` (без аргументов) позволяет получить состояние расчета сессий из SessionWindow для текущей строки.

Допускается только при наличии `SessionWindow()` в секции `PARTITION BY` определения окна.

Функции для работы со списками

ListCreate

Сконструировать пустой список. В единственном аргументе указывается строка с описанием типа данных ячейки списка, либо сам тип, полученный с помощью предназначенных для этого функций. Списков с неизвестным типом ячейки в YQL не бывает.

Документация по формату описания типа.

Примеры

```
SELECT ListCreate(Tuple<String,Double?>);
```

```
SELECT ListCreate(OptionalType(DataType("String")));
```

Сигнатура

```
ListCreate(T)->List<T>
```

AsList и AsListStrict

Сконструировать список из одного или более аргументов. Типы аргументов должны быть совместимы в случае `AsList` и строго совпадать в случае `AsListStrict`.

Примеры

```
SELECT AsList(1, 2, 3, 4, 5);
```

Сигнатура

```
AsList(T..)->List<T>
```

ListLength

Количество элементов в списке.

Сигнатура

```
ListLength(List<T>)->Uint64
ListLength(List<T>?)->Uint64?
```

ListHasItems

Проверка того, что список содержит хотя бы один элемент.

Сигнатура

```
ListHasItems(List<T>)->Bool
ListHasItems(List<T>?)->Bool?
```

ListCollect

Преобразовать ленивый список (строится, например, функциями ListFilter, ListMap, ListFlatMap) в энергичный. В отличие от ленивого списка, в котором каждый повторный проход заново вычисляет его содержимое, в энергичном списке содержимое списка строится сразу ценой большего потребления памяти.

Сигнатура

```
ListCollect(LazyList<T>)->List<T>
ListCollect(LazyList<T>?)->List<T>?
```

ListSort, ListSortAsc и ListSortDesc

Отсортировать список. По умолчанию выполняется сортировка по возрастанию (`ListSort` — алиас к `ListSortAsc`).

Аргументы:

1. Список;
2. Опциональное выражение для получения ключа сортировки из элемента списка (по умолчанию сам элемент).

Примеры

```
$list = AsList(
    AsTuple("x", 3),
    AsTuple("xx", 1),
    AsTuple("a", 2)
);

SELECT ListSort($list, ($x) -> {
    RETURN $x.1;
});
```

Примечание

В примере использовалась лямбда функция.

Сигнатура

```
ListSort(List<T>)->List<T>
ListSort(List<T>?)->List<T>?

ListSort(List<T>, (T)->U)->List<T>
ListSort(List<T>?, (T)->U)->List<T>?
```

ListExtend и ListExtendStrict

Последовательно соединить списки (конкатенация списков). В качестве аргументов могут быть списки, опциональные списки и `NULL`.
Типы элементов списков должны быть совместимы в случае `ListExtend` и строго совпадать в случае `ListExtendStrict`.
Если хотя бы один из списков является опциональным, то таким же является и результат.
Если хотя бы один аргумент является `NULL`, то тип результата - `NULL`.

```
$l1 = AsList("a", "b");
$l2 = AsList("b", "c");
$l3 = AsList("d", "e");

SELECT ListExtend($l1, $l2, $l3);  -- ["a","b","b","c","d","e"]
```

Сигнатура

```
ListExtend(List<T>..)->List<T>
ListExtend(List<T>?..)->List<T>?
```

ListUnionAll

Последовательно соединить списки структур (конкатенация списков). В выходном списке структур будет присутствовать поле, если оно есть хотя бы в одном исходном списке, при этом в случае отсутствия такого поля в каком-либо списке оно дополняется как NULL. В случае, когда поле присутствует в двух и более списках, поле в результате приводит в общий тип.

Если хотя бы один из списков является опциональным, то таким же является и результат.

Примеры

```
$l1 = AsList(
    <|value:1|>,
    <|value:2|>
);
$l2 = AsList(
    <|key:"a"|>,
    <|key:"b"|>
);
SELECT ListUnionAll($l1, $l2);  -- result: [("value":1),("value":2),("key":"a"),("key":"b")]
                                -- schema: List<Struct<key : String?, value : Int32?>>
```

Сигнатура

```
ListUnionAll(List<Struct<..>>, List<Struct<..>>..)->List<Struct<..>>
ListUnionAll(List<Struct<..>>?, List<Struct<..>>?..)->List<Struct<..>>?
```

ListZip и ListZipAll

По входящим спискам построить список пар, содержащих соответствующие по индексу элементы списков (`List<Tuple<first_list_element_type,second_list_element_type>>`).

Длина возвращаемого списка определяется самым коротким списком для ListZip и самым длинным — для ListZipAll.
Когда более короткий список исчерпан, в качестве пары к элементам более длинного списка подставляется пустое значение (`NULL`) соответствующего optional типа.

Примеры

```
$l1 = AsList("a", "b");
$l2 = AsList(1, 2, 3);

SELECT ListZip($l1, $l2);  -- [("a",1),("b",2)]
SELECT ListZipAll($l1, $l2);  -- [("a",1),("b",2),(null,3)]
```

Сигнатура

```
ListZip(List<T1>, List<T2>)->List<Tuple<T1, T2>>
ListZip(List<T1>?, List<T2>?)->List<Tuple<T1, T2>>?

ListZipAll(List<T1>, List<T2>)->List<Tuple<T1?, T2?>>
ListZipAll(List<T1>?, List<T2>?)->List<Tuple<T1?, T2?>>?
```

ListEnumerate

Построить список пар (Tuple), содержащих номер элемента и сам элемент (`List<Tuple<Uint64,list_element_type>>`).

Сигнатура

```
ListEnumerate(List<T>)->List<Tuple<Uint64, T>>
ListEnumerate(List<T>?)->List<Tuple<Uint64, T>>?
```

ListReverse

Развернуть список.

Сигнатура

```
ListReverse(List<T>)->List<T>
ListReverse(List<T>?)->List<T>?
```

ListSkip

Возвращает копию списка с пропущенным указанным числом первых элементов.

Первый аргумент — исходный список, второй — сколько элементов пропустить.

Примеры

```
$l1 = AsList(1, 2, 3, 4, 5);

SELECT ListSkip($l1, 2);  -- [3,4,5]
```

Сигнатура

```
ListSkip(List<T>, Uint64)->List<T>
ListSkip(List<T>?, Uint64)->List<T>?
```

ListTake

Возвращает копию списка, состоящую из ограниченного числа элементов второго списка.

Первый аргумент — исходный список, второй — не больше скольких элементов с начала оставить.

Примеры

```
$l1 = AsList(1, 2, 3, 4, 5);

SELECT ListTake($l1, 2);  -- [1,2]
```

Сигнатура

```
ListTake(List<T>, Uint64)->List<T>
ListTake(List<T>?, Uint64)->List<T>?
```

ListSample и ListSampleN

Возвращает выборку без повторений из элементов списка.

- `ListSample` выбирает каждый элемент независимо с заданной вероятностью.
- `ListSampleN` выбирает фиксированное количество элементов (если длина списка меньше размера выборки, то вернется исходный список).

Если вероятность/размер выборки является NULL, то вернется исходный список.

Дополнительный аргумент используется для управления случайностью, подробнее см. документацию к `Random`.

Примеры

```
$list = AsList(1, 2, 3, 4, 5);

SELECT ListSample($list, 0.5);  -- [1, 2, 5]
SELECT ListSampleN($list, 2);  -- [4, 2]
```

Сигнатура

```
ListSample(List<T>, Double?[, U])->List<T>
ListSample(List<T>?, Double?[, U])->List<T>?

ListSampleN(List<T>, Uint64?[, U])->List<T>
ListSampleN(List<T>?, Uint64?[, U])->List<T>?
```

ListShuffle

Возвращает копию списка с элементами, перестановленными в случайном порядке. Дополнительный аргумент используется для управления случайностью, подробнее см. документацию к `Random`.

Примеры

```
$list = AsList(1, 2, 3, 4, 5);

SELECT ListShuffle($list);  -- [1, 3, 5, 2, 4]
```

Сигнатура

```
ListShuffle(List<T>[, U])->List<T>
ListShuffle(List<T>?[, U])->List<T>?
```

ListIndexOf

Ищет элемент с указанным значением в списке и при первом обнаружении возвращает его индекс. Отсчет индексов начинается с 0, а в случае отсутствия элемента возвращается `NULL`.

Примеры

```
$l1 = AsList(1, 2, 3, 4, 5);

SELECT ListIndexOf($l1, 2);  -- 1
```

Сигнатура

```
ListIndexOf(List<T>, T)->Uint64?
ListIndexOf(List<T>?, T)->Uint64?
```

ListMap, ListFlatMap и ListFilter

Применяют к каждому элементу списка указанную в качестве второго аргумента функцию. Различаются возвращаемым результатом:

- `ListMap` — возвращает список с результатами;
- `ListFlatMap` — возвращает список с результатами, объединяя и разворачивая первый уровень результатов (списков или опциональных значений) по каждому элементу;
- `ListFilter` — оставляет только те элементы, для которых функция вернула `true`.

Примечание

В `ListFlatMap` использование опциональных значений в результатах функции является устаревшим, вместо этого следует использовать комбинацию `ListNotNull` и `ListMap`.

Аргументы:

1. Исходный список;
2. Функции для обработки элементов, например:
   - Лямбда функция;
   - `Module::Function` - C++ UDF;

Если исходный список является опциональным, то таким же является и выходной список.

Примеры

```
$list = AsList("a", "b", "c");

$filter = ($x) -> {
    RETURN $x == "b";
};

SELECT ListFilter($list, $filter);  -- ["b"]
```

```
$list = AsList(1,2,3,4);
$callable = Python::test(Callable<(Int64)->Bool>, "def test(i): return i % 2");
SELECT ListFilter($list, $callable);  -- [1,3]
```

Сигнатура

```
ListMap(List<T>, (T)->U)->List<U>
ListMap(List<T>?, (T)->U)->List<U>?

ListFlatMap(List<T>, (T)->List<U>)->List<U>
ListFlatMap(List<T>?, (T)->List<U>)->List<U>?
ListFlatMap(List<T>, (T)->U?)->List<U>
ListFlatMap(List<T>?, (T)->U?)->List<U>?

ListFilter(List<T>, (T)->Bool)->List<T>
ListFilter(List<T>?, (T)->Bool)->List<T>?
```

ListNotNull

Применяет трансформацию исходного списка, пропуская пустые опциональные элементы, и усиливает тип элемента до неопционального. Для списка с неопциональными элементами возвращает исходный список без изменений.

Если исходный список является опциональным, то таким же является и выходной список.

Примеры

```
SELECT ListNotNull([1,2]),   -- [1,2]
    ListNotNull([3,null,4]); -- [3,4]
```

Сигнатура

```
ListNotNull(List<T?>)->List<T>
ListNotNull(List<T?>?)->List<T>?
```

ListFlatten

Разворачивает список списков в плоский список с сохранением порядка элементов. В качестве элемента списка верхнего уровня поддерживается опциональный список, который интерпретируется как пустой в случае `NULL`.

Если исходный список является опциональным, то таким же является и выходной список.

Примеры

```
SELECT ListFlatten([[1,2],[3,4]]),   -- [1,2,3,4]
    ListFlatten([null,[3,4],[5,6]]); -- [3,4,5,6]
```

Сигнатура

```
ListFlatten(List<List<T>?>)->List<T>
ListFlatten(List<List<T>?>?)->List<T>?
```

ListUniq и ListUniqStable

Возвращает копию списка, в котором оставлен только уникальный набор элементов. В случае ListUniq порядок элементов результирующего набора не определен, в случае ListUniqStable элементы находятся в порядке вхождения в исходный список.

Примеры

```
SELECT ListUniq([1, 2, 3, 2, 4, 5, 1]) -- [5, 4, 2, 1, 3]
SELECT ListUniqStable([1, 2, 3, 2, 4, 5, 1]) -- [1, 2, 3, 4, 5]
SELECT ListUniqStable([1, 2, null, 7, 2, 8, null]) -- [1, 2, null, 7, 8]
```

Сигнатура

```
ListUniq(List<T>)->List<T>
ListUniq(List<T>?)->List<T>?

ListUniqStable(List<T>)->List<T>
ListUniqStable(List<T>?)->List<T>?
```

ListAny и ListAll

Для списка булевых значений возвращает `true`, если:

- `ListAny` — хотя бы один элемент равен `true`;
- `ListAll` — все элементы равны `true`.

В противном случае возвращает false.

Сигнатура

```
ListAny(List<Bool>)->Bool
ListAny(List<Bool>?)->Bool?
ListAll(List<Bool>)->Bool
ListAll(List<Bool>?)->Bool?
```

ListHas

Содержит ли список указанный элемент. При этом `NULL` значения считаются равными друг другу, а при `NULL` входном списке результат всегда `false`.

Примеры

```
$l1 = AsList(1, 2, 3, 4, 5);

SELECT ListHas($l1, 2);  -- true
SELECT ListHas($l1, 6);  -- false
```

Сигнатура

```
ListHas(List<T>, U)->Bool
ListHas(List<T>?, U)->Bool
```

ListHead, ListLast

Возвращают первый и последний элемент списка.

Сигнатура

```
ListHead(List<T>)->T?
ListHead(List<T>?)->T?
ListLast(List<T>)->T?
ListLast(List<T>?)->T?
```

ListMin, ListMax, ListSum и ListAvg

Применяет соответствующую агрегатную функцию ко всем элементам списка числовых значений.

Сигнатура

```
ListMin(List<T>)->T?
ListMin(List<T>?)->T?
```

ListFold, ListFold1

Свёртка списка.

Аргументы:

1. Список
2. Начальное состояние U для ListFold, initLambda(item:T)->U для ListFold1
3. updateLambda(item:T, state:U)->U

Возвращаемый тип:
U для ListFold, опциональный U для ListFold1.

Примеры

```
$l = [1, 4, 7, 2];
$y = ($x, $y) -> { RETURN $x + $y; };
$z = ($x) -> { RETURN 4 * $x; };

SELECT
    ListFold($l, 6, $y) AS fold,                        -- 20
    ListFold([], 3, $y) AS fold_empty,                  -- 3
    ListFold1($l, $z, $y) AS fold1,                     -- 17
    ListFold1([], $z, $y) AS fold1_empty;               -- Null
```

Сигнатура

```
ListFold(List<T>, U, (T, U)->U)->U
ListFold(List<T>?, U, (T, U)->U)->U?

ListFold1(List<T>, (T)->U, (T, U)->U)->U?
ListFold1(List<T>?, (T)->U, (T, U)->U)->U?
```

ListFoldMap, ListFold1Map

Преобразует каждый элемент i в списке путём вызова handler(i, state).

Аргументы:

1. Список
2. Начальное состояние S для ListFoldMap, initLambda(item:T)->кортеж (U S) для ListFold1Map
3. handler(item:T, state:S)->кортеж (U S)

Возвращаемый тип:
Список элементов U.

Примеры

```
$l = [1, 4, 7, 2];
$x = ($i, $s) -> { RETURN ($i * $s, $i + $s); };
$t = ($i) -> { RETURN ($i + 1, $i + 2); };

SELECT
    ListFoldMap([], 1, $x),               -- []
    ListFoldMap($l, 1, $x),               -- [1, 8, 42, 26]
    ListFold1Map([], $t, $x),             -- []
    ListFold1Map($l, $t, $x);             -- [2, 12, 49, 28]
```

Сигнатура

```
ListFoldMap(List<T>, S, (T, S)->Tuple<U,S>)->List<U>
ListFoldMap(List<T>?, S, (T, S)->Tuple<U,S>)->List<U>?

ListFold1Map(List<T>, (T)->Tuple<U,S>, (T, S)->Tuple<U,S>)->List<U>
ListFold1Map(List<T>?, (T)->Tuple<U,S>, (T, S)->Tuple<U,S>)->List<U>?
```

ListFromRange

Генерация последовательности чисел или дат с указанным шагом. Аналог `xrange` в Python 2, но дополнительно с поддержкой дат и чисел с плавающей точкой.

Аргументы:

1. Начало
2. Конец
3. Шаг. Опционально, по умолчанию 1 для числовых последовательностей, 1 день для `Date` / `TzDate`, 1 секунда для `Datetime` / `TzDatetime` и 1 микросекунда для `Timestamp` / `TzTimestamp` / `Interval`

Особенности:

- Конец не включительный, т.е. `ListFromRange(1,3) == AsList(1,2)`.
- Тип элементов результатов выбирается как наиболее широкий из типов аргументов, например результатом `ListFromRange(1, 2, 0.5)` получится список `Double`.
- Если начало и конец имеют один из типов дат, то шаг должен иметь тип `Interval`.
- Список является «ленивым», но при неправильном использовании всё равно может привести к потреблению большого объема оперативной памяти.
- Если шаг положительный и конец меньше или равен началу, то список будет пустой.
- Если шаг отрицательный и конец больше или равен началу, то список будет пустой.
- Если шаг не положительный и не отрицательный (0 или NaN), то список будет пустой.
- Если один из параметров опциональный, то результат будет опциональный список.
- Если один из параметров равен `NULL`, то результат будет `NULL`.

Примеры

```
SELECT
    ListFromRange(-2, 2), -- [-2, -1, 0, 1]
    ListFromRange(2, 1, -0.5); -- [2.0, 1.5]
```

```
SELECT ListFromRange(Datetime("2022-05-23T15:30:00Z"), Datetime("2022-05-30T15:30:00Z"), DateTime::IntervalFr
omDays(1));
```

Сигнатура

```
ListFromRange(T{Flags:AutoMap}, T{Flags:AutoMap}, T?)->LazyList<T> -- T — числовой тип
ListFromRange(T{Flags:AutoMap}, T{Flags:AutoMap}, I?)->LazyList<T> -- T — тип, представляющий дату/время, I —
интервал
```

ListReplicate

Создает список из нескольких копий указанного значения.

Обязательные аргументы:

1. Значение;
2. Число копий.

Примеры

```
SELECT ListReplicate(true, 3); -- [true, true, true]
```

Сигнатура

```
ListReplicate(T, Uint64)->List<T>
```

ListConcat

Объединяет список строк в одну строку.
Вторым параметром можно задать разделитель.

Примеры

```
$l1 = AsList("h", "e", "l", "l", "o");

SELECT ListConcat($l1);  -- "hello"
SELECT ListConcat($l1, " ");  -- "h e l l o"
```

Сигнатура

```
ListConcat(List<String>)->String?
ListConcat(List<String>?)->String?

ListConcat(List<String>, String)->String?
ListConcat(List<String>?, String)->String?
```

ListExtract

По списку структур возвращает список содержащихся в них полей с указанным именем.

Примеры

```
$l = AsList(
    <|key:"a", value:1|>,
    <|key:"b", value:2|>
);
SELECT ListExtract($l, "key");  -- ["a", "b"]
```

Сигнатура

```
ListExtract(List<Struct<..>>, String)->List<T>
ListExtract(List<Struct<..>>?, String)->List<T>?
```

ListTakeWhile, ListSkipWhile

`ListTakeWhile` выдает список от начала, пока предикат истинный, далее список заканчивается.

`ListSkipWhile` пропускает отрезок списка от начала, пока предикат истинный, далее выдает остаток список не обращая внимания на предикат.
`ListTakeWhileInclusive` выдает список от начала, пока предикат истинный, далее список заканчивается, но также включает элемент, на котором сработал останавливающий предикат.
`ListSkipWhileInclusive` пропускает отрезок списка от начала, пока предикат истинный, далее выдает остаток список не обращая внимания на предикат, но не включая элемент, на котором сработал предикат, а начинает со следующего за ним.

Обязательные аргументы:

1. Список;
2. Предикат.

Если входной список является опциональным, то таким же является и результат.

Примеры

```
$data = AsList(1, 2, 5, 1, 2, 7);

SELECT
    ListTakeWhile($data, ($x) -> {return $x <= 3}), -- [1, 2]
    ListSkipWhile($data, ($x) -> {return $x <= 3}), -- [5, 1, 2, 7]
    ListTakeWhileInclusive($data, ($x) -> {return $x <= 3}), -- [1, 2, 5]
    ListSkipWhileInclusive($data, ($x) -> {return $x <= 3}); -- [1, 2, 7]
```

Сигнатура

```
ListTakeWhile(List<T>, (T)->Bool)->List<T>
ListTakeWhile(List<T>?, (T)->Bool)->List<T>?
```

ListAggregate

Применить фабрику агрегационных функций для переданного списка.
Если переданный список является пустым, то результат агрегации будет такой же, как для пустой таблицы: 0 для функции `COUNT` и `NULL` для других функций.
Если переданный список является опциональным и равен `NULL`, то в результате также будет `NULL`.

Аргументы:

1. Список;
2. Фабрика агрегационных функций.

Примеры

```
SELECT ListAggregate(AsList(1, 2, 3), AggregationFactory("Sum")); -- 6
```

Сигнатура

```
ListAggregate(List<T>, AggregationFactory)->T
ListAggregate(List<T>?, AggregationFactory)->T?
```

ToDict и ToMultiDict

Преобразуют список из кортежей с парами ключ-значение в словарь. В случае конфликтов по ключам во входном списке `ToDict` оставляет первое значение, а `ToMultiDict` — собирает из всех значений список.

Таким образом:

- `ToDict` из `List<Tuple<K, V>>` делает `Dict<K, V>`
- `ToMultiDict` из `List<Tuple<K, V>>` делает `Dict<K, List<V>>`

Также поддерживаются опциональные списки, что приводит к опциональному словарю в результате.

Примеры

```
$l = AsList(("a",1), ("b", 2), ("a", 3));
SELECT ToDict($l);  -- {"a": 1,"b": 2}
```

Сигнатура

```
ToDict(List<Tuple<K,V>>)->Dict<K,V>
ToDict(List<Tuple<K,V>>?)->Dict<K,V>?
```

ToSet

Преобразует список в словарь, в котором ключи являются уникальными элементами этого списка, а значения отсутствуют и имеют тип `Void`. Для списка `List<T>` тип результата будет `Dict<T, Void>`.
Также поддерживается опциональный список, что приводит к опциональному словарю в результате.

Обратная функция - получить список ключей словаря DictKeys.

Примеры

```
$l = AsList(1,1,2,2,3);
SELECT ToSet($l);  -- {1,2,3}
```

Сигнатура

```
ToSet(List<T>)->Set<T>
ToSet(List<T>?)->Set<T>?
```

ListFromTuple

Строит список из кортежа, в котором типы элементов совместимы друг с другом. Для опционального кортежа на выходе получается опциональный список. Для NULL аргумента - NULL. Для пустого кортежа - EmptyList.

Примеры

```
$t = (1,2,3);
SELECT ListFromTuple($t);  -- [1,2,3]
```

Сигнатура

```
ListFromTuple(Null)->Null
ListFromTuple(Tuple<>)->EmptyList
ListFromTuple(Tuple<T1,T2,...>)->List<T>
ListFromTuple(Tuple<T1,T2,...>?)->List<T>?
```

ListToTuple

Строит кортеж из списка и явно указанной ширины кортежа. Все элементы кортежа будут иметь тот же тип, что и тип элемента списка. Если длина списка не соотвествует указанной ширине кортежа, будет возвращена ошибка. Для опционального списка на выходе получается опциональный кортеж. Для NULL аргумента - NULL.

Примеры

```
$l = [1,2,3];
SELECT ListToTuple($l, 3);  -- (1,2,3)
```

Сигнатура

```
ListToTuple(Null,N)->Null
ListToTuple(EmptyList,N)->()) -- N должен быть 0
ListToTuple(List<T>, N)->Tuple<T,T,...T> -- ширина кортежа N
ListToTuple(List<T>?, N)->Tuple<T,T,...T>? -- ширина кортежа N
```

ListTop, ListTopAsc, ListTopDesc, ListTopSort, ListTopSortAsc и ListTopSortDesc

Выбрать топ значениий из списка. `ListTopSort*` — дополнительно отсортировать список возвращенных значений. По умолчанию выбираются наименьшие значения (функции без суффикса — алиас к функциям `*Asc`; `*Desc` — выбор наибольших значений).

`ListTopSort` эффективнее, чем последовательные `ListTop` и `ListSort`, так как `ListTop` может уже частично отсортировать список для поиска нужных значений. Однако, `ListTop` эффективнее `ListTopSort`, если сортировка не нужна.

Аргументы:

1. Список;

2. Размер выборки;
3. Опциональное выражение для получения ключа сортировки из элемента списка (по умолчанию сам элемент).

Сигнатура

```
ListTop(List<T>{Flags:AutoMap}, N)->List<T>
ListTop(List<T>{Flags:AutoMap}, N, (T)->U)->List<T>
```

Сигнатуры остальных функций совпадают с `ListTop`.

Функции для работы со словарями

DictCreate

Сигнатура

```
DictCreate(K,V)->Dict<K,V>
```

Сконструировать пустой словарь. Передается два аргумента — для ключа и значения, в каждом из которых указывается строка с описанием типа данных, либо сам тип, полученный с помощью предназначенных для этого функций. Словарей с неизвестным типом ключа или значения в YQL не бывает.

В качестве типа ключа могут быть заданы:

- примитивный тип данных (кроме `Yson` и `Json`),
- примитивный тип данных (кроме `Yson` и `Json`) с признаком опциональности,
- кортеж длины не менее два из типов, перечисленных выше.

Документация по формату описания типа.

Примеры

```
SELECT DictCreate(String, Tuple<String,Double?>);
```

```
SELECT DictCreate(Tuple<Int32?,String>, OptionalType(DataType("String")));
```

```
SELECT DictCreate(ParseType("Tuple<Int32?,String>"), ParseType("Tuple<String,Double?>"));
```

SetCreate

Сигнатура

```
SetCreate(T)->Set<T>
```

Сконструировать пустое множество. Передается аргумент - тип ключа, возможно, полученный с помощью предназначенных для этого функций. Множеств с неизвестным типом ключа в YQL не бывает. Ограничения на тип ключа такие же как и на тип ключа для словаря. Следует иметь ввиду, что множество это словарь с типом значения `Void` и множество также можно создать и с помощью функции `DictCreate`. Отсюда также следует, что все функции, которые принимают на вход `Dict<K,V>` могут также принимать `Set<K>`.

Документация по формату описания типа.

Примеры

```
SELECT SetCreate(String);
```

```
SELECT SetCreate(Tuple<Int32?,String>);
```

DictLength

Сигнатура

```
DictLength(Dict<K,V>)->Uint64
DictLength(Dict<K,V>?)->Uint64?
```

Количество элементов в словаре.

Примеры

```
SELECT DictLength(AsDict(AsTuple(1, AsList("foo", "bar"))));
```

DictHasItems

Сигнатура

```
DictHasItems(Dict<K,V>)->Bool
DictHasItems(Dict<K,V>?)->Bool?
```

Проверка того, что словарь содержит хотя бы один элемент.

Примеры

```
SELECT DictHasItems(AsDict(AsTuple(1, AsList("foo", "bar")))) FROM my_table;
```

DictItems

Сигнатура

```
DictItems(Dict<K,V>)->List<Tuple<K,V>>
DictItems(Dict<K,V>?)->List<Tuple<K,V>>?
```

Получение содержимого словаря в виде списка кортежей с парами ключ-значение (`List<Tuple<key_type,value_type>>`).

Примеры

```
SELECT DictItems(AsDict(AsTuple(1, AsList("foo", "bar"))));
-- [ ( 1, [ "foo", "bar" ] ) ]
```

DictKeys

Сигнатура

```
DictKeys(Dict<K,V>)->List<K>
DictKeys(Dict<K,V>?)->List<K>?
```

Получение списка ключей словаря.

Примеры

```
SELECT DictKeys(AsDict(AsTuple(1, AsList("foo", "bar"))));
-- [ 1 ]
```

DictPayloads

Сигнатура

```
DictPayloads(Dict<K,V>)->List<V>
DictPayloads(Dict<K,V>?)->List<V>?
```

Получение списка значений словаря.

Примеры

```
SELECT DictPayloads(AsDict(AsTuple(1, AsList("foo", "bar"))));
-- [ [ "foo", "bar" ] ]
```

DictLookup

Сигнатура

```
DictLookup(Dict<K,V>, K)->V?
DictLookup(Dict<K,V>?, K)->V?
DictLookup(Dict<K,V>, K?)->V?
DictLookup(Dict<K,V>?, K?)->V?
```

Получение элемента словаря по ключу.

Примеры

```
SELECT DictLookup(AsDict(
    AsTuple(1, AsList("foo", "bar")),
    AsTuple(2, AsList("bar", "baz"))
), 1);
-- [ "foo", "bar" ]
```

DictContains

Сигнатура

```
DictContains(Dict<K,V>, K)->Bool
DictContains(Dict<K,V>?, K)->Bool
DictContains(Dict<K,V>, K?)->Bool
DictContains(Dict<K,V>?, K?)->Bool
```

Проверка наличия элемента в словаре по ключу. Возвращает true или false.

Примеры

```
SELECT DictContains(AsDict(
    AsTuple(1, AsList("foo", "bar")),
    AsTuple(2, AsList("bar", "baz"))
), 42);
-- false
```

DictAggregate

Сигнатура

```
DictAggregate(Dict<K,List<V>>, List<V>->T)->Dict<K,T>
DictAggregate(Dict<K,List<V>>?, List<V>->T)->Dict<K,T>?
```

Применить фабрику агрегационных функций для переданного словаря, в котором каждое значение является списком.
Фабрика применяется отдельно внутри каждого ключа.
Если список является пустым, то результат агрегации будет такой же, как для пустой таблицы: 0 для функции `COUNT` и `NULL` для других функций.
Если в переданном словаре список по некоторому ключу является пустым, то такой ключ удаляется из результата.
Если переданный словарь является опциональным и содержит значение `NULL`, то в результате также будет `NULL`.

Аргументы:

1. Словарь;
2. Фабрика агрегационных функций.

Примеры

```
SELECT DictAggregate(AsDict(
    AsTuple(1, AsList("foo", "bar")),
    AsTuple(2, AsList("baz", "qwe"))),
    AggregationFactory("Max"));
-- {1 : "foo", 2 : "qwe" }
```

SetIsDisjoint

Сигнатура

```
SetIsDisjoint(Dict<K,V1>, Dict<K,V2>)->Bool
SetIsDisjoint(Dict<K,V1>?, Dict<K,V2>)->Bool?
SetIsDisjoint(Dict<K,V1>, Dict<K,V2>?)->Bool?
SetIsDisjoint(Dict<K,V1>?, Dict<K,V2>?)->Bool?

SetIsDisjoint(Dict<K,V1>, List<K>)->Bool
SetIsDisjoint(Dict<K,V1>?, List<K>)->Bool?
SetIsDisjoint(Dict<K,V1>, List<K>?)->Bool?
SetIsDisjoint(Dict<K,V1>?, List<K>?)->Bool?
```

Проверка того, что словарь и список или другой словарь не пересекаются по ключам.

Таким образом есть два варианта вызова:

- С аргументами `Dict<K,V1>` и `List<K>`;
- С аргументами `Dict<K,V1>` и `Dict<K,V2>`.

Примеры

```
SELECT SetIsDisjoint(ToSet(AsList(1, 2, 3)), AsList(7, 4)); -- true
SELECT SetIsDisjoint(ToSet(AsList(1, 2, 3)), ToSet(AsList(3, 4))); -- false
```

SetIntersection

Сигнатура

```
SetIntersection(Dict<K,V1>, Dict<K,V2>)->Set<K>
SetIntersection(Dict<K,V1>?, Dict<K,V2>)->Set<K>?
SetIntersection(Dict<K,V1>, Dict<K,V2>?)->Set<K>?
SetIntersection(Dict<K,V1>?, Dict<K,V2>?)->Set<K>?

SetIntersection(Dict<K,V1>, Dict<K,V2>, (K,V1,V2)->U)->Dict<K,U>
SetIntersection(Dict<K,V1>?, Dict<K,V2>, (K,V1,V2)->U)->Dict<K,U>?
SetIntersection(Dict<K,V1>, Dict<K,V2>?, (K,V1,V2)->U)->Dict<K,U>?
SetIntersection(Dict<K,V1>?, Dict<K,V2>?, (K,V1,V2)->U)->Dict<K,U>?
```

Строит пересечение двух словарей по ключам.

Аргументы:

- Два словаря: `Dict<K,V1>` и `Dict<K,V2>`.
- Необязательная функция, которая объединяет значения из исходных словарей для построения значений выходного словаря. Если тип такой функции `(K,V1,V2) -> U`, то типом результата будет `Dict<K,U>`. Если функция не задана, типом результата будет `Dict<K,Void>`, а значения из исходных словарей игнорируются.

Примеры

```
SELECT SetIntersection(ToSet(AsList(1, 2, 3)), ToSet(AsList(3, 4))); -- { 3 }
SELECT SetIntersection(
    AsDict(AsTuple(1, "foo"), AsTuple(3, "bar")),
    AsDict(AsTuple(1, "baz"), AsTuple(2, "qwe")),
    ($k, $a, $b) -> { RETURN AsTuple($a, $b) });
-- { 1 : ("foo", "baz") }
```

Примечание

В примере использовалась лямбда функция.

SetIncludes

Сигнатура

```
SetIncludes(Dict<K,V1>, List<K>)->Bool
SetIncludes(Dict<K,V1>?, List<K>)->Bool?
SetIncludes(Dict<K,V1>, List<K>?)->Bool?
SetIncludes(Dict<K,V1>?, List<K>?)->Bool?

SetIncludes(Dict<K,V1>, Dict<K,V2>)->Bool
SetIncludes(Dict<K,V1>?, Dict<K,V2>)->Bool?
SetIncludes(Dict<K,V1>, Dict<K,V2>?)->Bool?
SetIncludes(Dict<K,V1>?, Dict<K,V2>?)->Bool?
```

Проверка того, что в ключи заданного словаря входят все элементы списка или ключи второго словаря.

Таким образом есть два варианта вызова:

- С аргументами `Dict<K,V1>` и `List<K>`;
- С аргументами `Dict<K,V1>` и `Dict<K,V2>`.

Примеры

```
SELECT SetIncludes(ToSet(AsList(1, 2, 3)), AsList(3, 4)); -- false
SELECT SetIncludes(ToSet(AsList(1, 2, 3)), ToSet(AsList(2, 3))); -- true
```

SetUnion

Сигнатура

```
SetUnion(Dict<K,V1>, Dict<K,V2>)->Set<K>
SetUnion(Dict<K,V1>?, Dict<K,V2>)->Set<K>?
SetUnion(Dict<K,V1>, Dict<K,V2>?)->Set<K>?
SetUnion(Dict<K,V1>?, Dict<K,V2>?)->Set<K>?

SetUnion(Dict<K,V1>, Dict<K,V2>,(K,V1?,V2?)->U)->Dict<K,U>
SetUnion(Dict<K,V1>?, Dict<K,V2>,(K,V1?,V2?)->U)->Dict<K,U>?
SetUnion(Dict<K,V1>, Dict<K,V2>?,(K,V1?,V2?)->U)->Dict<K,U>?
SetUnion(Dict<K,V1>?, Dict<K,V2>?,(K,V1?,V2?)->U)->Dict<K,U>?
```

Строит объединение двух словарей по ключам.

Аргументы:

- Два словаря: `Dict<K,V1>` и `Dict<K,V2>`.
- Необязательная функция, которая объединяет значения из исходных словарей для построения значений выходного словаря. Если тип такой функции `(K,V1?,V2?) -> U`, то типом результата будет `Dict<K,U>`. Если функция не задана, типом результата будет `Dict<K,Void>`, а значения из исходных словарей игнорируются.

Примеры

```
SELECT SetUnion(ToSet(AsList(1, 2, 3)), ToSet(AsList(3, 4))); -- { 1, 2, 3, 4 }
SELECT SetUnion(
    AsDict(AsTuple(1, "foo"), AsTuple(3, "bar")),
    AsDict(AsTuple(1, "baz"), AsTuple(2, "qwe")),
    ($k, $a, $b) -> { RETURN AsTuple($a, $b) });
-- { 1 : ("foo", "baz"), 2 : (null, "qwe"), 3 : ("bar", null) }
```

SetDifference

Сигнатура

```
SetDifference(Dict<K,V1>, Dict<K,V2>)->Dict<K,V1>
SetDifference(Dict<K,V1>?, Dict<K,V2>)->Dict<K,V1>?
SetDifference(Dict<K,V1>, Dict<K,V2>?)->Dict<K,V1>?
SetDifference(Dict<K,V1>?, Dict<K,V2>?)->Dict<K,V1>?
```

Строит словарь, в котором есть все ключи с соответствующими значениями первого словаря, для которых нет ключа во втором словаре.

Примеры

```
SELECT SetDifference(ToSet(AsList(1, 2, 3)), ToSet(AsList(3, 4))); -- { 1, 2 }
SELECT SetDifference(
    AsDict(AsTuple(1, "foo"), AsTuple(2, "bar")),
    ToSet(AsList(2, 3)));
-- { 1 : "foo" }
```

SetSymmetricDifference

Сигнатура

```
SetSymmetricDifference(Dict<K,V1>, Dict<K,V2>)->Set<K>
SetSymmetricDifference(Dict<K,V1>?, Dict<K,V2>)->Set<K>?
SetSymmetricDifference(Dict<K,V1>, Dict<K,V2>?)->Set<K>?
SetSymmetricDifference(Dict<K,V1>?, Dict<K,V2>?)->Set<K>?

SetSymmetricDifference(Dict<K,V1>, Dict<K,V2>,(K,V1?,V2?)->U)->Dict<K,U>
SetSymmetricDifference(Dict<K,V1>?, Dict<K,V2>,(K,V1?,V2?)->U)->Dict<K,U>?
SetSymmetricDifference(Dict<K,V1>, Dict<K,V2>?,(K,V1?,V2?)->U)->Dict<K,U>?
SetSymmetricDifference(Dict<K,V1>?, Dict<K,V2>?,(K,V1?,V2?)->U)->Dict<K,U>?
```

Строит симметрическую разность двух словарей по ключам.

Аргументы:

- Два словаря: `Dict<K,V1>` и `Dict<K,V2>`.
- Необязательная функция, которая объединяет значения из исходных словарей для построения значений выходного словаря. Если тип такой функции `(K,V1?,V2?) -> U`, то типом результата будет `Dict<K,U>`. Если функция не задана, типом результата будет `Dict<K,Void>`, а значения из исходных словарей игнорируются.

Примеры

```
SELECT SetSymmetricDifference(ToSet(AsList(1, 2, 3)), ToSet(AsList(3, 4))); -- { 1, 2, 4 }
SELECT SetSymmetricDifference(
    AsDict(AsTuple(1, "foo"), AsTuple(3, "bar")),
    AsDict(AsTuple(1, "baz"), AsTuple(2, "qwe")),
    ($k, $a, $b) -> { RETURN AsTuple($a, $b) });
-- { 2 : (null, "qwe"), 3 : ("bar", null) }
```

Функции для работы со структурами

TryMember

Попытка получить значение поля из структуры, а в случае его отсутствия среди полей или null в значении структуры использовать значение по умолчанию.
Тип `default_value` должен совпадать с типом поля `key` из структуры.

Сигнатура

```
TryMember(struct:Struct<...>, key:String, default_value:T) -> T
TryMember(struct:Struct<...>?, key:String, default_value:T) -> T?
```

Аргументы:

1. struct - исходная структура;
2. key - имя поля;
3. default_value - значение по умолчанию если поле отсутствует

> **Примечание**
>
> Имя поля (key) не может зависеть от данных или от аргументов лямбды. В этом случае функция TryMember не может быть протипизирована.

Примеры

```
$struct = <|a:1|>;
SELECT
  TryMember(
    $struct,
    "a",
    123
  ) AS a, -- 1
  TryMember(
    $struct,
    "b",
    123
  ) AS b; -- 123
```

ExpandStruct

Добавление одного или нескольких новых полей в структуру. Возвращается новая раширенная структура. В случае возникновения дублей в наборе полей будет возвращена ошибка.

Сигнатура

```
ExpandStruct(struct:Struct<...>, value_1:T1 AS key_1:K, value_2:T2 AS key_2:K, ....) -> Struct<...>
```

Аргументы:

- В первый аргумент передается исходная структура для расширения.
- Все остальные аргументы должны быть именованными, каждый аргумент добавляет новое поле и имя аргумента используется в роли имени поля (по аналогии с AsStruct).

Примеры

```
$struct = <|a:1|>;
SELECT
  ExpandStruct(
    $struct,
    2 AS b,
    "3" AS c
  ) AS abc;  -- ("a": 1, "b": 2, "c": "3")
```

AddMember

Добавление одного нового поля в структуру. Если необходимо добавление нескольких полей, предпочтительнее использовать ExpandStruct.

В случае возникновения дублей в наборе полей будет возвращена ошибка.

Сигнатура

```
AddMember(struct:Struct<...>, new_key:String, new_value:T) -> Struct<...>
```

Аргументы:

1. struct - исходная структура;
2. new_key - имя нового поля;
3. new_value - значение нового поля.

Примеры

```
$struct = <|a:1|>;
SELECT
  AddMember(
    $struct,
    "b",
    2
  ) AS ab; -- ("a": 1, "b": 2)
```

RemoveMember

Удаление поля из структуры. Если указанного поля не существовало, будет возвращена ошибка.

Сигнатура

```
RemoveMember(struct:Struct<...>, key_to_delete:String) -> Struct<...>
```

Аргументы:

1. Исходная структура;
2. Имя поля для удаления

Примеры

```
$struct = <|a:1, b:2|>;
SELECT
  RemoveMember(
    $struct,
    "b"
  ) AS a; -- ("a": 1)
```

ForceRemoveMember

Удаление поля из структуры.

Если указанного поля не существовало, в отличии от RemoveMember ошибка возвращена не будет.

Сигнатура

```
ForceRemoveMember(struct:Struct<...>, key_to_delete:String) -> Struct<...>
```

Аргументы:

1. Исходная структура;
2. Имя поля для удаления.

Примеры

```
$struct = <|a:1, b:2|>;
SELECT
  ForceRemoveMember(
    $struct,
    "c"
  ) AS ab; -- ("a": 1, "b": 2)
```

ChooseMembers

Выделение из структуры полей с заданными именами. Возвращается новая структура только из заданных полей.

Если какого-либо из полей не существовало, будет возвращена ошибка.

Сигнатура

```
ChooseMembers(struct:Struct<...>, list_of_keys:List<String>) -> Struct<...>
```

Аргументы:

1. Исходная структура;
2. Список имен полей.

Примеры

```
$struct = <|a:1, b:2, c:3|>;
SELECT
  ChooseMembers(
    $struct,
    ["a", "b"]
  ) AS ab; -- ("a": 1, "b": 2)
```

RemoveMembers

Исключение из структуры полей с заданными именами.

Если какого-либо из полей не существовало, будет возвращена ошибка.

Сигнатура

```
RemoveMembers(struct:Struct<...>, list_of_delete_keys:List<String>) -> Struct<...>
```

Аргументы:

1. Исходная структура;
2. Список имен полей.

Примеры

```
$struct = <|a:1, b:2, c:3|>;
SELECT
  RemoveMembers(
    $struct,
    ["a", "b"]
  ) AS c; -- ("c": 3)
```

ForceRemoveMembers

Исключение из структуры полей с заданными именами.

Если какого-либо из полей не существовало, то оно игнорируется.

Сигнатура

```
ForceRemoveMembers(struct:Struct<...>, list_of_delete_keys:List<String>) -> Struct<...>
```

Аргументы:

1. Исходная структура;
2. Список имен полей.

Примеры

```
$struct = <|a:1, b:2, c:3|>;
SELECT
  ForceRemoveMembers(
    $struct,
    ["a", "b", "z"]
  ) AS c; -- ("c": 3)
```

CombineMembers

Объединение полей нескольких структур в новую структуру.

В случае возникновения дублей в результирующем наборе полей будет возвращена ошибка.

Сигнатура

```
CombineMembers(struct1:Struct<...>, struct2:Struct<...>, .....) -> Struct<...>
CombineMembers(struct1:Struct<...>?, struct2:Struct<...>?, .....) -> Struct<...>
```

Аргументы: две и более структуры.

Примеры

```
$struct1 = <|a:1, b:2|>;
$struct2 = <|c:3|>;
SELECT
  CombineMembers(
    $struct1,
    $struct2
  ) AS abc; -- ("a": 1, "b": 2, "c": 3)
```

FlattenMembers

Объединение полей нескольких новых структур в новую структуру с поддержкой префиксов.

В случае возникновения дублей в результирующем наборе полей будет возвращена ошибка.

Сигнатура

```
FlattenMembers(prefix_struct1:Tuple<String, Struct<...>>, prefix_struct2:Tuple<String, Struct<...>>, ...) ->
Struct<...>
```

Аргументы: два и более кортежа из двух элементов: префикс и структура.

Примеры

```
$struct1 = <|a:1, b:2|>;
$struct2 = <|c:3|>;
SELECT
  FlattenMembers(
```

```
    AsTuple("foo", $struct1), -- fooa, foob
    AsTuple("bar", $struct2)  -- barc
  ) AS abc; -- ("barc": 3, "fooa": 1, "foob": 2)
```

StructMembers

Возвращает неупорядоченный список имен полей (возможно, сняв один уровень опциональности) для единственного аргумента - структуры. Для `NULL` аргумента возвращается пустой список строк.

Сигнатура

```
StructMembers(struct:Struct<...>) -> List<String>
StructMembers(struct:Struct<...>?) -> List<String>
StructMembers(NULL) -> []
```

Аргумент: структура

Примеры

```
$struct = <|a:1, b:2|>;
SELECT
  StructMembers($struct) AS a, -- ['a', 'b']
  StructMembers(NULL) AS b; -- []
```

RenameMembers

Переименовывает поля в переданной структуре. При этом исходное поле можно переименовать в несколько новых. Все поля, не упомянутые в переименовании как исходные, переносятся в результирующую структуру. Если нет какого-то исходного поля в списке для переименования, выдается ошибка. Для опциональной структуры либо `NULL` таким же является и результат.

Сигнатура

```
RenameMembers(struct:Struct<...>, rename_rules:List<Tuple<String, String>>) -> Struct<...>
```

Аргументы:

1. Исходная структура;
2. Список имен полей в форме списка таплов: исходное имя, новое имя.

Примеры

```
$struct = <|a:1, b:2|>;
SELECT
  RenameMembers($struct, [('a', 'c'), ('a', 'e')]); -- (b:2, c:1, e:1)
```

ForceRenameMembers

Переименовывает поля в переданной структуре. При этом исходное поле можно переименовать в несколько новых. Все поля, не упомянутые в переименовании как исходные, переносятся в результирующую структуру. Если нет какого-то исходного поля в списке для переименования, оно игнорируется. Для опциональной структуры либо `NULL` таким же является и результат.

Сигнатура

```
ForceRenameMembers(struct:Struct<...>, rename_rules:List<Tuple<String, String>>) -> Struct<...>
```

Аргументы:

1. Исходная структура;
2. Список имен полей в форме списка таплов: исходное имя, новое имя.

Примеры

```
$struct = <|a:1, b:2|>;
SELECT
  ForceRenameMembers($struct, [('a', 'c'), ('d', 'e')]); -- (b:2, c:1)
```

GatherMembers

Возвращает неупорядоченный список таплов из имени поля и значения. Для `NULL` аргумента возвращается `EmptyList`. Можно использовать только в тех случаях, когда типы элементов в структуре одинаковы или совместимы. Для опциональной структуры возвращает опциональный список.

Сигнатура

```
GatherMembers(struct:Struct<...>) -> List<Tuple<String,V>>
GatherMembers(struct:Struct<...>?) -> List<Tuple<String,V>>?
GatherMembers(NULL) -> []
```

Аргумент: структура

Примеры

```
$struct = <|a:1, b:2|>;
SELECT
```

```
  GatherMembers($struct), -- [('a', 1), ('b', 2)]
  GatherMembers(null); -- []
```

SpreadMembers

Создает структуру с заданным списком полей и применяет к ней заданный список исправлений в формате (имя поля, значение). Все типы полей результирующей структуры совпадают, и равны типу значений в списке исправлений с добавленной опциональностью (если еще не были таковыми). Если поле не было упомянуто среди списка редактируемых полей, оно возвращается как `NULL`. Среди всех исправлений по одному полю сохраняется последнее. Если в списке исправлений встречается поле, которого нет в списке ожидаемых полей, выдается ошибка.

Сигнатура

```
SpreadMembers(list_of_tuples:List<Tuple<String, T>>, result_keys:List<String>) -> Struct<...>
```

Аргументы:

1. Список таплов: имя поля, значение поля;
2. Список всех возможных имен полей в структуре.

Примеры

```
SELECT
  SpreadMembers([('a',1),('a',2)],['a','b']); -- (a: 2, b: null)
```

ForceSpreadMembers

Создает структуру с заданным списком полей и применяет к ней заданный список исправлений в формате (имя поля, значение). Все типы полей результирующей структуры совпадают, и равны типу значений в списке исправлений с добавленной опциональностью (если еще не были таковыми). Если поле не было упомянуто среди списка редактируемых полей, оно возвращается как `NULL`. Среди всех исправлений по одному полю сохраняется последнее. Если в списке исправлений встречается поле, которого нет в списке ожидаемых полей, то это исправление игнорируется.

Сигнатура

```
ForceSpreadMembers(list_of_tuples:List<Tuple<String, T>>, result_keys:List<String>) -> Struct<...>
```

Аргументы:

1. Список таплов: имя поля, значение поля;
2. Список всех возможных имен полей в структуре.

Примеры

```
SELECT
  ForceSpreadMembers([('a',1),('a',2),('c',100)],['a','b']); -- (a: 2, b: null)
```

StructUnion, StructIntersection, StructDifference, StructSymmetricDifference

Комбинируют две структуры одним из четырех способов, используя предоставленную функцию для слияния полей с одинаковыми именами:

- `StructUnion` добавляет в результат все поля обеих структур;
- `StructIntersection` — поля, которые есть в обеих структурах;
- `StructDifference` — поля которые есть в left, но которых нет в right;
- `StructSymmetricDifference` — все поля, которые есть только в одной из структур.

Сигнатуры

```
StructUnion(left:Struct<...>, right:Struct<...>[, mergeLambda:(name:String, l:T1?, r:T2?)->T])->Struct<...>
StructIntersection(left:Struct<...>, right:Struct<...>[, mergeLambda:(name:String, l:T1?, r:T2?)->T])->Struct
<...>
StructDifference(left:Struct<...>, right:Struct<...>)->Struct<...>
StructSymmetricDifference(left:Struct<...>, right:Struct<...>)->Struct<...>
```

Аргументы:

1. `left` - первая структура;
2. `right` - вторая структура;
3. `mergeLambda` - *(опционально)* позволяет задать функцию для объединения полей (аргументы: имя поля, `Optional` значение поля в первой структуре, `Optional` значение поля во второй структуре - аргументы принимают значение `Nothing<T?>` в случае отсутствия соответствующего поля); по умолчанию выбирается значение поля из первой структуры, а если в первой отсутствует — из второй.

Примеры

```
$merge = ($name, $l, $r) -> {
    return ($l ?? 0) + ($r ?? 0);
};
$left = <|a: 1, b: 2, c: 3|>;
$right = <|c: 1, d: 2, e: 3|>;

SELECT
    StructUnion($left, $right),                -- <|a: 1, b: 2, c: 3, d: 2, e: 3|>
```

```
    StructUnion($left, $right, $merge),         -- <|a: 1, b: 2, c: 4, d: 2, e: 3|>
    StructIntersection($left, $right, $merge),  -- <|c: 4|>
    StructDifference($left, $right),            -- <|a: 1, b: 1|>
    StructSymmetricDifference($left, $right)    -- <|a: 1, b: 2, d: 2, e: 3|>
;
```

Функции для работы с типами данных

Помимо обычных функций, которые работают с конкретными значениями (типа FIND, COALESCE), YQL поддерживает функции для работами с типами.
Функции позволяют узнать тип произвольного выражения, проанализировать контейнерный тип и создать новый контейнерный тип на основе имеющегося.

Примеры

```
$itemType = TypeOf($item);
SELECT CAST($foo AS ListType($itemType));  -- каст $foo к типу List<$itemType>
```

FormatType

Сигнатура

```
FormatType(Type)->String
FormatType(TypeHandle)->String
```

Сериализация типа или хендла типа в человекочитаемую строку. Это полезно для отладки, а также будет использоваться в последующих примерах данного раздела. Документация по формату.

FormatTypeDiff и FormatTypeDiffPretty

Сигнатура

```
FormatTypeDiff(Type, Type)->String
FormatTypeDiff(TypeHandle, TypeHandle)->String

FormatTypeDiffPretty(Type, Type)->String
FormatTypeDiffPretty(TypeHandle, TypeHandle)->String
```

Получение строкового представления разницы двух типов или двух хендлов типов. Pretty-версия делает результирующую строку более читаемой путем добавления переводов строк и пробелов.

ParseType

Сигнатура

```
ParseType(String)->Type
```

Построение типа по строке с его описанием. Документация по её формату.

Примеры

```
SELECT FormatType(ParseType("List<Int32>"));  -- List<int32>
```

TypeOf

Сигнатура

```
TypeOf(<any expression>)->Type
```

Получение типа значения, переданного в аргумент.

Примеры

```
SELECT FormatType(TypeOf("foo"));  -- String
```

```
SELECT FormatType(TypeOf(AsTuple(1, 1u))); -- Tuple<Int32,Uint32>
```

InstanceOf

Сигнатура

```
InstanceOf(Type)->объект типа Type
```

Возвращает экземпляр объекта указанного типа. Полученный объект не имеет какого-то определенного значения.
InstanceOf можно использовать только в том случае, если результат выражения в котором InstanceOf используется зависит от типа InstanceOf, но не от значения.
В противном случае операция будет завершена с ошибкой.

Примеры

```
SELECT InstanceOf(ParseType("Int32")) + 1.0; -- ошибка (Can't execute InstanceOf): результат зависит от (неоп
ределенного) значения InstanceOf
SELECT FormatType(TypeOf(
    InstanceOf(ParseType("Int32")) +
    InstanceOf(ParseType("Double"))
```

```
)); -- вернет Double, так как сложение Int32 и Double возвращает Double (InstanceOf используется в контексте,
где важен только его тип, но не значение)
```

DataType

Сигнатура

```
DataType(String, [String, ...])->Type
```

Возвращает тип для примитивных типов данных по его имени.
Для некоторых типов (например Decimal) необходимо передавать параметры типа в качестве дополнительных аргументов.

Примеры

```
SELECT FormatType(DataType("Bool")); -- Bool
SELECT FormatType(DataType("Decimal","5","1")); -- Decimal(5,1)
```

OptionalType

Сигнатура

```
OptionalType(Type)->опциональный Type
```

Добавляет в переданный тип возможность содержать `NULL`.

Примеры

```
SELECT FormatType(OptionalType(DataType("Bool"))); -- Bool?
SELECT FormatType(OptionalType(ParseType("List<String?>"))); -- List<String?>?
```

ListType и StreamType

Сигнатура

```
ListType(Type)->тип списка с элементами типа Type
StreamType(Type)->тип потока с элементами типа Type
```

Строит тип списка или потока по переданному типу элемента.

Примеры

```
SELECT FormatType(ListType(DataType("Bool"))); -- List<Bool>
```

DictType

Сигнатура

```
DictType(Type, Type)->тип словаря
```

Строит тип словаря по переданным типам ключа (первый аргумент) и значения (второй аргумент).

Примеры

```
SELECT FormatType(DictType(
    DataType("String"),
    DataType("Double")
)); -- Dict<String,Double>
```

TupleType

Сигнатура

```
TupleType(Type, ...)->тип кортежа
```

Строит тип кортежа по переданным типам элементов.

Примеры

```
SELECT FormatType(TupleType(
    DataType("String"),
    DataType("Double"),
    OptionalType(DataType("Bool"))
)); -- Tuple<String,Double,Bool?>
```

StructType

Сигнатура

```
StructType(Type AS ElementName1, Type AS ElementName2, ...)->тип структуры
```

Строит тип структуры по переданным типам элементов. Для указания имен элементов используется стандартный синтаксис именованных аргументов.

Примеры

```
SELECT FormatType(StructType(
    DataType("Bool") AS MyBool,
    ListType(DataType("String")) AS StringList
)); -- Struct<'MyBool':Bool,'StringList':List<String>>
```

VariantType

Сигнатура

```
VariantType(StructType)->тип варианта над структурой
VariantType(TupleType)->тип варианта над кортежем
```

Возвращает тип варианта по низлежащему типу (структуры или кортежа).

Примеры

```
SELECT FormatType(VariantType(
  ParseType("Struct<foo:Int32,bar:Double>")
)); -- Variant<'bar':Double,'foo':Int32>
```

ResourceType

Сигнатура

```
ResourceType(String)->тип ресурса
```

Возвращает тип ресурса по переданной строковой метке.

Примеры

```
SELECT FormatType(ResourceType("Foo")); -- Resource<'Foo'>
```

CallableType

Сигнатура

```
CallableType(Uint32, Type, [Type, ...])->тип вызываемого значения
```

Строит тип вызываемого значения по следующим аргументам:

1. Число опциональных аргументов (если все обязательные — 0).
2. Тип результата.
3. Все последующие аргументы CallableType трактуются как типы аргументов вызываемого значения со сдвигом на два обязательных (например, третий аргумент CallableType описывает тип первого аргумента вызываемого значения).

Примеры

```
SELECT FormatType(CallableType(
  1, -- optional args count
  DataType("Double"), -- result type
  DataType("String"), -- arg #1 type
  OptionalType(DataType("Int64")) -- arg #2 type
)); -- Callable<(String,[Int64?])->Double>
```

GenericType, UnitType и VoidType

Сигнатура

```
GenericType()->тип
UnitType()->тип
VoidType()->тип
```

Возвращают одноименные специальные типы данных. Аргументов нет, так как они не параметризуются.

Примеры

```
SELECT FormatType(VoidType()); -- Void
```

OptionalItemType, ListItemType и StreamItemType

Сигнатура

```
OptionalItemType(OptionalType)->тип элемента опционального типа
ListItemType(ListType)->тип элемента списочного типа
```

```
StreamItemType(StreamType)->тип элемента потокового типа
```

Если этим функциям передается тип, то они выполняют действие, обратное OptionalType, ListType и StreamType — возвращают тип элемента по типу соответствующего контейнера.

Если этим функциям передается хендл типа, то выполняют действие, обратное OptionalTypeHandle, ListTypeHandle и StreamTypeHandle - возвращают хендл типа элемента по хендлу типа соответствующего контейнера.

Примеры

```
SELECT FormatType(ListItemType(
  ParseType("List<Int32>")
)); -- Int32
```

```
SELECT FormatType(ListItemType(
  ParseTypeHandle("List<Int32>")
)); -- Int32
```

DictKeyType и DictPayloadType

Сигнатура

```
DictKetType(DictType)->тип ключа словаря
DictPayloadType(DictType)->тип значения словаря
```

Возвращают тип ключа или значения по типу словаря.

Примеры

```
SELECT FormatType(DictKeyType(
  ParseType("Dict<Int32,String>")
)); -- Int32
```

TupleElementType

Сигнатура

```
TupleElementType(TupleType, String)->тип элемента кортежа
```

Возвращает тип элемента кортежа по типу кортежа и индексу элемента (индекс с нуля).

Примеры

```
SELECT FormatType(TupleElementType(
  ParseType("Tuple<Int32,Double>"), "1"
)); -- Double
```

StructMemberType

Сигнатура

```
StructMemberType(StructType, String)->тип элемента структуры
```

Возвращает тип элемента структуры по типу структуры и имени элемента.

Примеры

```
SELECT FormatType(StructMemberType(
  ParseType("Struct<foo:Int32,bar:Double>"), "foo"
)); -- Int32
```

CallableResultType и CallableArgumentType

Сигнатура

```
CallableResultType(CallableType)->тип результата вызываемого значения
CallableArgumentType(CallableType, Uint32)->тип аругмента вызываемого значения
```

`CallableResultType` возвращает тип результата по типу вызываемого значения, а `CallableArgumentType` — тип аргумента по типу вызываемого значения и его индексу (индекс с нуля).

Примеры

```
$callable_type = ParseType("(String,Bool)->Double");

SELECT FormatType(CallableResultType(
    $callable_type
)), -- Double
FormatType(CallableArgumentType(
```

```
    $callable_type, 1
)); -- Bool
```

VariantUnderlyingType

Сигнатура

```
VariantUnderlyingType(VariantType)->низлежащий тип варианта
```

Если этой функции передается тип, то она выполняет действие, обратное VariantType — возвращает низлежащий тип по типу варианта.

Если этой функции передается хендл типа, то она выполняет действие, обратное VariantTypeHandle — возвращает хендл низлежащего типа по хендлу типа варианта.

Примеры

```
SELECT FormatType(VariantUnderlyingType(
  ParseType("Variant<foo:Int32,bar:Double>")
)), -- Struct<'bar':Double,'foo':Int32>
FormatType(VariantUnderlyingType(
  ParseType("Variant<Int32,Double>")
)); -- Tuple<Int32,Double>
```

```
SELECT FormatType(VariantUnderlyingType(
  ParseTypeHandle("Variant<foo:Int32,bar:Double>")
)), -- Struct<'bar':Double,'foo':Int32>
FormatType(VariantUnderlyingType(
  ParseTypeHandle("Variant<Int32,Double>")
)); -- Tuple<Int32,Double>
```

Функции для работы с типами данных во время выполнения вычислений

Для работы с типами данных во время выполнения вычислений используется механизм хендлов типов - ресурс, содержащий непрозрачное описание типа. После конструирования хендла типа можно вернуться к обычному типу с помощью функции EvaluateType. Для отладки сконвертировать хендл типа в строку можно с помощью функции FormatType.

TypeHandle

Получение хендла типа из типа, переданного в аргумент.

Сигнатура

```
TypeHandle(Type)->хэндл типа
```

Примеры

```
SELECT FormatType(TypeHandle(TypeOf("foo")));  -- String
```

EvaluateType

Сигнатура

```
EvaluateType(TypeHandle)->тип
```

Получение типа из хендла типа, переданного в аргумент. Функция вычисляется до начала основного расчета, как и EvaluateExpr.

Примеры

```
SELECT FormatType(EvaluateType(TypeHandle(TypeOf("foo"))));  -- String
```

ParseTypeHandle

Сигнатура

```
ParseTypeHandle(String)->хэндл типа
```

Построение хендла типа по строке с его описанием. Документация по её формату.

Примеры

```
SELECT FormatType(ParseTypeHandle("List<Int32>"));  -- List<int32>
```

TypeKind

Сигнатура

```
TypeKind(TypeHandle)->String
```

Получение названия верхнего уровня типа из хендла типа, переданного в аргумент.

Примеры

```
SELECT TypeKind(TypeHandle(TypeOf("foo")));  -- Data
SELECT TypeKind(ParseTypeHandle("List<Int32>"));  -- List
```

DataTypeComponents

Сигнатура

```
DataTypeComponents(DataTypeHandle)->List<String>
```

Получение названия и параметров примитивного типа данных из хендла примитивного типа, переданного в аргумент. Обратная функция - DataTypeHandle.

Примеры

```
SELECT DataTypeComponents(TypeHandle(TypeOf("foo")));  -- ["String"]
SELECT DataTypeComponents(ParseTypeHandle("Decimal(4,1)"));  -- ["Decimal", "4", "1"]
```

DataTypeHandle

Сигнатура

```
DataTypeHandle(List<String>)->хэндл примитивного типа данных
```

Построение хендла примитивного типа данных из его названия и параметров, переданных списком в аргумент. Обратная функция - DataTypeComponents.

Примеры

```
SELECT FormatType(DataTypeHandle(
    AsList("String")
)); -- String

SELECT FormatType(DataTypeHandle(
    AsList("Decimal", "4", "1")
)); -- Decimal(4,1)
```

OptionalTypeHandle

Сигнатура

```
OptionalTypeHandle(TypeHandle)->хэндл опционального типа
```

Добавляет в переданный хендл типа возможность содержать `NULL`.

Примеры

```
SELECT FormatType(OptionalTypeHandle(
    TypeHandle(DataType("Bool"))
)); -- Bool?
```

PgTypeName

Сигнатура

```
PgTypeName(PgTypeHandle)->String
```

Получение имени PostgreSQL типа из хендла типа, переданного в аргумент. Обратная функция - PgTypeHandle.

Примеры

```
SELECT PgTypeName(ParseTypeHandle("pgint4")); -- int4
```

PgTypeHandle

Сигнатура

```
PgTypeHandle(String)->хендл типа
```

Построение хендла типа по имени PostgreSQL типа, переданного в аргумент. Обратная функция - PgTypeName.

Примеры

```
SELECT FormatType(PgTypeHandle("int4")); -- pgint4
```

ListTypeHandle и StreamTypeHandle

Сигнатура

```
ListTypeHandle(TypeHandle)->хэндл списочного типа
StreamTypeHandle(TypeHandle)->хэндл потокового типа
```

Строит хендл типа списка или потока по переданному хендлу типа элемента.

Примеры

```
SELECT FormatType(ListTypeHandle(
    TypeHandle(DataType("Bool"))
)); -- List<Bool>
```

EmptyListTypeHandle и EmptyDictTypeHandle

Сигнатура

```
EmptyListTypeHandle()->хэндл типа пустого списка
EmptyDictTypeHandle()->хэндл типа пустого словаря
```

Строит хендл типа пустого списка или словаря.

Примеры

```
SELECT FormatType(EmptyListTypeHandle()); -- EmptyList
```

TupleTypeComponents

Сигнатура

```
TupleTypeComponents(TupleTypeHandle)->List<TypeHandle>
```

Получение списка хендлов типов элементов из хендла типа кортежа, переданного в аргумент. Обратная функция - TupleTypeHandle.

Примеры

```
SELECT ListMap(
   TupleTypeComponents(
       ParseTypeHandle("Tuple<Int32, String>")
   ),
   ($x)->{
       return FormatType($x)
   }
); -- ["Int32", "String"]
```

TupleTypeHandle

Сигнатура

```
TupleTypeHandle(List<TypeHandle>)->хэндл типа кортежа
```

Построение хендла типа кортежа из хендлов типов элементов, переданных списком в аргумент. Обратная функция - TupleTypeComponents.

Примеры

```
SELECT FormatType(
    TupleTypeHandle(
        AsList(
            ParseTypeHandle("Int32"),
            ParseTypeHandle("String")
        )
    )
); -- Tuple<Int32,String>
```

StructTypeComponents

Сигнатура

```
StructTypeComponents(StructTypeHandle)->List<Struct<Name:String, Type:TypeHandle>>
```

Получение списка хендлов типов элементов и их имен из хендла типа структуры, переданного в аргумент. Обратная функция - StructTypeHandle.

Примеры

```
SELECT ListMap(
    StructTypeComponents(
        ParseTypeHandle("Struct<a:Int32, b:String>")
```

```
    ),
    ($x) -> {
        return AsTuple(
            FormatType($x.Type),
            $x.Name
        )
    }
); -- [("Int32","a"), ("String","b")]
```

StructTypeHandle

Сигнатура

```
StructTypeHandle(List<Struct<Name:String, Type:TypeHandle>>)->хэндл типа структуры
```

Построение хендла типа структуры из хендлов типов элементов и имен, переданных списком в аргумент. Обратная функция - StructTypeComponents.

Примеры

```
SELECT FormatType(
    StructTypeHandle(
        AsList(
            AsStruct(ParseTypeHandle("Int32") as Type,"a" as Name),
            AsStruct(ParseTypeHandle("String") as Type, "b" as Name)
        )
    )
); -- Struct<'a':Int32,'b':String>
```

DictTypeComponents

Сигнатура

```
DictTypeComponents(DictTypeHandle)->Struct<Key:TypeHandle, Payload:TypeHandle>
```

Получение хендла типа-ключа и хендла типа-значения - из хендла типа словаря, переданного в аргумент. Обратная функция - DictTypeHandle.

Примеры

```
$d = DictTypeComponents(ParseTypeHandle("Dict<Int32,String>"));

SELECT
    FormatType($d.Key),     -- Int32
    FormatType($d.Payload); -- String
```

DictTypeHandle

Сигнатура

```
DictTypeHandle(TypeHandle, TypeHandle)->хэндл типа словаря
```

Построение хендла типа словаря из хендла типа-ключа и хендла типа-значения, переданных в аргументы. Обратная функция - DictTypeComponents.

Примеры

```
SELECT FormatType(
    DictTypeHandle(
        ParseTypeHandle("Int32"),
        ParseTypeHandle("String")
    )
); -- Dict<Int32, String>
```

ResourceTypeTag

Сигнатура

```
ResourceTypeTag(ResourceTypeHandle)->String
```

Получение тега из хендла типа ресурса, переданного в аргумент. Обратная функция - ResourceTypeHandle.

Примеры

```
SELECT ResourceTypeTag(ParseTypeHandle("Resource<foo>")); -- foo
```

ResourceTypeHandle

Сигнатура

```
ResourceTypeHandle(String)->хэндл типа ресурса
```

Построение хендла типа ресурса по значению тега, переданного в аргумент. Обратная функция - ResourceTypeTag.

Примеры

```
SELECT FormatType(ResourceTypeHandle("foo")); -- Resource<'foo'>
```

TaggedTypeComponents

Сигнатура

```
TaggedTypeComponents(TaggedTypeHandle)->Struct<Base:TypeHandle, Tag:String>
```

Получение тега и базового типа из хендла декорированного типа, переданного в аргумент. Обратная функция - TaggedTypeHandle.

Примеры

```
$t = TaggedTypeComponents(ParseTypeHandle("Tagged<Int32,foo>"));

SELECT FormatType($t.Base), $t.Tag; -- Int32, foo
```

TaggedTypeHandle

Сигнатура

```
TaggedTypeHandle(TypeHandle, String)->хэндл декорированного типа
```

Построение хендла декорированного типа по хендлу базового типа и имени тега, переданных в аргументах. Обратная функция - TaggedTypeComponents.

Примеры

```
SELECT FormatType(TaggedTypeHandle(
    ParseTypeHandle("Int32"), "foo"
)); -- Tagged<Int32, 'foo'>
```

VariantTypeHandle

Сигнатура

```
VariantTypeHandle(StructTypeHandle)->хэндл типа варианта над структурой
VariantTypeHandle(TupleTypeHandle)->хэндл типа варианта над кортежем
```

Построение хендла типа варианта по хендлу низлежащего типа, переданного в аргумент. Обратная функция - VariantUnderlyingType.

Примеры

```
SELECT FormatType(VariantTypeHandle(
    ParseTypeHandle("Tuple<Int32, String>")
)); -- Variant<Int32, String>
```

VoidTypeHandle и NullTypeHandle

Сигнатура

```
VoidTypeHandle()->хэндл типа Void
NullTypeHandle()->хэндл типа Null
```

Построение хендла типов Void и Null соответственно.

Примеры

```
SELECT FormatType(VoidTypeHandle()); -- Void
SELECT FormatType(NullTypeHandle()); -- Null
```

CallableTypeComponents

Сигнатура

```
CallableTypeComponents(CallableTypeHandle)->
Struct<
    Arguments:List<Struct<
        Flags:List<String>,
        Name:String,
        Type:TypeHandle>>,
    OptionalArgumentsCount:Uint32,
    Payload:String,
    Result:TypeHandle
>
```

Получение описания хендла типа вызываемого значения, переданного в аргумент. Обратная функция - CallableTypeHandle.

Примеры

```
$formatArgument = ($x) -> {
    return AsStruct(
        FormatType($x.Type) as Type,
        $x.Name as Name,
        $x.Flags as Flags
    )
};

$formatCallable = ($x) -> {
    return AsStruct(
        $x.OptionalArgumentsCount as OptionalArgumentsCount,
        $x.Payload as Payload,
        FormatType($x.Result) as Result,
        ListMap($x.Arguments, $formatArgument) as Arguments
    )
};

SELECT $formatCallable(
    CallableTypeComponents(
        ParseTypeHandle("(Int32,[bar:Double?{Flags:AutoMap}])->String")
    )
);  -- (OptionalArgumentsCount: 1, Payload: "", Result: "String", Arguments: [
    --   (Type: "Int32", Name: "", Flags: []),
    --   (Type: "Double?", Name: "bar", Flags: ["AutoMap"]),
    -- ])
```

CallableArgument

Сигнатура

```
CallableArgument(TypeHandle, [String, [List<String>]])->Struct<Flags:List<String>,Name:String,Type:TypeHandle
>
```

Упаковка в структуру описания аргумента вызываемого значения для передачи в функцию CallableTypeHandle по следующим аргументам:

1. Хендл типа аргумента.
2. Необязательное имя аргумента. Значение по умолчанию - пустая строка.
3. Необязательные флаги аргумента в виде списка строк. Значение по умолчанию - пустой список. Поддерживаемые флаги - "AutoMap".

CallableTypeHandle

Сигнатура

```
CallableTypeHandle(TypeHandle, List<Struct<Flags:List<String>,Name:String,Type:TypeHandle>>, [Uint32, [Strin
g]])->хэндл типа вызываемого значения
```

Построение хендла типа вызываемого значения по следующим аргументам:

1. Хендл типа возвращаемого значения.
2. Список описаний аргументов, полученных через функцию CallableArgument.
3. Необязательное количество необязательных аргументов в вызываемом значении. Значение по умолчанию - 0.
4. Необязательная метка для типа вызываемого значения. Значение по умолчанию - пустая строка.

Обратная функция - CallableTypeComponents.

Примеры

```
SELECT FormatType(
    CallableTypeHandle(
        ParseTypeHandle("String"),
        AsList(
            CallableArgument(ParseTypeHandle("Int32")),
            CallableArgument(ParseTypeHandle("Double?"), "bar", AsList("AutoMap"))
        ),
        1
    )
);  -- Callable<(Int32,['bar':Double?{Flags:AutoMap}])->String>
```

LambdaArgumentsCount

Сигнатура

```
LambdaArgumentsCount(LambdaFunction)->Uint32
```

Получение количества аргументов в лямбда-функции.

Примеры

```
SELECT LambdaArgumentsCount(($x, $y)->($x+$y))
; -- 2
```

LambdaOptionalArgumentsCount

Сигнатура

```
LambdaOptionalArgumentsCount(LambdaFunction)->Uint32
```

Получение количества опциональных аргументов в лямбда-функции.

Примеры

```
SELECT LambdaOptionalArgumentsCount(($x, $y, $z?)->(if($x,$y,$z)))
; -- 1
```

Функции для работы с генерацией кода

Во время выполнения вычислений можно сгенерировать код, состоящий из узлов S-expressions. Для этого используется механизм представления кода, упакованного в ресурс. После конструирования кода можно подставить его в основную программу с помощью функции EvaluateCode. Для отладки сконвертировать код в строку можно с помощью функции FormatCode.

Возможные типы узлов в S-expressions, которые можно использовать для генерации кода:

- Атом - нетипизированная строка из нуля и более символов.
- Список - последовательность из нуля и более узлов. Соответствует типу `кортеж` в SQL.
- Вызов встроенной функции - состоит из имени, выраженного атомом, и последовательности из нуля и более узлов, которые являются аргументами этой функции.
- Объявление лямбда функции - состоит из объявления имен аргументов и узла, который является корнем тела этой лямбда функции.
- Аргумент лямбды функции - узел, который может использоваться только внутри тела лямбда-функции.
- Мир - специальный узел, маркирующий операции ввода/вывода.

Узлы S-expressions образуют ориентированный граф. При этом атомы - всегда листовые узлы, так как не могут содержать дочерних узлов.

В текстовой записи S-expressions записываются следующим образом:

- Атом - `'"foo"`. Символ апострофа (') является признаком цитирования последующей строки, обычно заключенной в кавычки.
- Список - `'("foo" "bar")`. Символ апострофа (') является признаком того, что в скобках не будет вызова функции.
- Вызов встроенной функции - `(foo "bar")`. Первый элемент внутри скобок - обязательное имя функции, а далее указываются ее аргументы.
- Объявление лямбда функции - `(lambda '(x y) (+ x y))`. После ключевого слова `lambda` стоит список из имен аргументов, за которым следует тело лямбда функции.
- Аргумент лямбда функции - `x`. В отличие от атома, строка без символа апострофа (') является ссылкой на имя в текущей области видимости. При объявлении лямбда функции в область видимости тела добавляются имена аргументов, причем при необходимости скрывается имя из объемлющей области видимости.
- Мир - `world`.

FormatCode

Сериализация кода в виде S-expressions. Код не должен содержать свободных аргументов функций, т.е. для сериализации кода лямбда функции нужно передавать ее целиком, а не выражения, потенциально содержащие аргументы лямбда функции.

Примеры

```
SELECT FormatCode(AtomCode("foo"));
-- (
-- (return '"foo")
-- )
```

WorldCode

Построить узел кода с типом `мир`.

Примеры

```
SELECT FormatCode(WorldCode());
-- (
-- (return world)
-- )
```

AtomCode

Построить узел кода с типом `атом` из строки, переданной в аргумент.

Примеры

```
SELECT FormatCode(AtomCode("foo"));
-- (
-- (return '"foo")
-- )
```

ListCode

Построить узел кода с типом `список` из набора узлов или списков узлов кода, переданных в аргументы. При этом списки из аргументов встраиваются как отдельно перечисленные узлы кода.

Примеры

```
SELECT FormatCode(ListCode(
    AtomCode("foo"),
    AtomCode("bar")));
-- (
-- (return '('"foo" '"bar"))
-- );

SELECT FormatCode(ListCode(AsList(
    AtomCode("foo"),
    AtomCode("bar"))));
```

```
-- (
-- (return '('"foo" '"bar"))
-- )
```

FuncCode

Построить узел кода с типом `вызов встроенной функции` из строки с именем функции и набора узлов или списков узлов кода, переданных в аргументы. При этом списки из аргументов встраиваются как отдельно перечисленные узлы кода.

Примеры

```
SELECT FormatCode(FuncCode(
    "Baz",
    AtomCode("foo"),
    AtomCode("bar")));
-- (
-- (return (Baz '"foo" '"bar"))
-- )

SELECT FormatCode(FuncCode(
    "Baz",
    AsList(
        AtomCode("foo"),
        AtomCode("bar"))));
-- (
-- (return (Baz '"foo" '"bar"))
-- )
```

LambdaCode

Построить узел кода с типом `объявление лямбда функции` можно из:

- Лямбда функции, если заранее известно количество аргументов. В этом случае в качестве аргументов этой лямбда функции будут переданы узлы типа `аргумент`.
- Количества аргументов и лямбда функции с одним аргументом. В этом случае в качестве аргумента этой лямбды функции будет передан список узлов типа `аргумент`.

Примеры

```
SELECT FormatCode(LambdaCode(($x, $y) -> {
    RETURN FuncCode("+", $x, $y);
}));
-- (
-- (return (lambda '($1 $2) (+ $1 $2)))
-- )

SELECT FormatCode(LambdaCode(2, ($args) -> {
    RETURN FuncCode("*", Unwrap($args[0]), Unwrap($args[1]));
}));
-- (
-- (return (lambda '($1 $2) (* $1 $2)))
-- )
```

EvaluateCode

Подстановка в основную программу узла кода, переданного в аргумент.

Примеры

```
SELECT EvaluateCode(FuncCode("Int32", AtomCode("1"))); -- 1

$lambda = EvaluateCode(LambdaCode(($x, $y) -> {
    RETURN FuncCode("+", $x, $y);
}));
SELECT $lambda(1, 2); -- 3
```

ReprCode

Подстановка в основную программу узла кода, который является представлением результата вычисления выражения, переданного в аргумент.

Примеры

```
$add3 = EvaluateCode(LambdaCode(($x) -> {
    RETURN FuncCode("+", $x, ReprCode(1 + 2));
}));
SELECT $add3(1); -- 4
```

QuoteCode

Подстановка в основную программу узла кода, который является представлением выражения или лямбда функции, переданной в аргумент. Если во время подстановки были найдены свободные аргументы лямбда функций, то они вычисляются и подставляются в код как в функции ReprCode.

Примеры

```
$lambda = ($x, $y) -> { RETURN $x + $y };
$makeClosure = ($y) -> {
    RETURN EvaluateCode(LambdaCode(($x) -> {
        RETURN FuncCode("Apply", QuoteCode($lambda), $x, ReprCode($y))
    }))
};

$closure = $makeClosure(2);
SELECT $closure(1); -- 3
```

Функции для работы с JSON

JSON - слабо схематизированный формат данных, представленный в YQL типом `Json`. В отличии от реляционных таблиц, JSON может хранить данные, для которых схема не определена. Вот пример валидного JSON:

```
[
    {
        "name": "Jim Holden",
        "age": 30
    },
    {
        "name": "Naomi Nagata",
        "age": "twenty years old"
    }
]
```

Несмотря на то, что в первом объекте поле `age` имеет тип `Number` (`"age": 21`), а во втором `String` (`"age": "twenty years old"`), это полностью валидный JSON.

Для работы с JSON, YQL реализует подмножество стандарта SQL support for JavaScript Object Notation (JSON), являющегося частью общепринятого стандарта ANSI SQL.

Cookbook

```
$json = CAST(@@{
    "friends": [
        {
            "name": "James Holden",
            "age": 35
        },
        {
            "name": "Naomi Nagata",
            "age": 30
        }
    ]
}@@ AS Json);

SELECT
    JSON_EXISTS($json, "$.friends[*].name"), -- True,
    JSON_VALUE($json, "$.friends[0].age"), -- "35" (тип Utf8?)
    JSON_QUERY($json, "$.friends[0]"); -- {"name": "James Holden", "age": 35}
```

Обращение к полю в базе данных

В таблицах данные могут хранить в JSON или в строковом представлении. Функции `JSON_*` для работы ожидают на вход JSON. Для преобразования `String->JSON` нужно воспользоваться функцией `CAST`, например `CAST (my_string AS JSON)`.

JsonPath

Для обращения к значениям внутри JSON используется язык запросов JsonPath. Все функции для работы с JSON принимают JsonPath запрос в качестве аргумента.

Сразу разберем пример. Пусть у нас есть JSON:

```
{
    "comments": [
        {
            "id": 123,
            "text": "A whisper will do, if it's all that you can manage."
        },
        {
            "id": 456,
            "text": "My life has become a single, ongoing revelation that I haven’t been cynical enough."
        }
    ]
}
```

Тогда чтобы получить текст второго комментария можно написать JsonPath запрос:

```
$.comments[1].text
```

В этом запросе:

1. `$` - это способ обратиться ко всему JSON
2. `$.comments` - обращение к ключу `comments` объекта JSON
3. `$.comments[1]` - обращение ко второму элементу массива JSON (нумерация с элементов начинается с 0)
4. `$.comments[1].text` - обращение к ключу `text` объекта JSON
5. Результат выполнения: `"My life has become a single, ongoing revelation that I haven’t been cynical enough."`

Quick reference

Операция	Пример
Извлечь ключ JSON объекта	`$.key`

Извлечь все ключи JSON объекта	`$.*`
Обращение к элементу массива	`$[25]`
Извлечение подотрезка массива	`$[2 to 5]`
Обращение к последнему элементу массива	`$[last]`
Обращение ко всем элементам массива	`$[*]`
Унарные операции	`- 1`
Бинарные операции	`(12 * 3) % 4 + 8`
Обращение к переменной	`$variable`
Логические операции	`(1 > 2) || (3 <= 4) && ("string" == "another")|`
Соответствие регулярному выражению	`$.name like_regex "^[A-Za-z]+$"`
Проверка префикса строки	`$.name starts with "Bobbie"`
Проверка существования пути	`exists ($.profile.name)`
Проверка булевского выражения на null	`($.age > 20) is unknown`
Фильтрация значений	`$.friends ? (@.age >= 18 && @.gender == "male")`
Получение типа значения	`$.name.type()`
Получение размера массива	`$.friends.size()`
Преобразование строки в число	`$.number.double()`
Округление числа вверх	`$.number.ceiling()`
Округление числа вниз	`$.number.floor()`
Абсолютное значение числа	`$.number.abs()`
Получение пар ключ-значения из объекта	`$.profile.keyvalue()`

Модель данных

Результат выполнения всех JsonPath выражений - это последовательность JSON значений. Например:

- Результат выражения `"Bobbie"` - это последовательность длины 1 с единственным элементом `"Bobbie"`.
- Результат выражения `$` (взятие всего JSON объекта) на JSON `[1, 2, 3]` - это `[1, 2, 3]`. Последовательность из 1 элемента, массива `[1, 2, 3]`
- Результат выражения `$[*]` (извлечение всех элементов массива) на JSON `[1, 2, 3]` - это `1, 2, 3`. Последовательность из трех элементов `1`, `2` и `3`

Если входная последовательность состоит из нескольких значений, некоторые операции исполнятся для каждого элемента (например, доступ к ключу JSON объекта). При этом другие операции требуют последовательности из одного элемента на вход (например, бинарные арифметические операции).

Поведение конкретной операции описано в соответствующей секции документации.

Режим выполнения

JsonPath поддерживает два режима выполнения - `lax` и `strict`. Указание режима не обязательно, по умолчанию используется `lax`. Режим указывается в начале запроса, например: `strict $.key`.

Поведение при каждом режиме описано в разделе соответствующих JsonPath операций.

Автоматическая распаковка массива

При обращении к ключу JSON объекта в `lax` режиме массивы автоматически распаковываются.

Пример

```
[
    {
        "key": 123
    },
    {
        "key": 456
    }
]
```

Запрос `lax $.key` успешно выполнится с результатом `123, 456`. Поскольку `$` является массивом, он автоматически распакуется, и обращение к ключу JSON объекта `$.key` будет выполнено для каждого элемента массива.

При этом запрос `strict $.key` завершится с ошибкой. В `strict` режиме нет автоматической распаковки массивов. `$` является массивом, а не объектом, поэтому обращение к ключу объекта `$.key` не может быть выполнено. Это можно исправить, написав `strict $[*].key`.

Распаковка происходит только на 1 уровень вглубь. В случае вложенных массивов распаковывается только самый внешний из них.

Обертка в массив

При обращении к элементу массива в `lax` режиме JSON значения автоматически оборачиваются в массив.

Пример

```
{
    "name": "Avasarala"
}
```

Запрос `lax $[0].name` выполнится успешно с результатом `"Avasarala"`. Поскольку `$` не является массивом, он будет автоматически обернут в массив длины один. Обращение к первому элементу `$[0]` вернет исходный JSON объект, в котором будет взят ключ `name`.

При этом запрос `strict $[0].name` завершится с ошибкой. В `strict` режиме нет автоматической обертки в массив. `$` является объектом, а не массивом, поэтому обращение к элементу `$[0]` не может быть выполнено. Это можно исправить, написав `strict $.name`.

Обработка ошибок

Некоторые ошибки конвертируются в пустой результат при выполнении в `lax` режиме.

Литералы

Значения некоторых типов можно указать в JsonPath запросе используя литералы:

Тип	Пример
Числа	`42`, `-1.23e-5`
Булевские значения	`false`, `true`
Null	`null`
Строки	`"Belt"`

Обращение к ключу JSON объекта

JsonPath поддерживает обращение к ключам JSON объектов: `$.session.user.name`.

> **Примечание**
>
> Обращение без кавычек можно использовать только для ключей, которые начинаются с буквы английского алфавита или подчеркивания, и содержат в себе только буквы английского алфавита, подчеркивания, цифры и знак доллара. Для всех остальных ключей необходимо использовать кавычки. Например: `$.profile."this string has spaces"`, `$.user."42 is the answer"`

Для каждого значения из входной последовательности:

1. Если значение является массивом, в `lax` режиме происходит автоматическая распаковка массива
2. Если значение не является JSON объектом или указанный ключ в этом JSON объекте отсутствует, в `strict` режиме запрос завершается ошибкой. В `lax` режиме для этого значения возвращается пустой результат

Результат выражения - конкатенация результатов для каждого значения из входной последовательности.

Пример

```
{
    "name": "Amos",
    "friends": [
        {
            "name": "Jim"
        },
        {
            "name": "Alex"
        }
    ]
}
```

	`lax`	`strict`
`$.name`	`"Amos"`	`"Amos"`
`$.surname`	Пустой результат	Ошибка
`$.friends.name`	`"Jim", "Alex"`	Ошибка

Обращение ко всем ключам JSON объекта

JsonPath поддерживает обращение ко всем ключам JSON объектов сразу: `$.*`.

Для каждого значения из входной последовательности:

1. Если значение является массивом, в `lax` режиме происходит автоматическая распаковка массива
2. Если значение не является JSON объектом, в `strict` режиме запрос завершается ошибкой. В `lax` режиме для этого значения возвращается пустой результат

Результат выражения - конкатенация результатов для каждого значения из входной последовательности.

Пример

```
{
    "profile": {
        "id": 123,
        "name": "Amos"
    },
    "friends": [
        {
            "name": "Jim"
        },
        {
            "name": "Alex"
        }
    ]
}
```

	`lax`	`strict`
`$.profile.*`	`123, "Amos"`	`123, "Amos"`
`$.friends.*`	`"Jim", "Alex"`	Ошибка

Обращение к элементу массива

JsonPath поддерживает обращение к элементам массивов: `$.friends[1, 3 to last - 1]`.

Для каждого значения из входной последовательности:

1. Если значение не является массивом, то в `strict` режиме запрос завершается ошибкой. В `lax` режиме происходит автоматическая обертка в массив
2. Ключевое слово `last` заменяется на последний индекс массива. Использование `last` вне обращения к массиву - это ошибка в обоих режимах
3. Вычисляются указанные индексы. Каждый из них должен быть единственным числом, иначе запрос завершается ошибкой в обоих режимах
4. Если индекс является дробным числом, он округляется вниз
5. Если индекс выходит за границы массива, то в `strict` режиме запрос завершается ошибкой. В `lax` режиме такой индекс игнорируется
6. Если указан отрезок и его стартовый индекс больше конечного индекса (например `$[20 to 1]`), то в `strict` режиме запрос завершается ошибкой. В `lax` режиме такой отрезок игнорируется.
7. К результату добавляются все элементы по указанным индексам. Отрезки включают в себя **оба конца**

Примеры

```
[
    {
        "name": "Camina",
        "surname": "Drummer"
    },
    {
        "name": "Josephus",
        "surname": "Miller"
    },
    {
        "name": "Bobbie",
        "surname": "Draper"
    },
    {
        "name": "Julie",
        "surname": "Mao"
    }
]
```

	`lax`	`strict`
`$[0].name`	`"Camina"`	`"Camina"`
`$[1, 2 to 3].name`	`"Josephus", "Bobbie", "Julie"`	`"Josephus", "Bobbie", "Julie"`
`$[last - 2].name`	`"Josephus"`	`"Josephus"`
`$[2, last + 200 to 50].name`	`"Bobbie"`	Ошибка
`$[50].name`	Пустой результат	Ошибка

Обращение ко всем элементам массива

JsonPath поддерживает обращение ко всем элементам массива сразу: `$[*]`.

Для каждого значения из входной последовательности:

1. Если значение не является массивом, то в `strict` режиме запрос завершается ошибкой. В `lax` режиме происходит автоматическая обертка в массив
2. К результату добавляются все элементы текущего массива

Примеры

```
[
    {
        "class": "Station",
        "title": "Medina"
    },
    {
        "class": "Corvette",
        "title": "Rocinante"
    }
]
```

	`lax`	`strict`
`$[*].title`	`"Medina", "Rocinante"`	`"Medina", "Rocinante"`
`lax $[0][*].class`	`"Station"`	Ошибка

Разберем последний пример по шагам:

1. `$[0]` возвращает первый элемент массива, то есть `{"class": "Station", "title": "Medina"}`
2. `$[0][*]` ожидает массив на вход, но был дан объект. Происходит автоматическая обертка в массив, получается `[ {"class": "Station", "title": "Medina"} ]`
3. Теперь `$[0][*]` может выполниться и возвращает все элементы массива, то есть `{"class": "Station", "title": "Medina"}`
4. `$[0][*].class` возвращает поле `class`, то есть `"Station"`.

Арифметические операции

> **Примечание**
>
> Все арифметические операции работают с числами как с Double. Возможна потеря точности при вычислениях.

Унарные операции

JsonPath поддерживает унарный `+` и `-`.

Унарная операция применяется ко всем значениям из входной последовательности. Если унарной операции подать на вход не число, запрос завершится ошибкой в обоих режимах.

Пример

```
[1, 2, 3, 4]
```

Запрос `strict -$[*]` завершится успешно с результатом `-1, -2, -3, -4`.

Запрос `lax -$` завершится с ошибкой, потому что `$` является массивом, а не числом.

Бинарные операции

JsonPath поддерживает бинарные арифметические операции (в порядке убывания приоритета):

1. Умножение `*`, деление чисел с плавающей точкой `/`, взятие остатка `%` (работает как функция `MOD` в `SQL`)
2. Сложение `+`, вычитание `-`

Порядок выполнения операций можно поменять используя скобочки.

В случае если каждый аргумент бинарной операции не является единственным числом или происходит деление на ноль, запрос завершается ошибкой в обоих режимах.

Примеры

- `(1 + 2) * 3` даст `9`
- `1 / 2` даст `0.5`
- `5 % 2` даст `1`
- `1 / 0` завершится ошибкой
- При JSON `[-32.4, 5.2]` запрос `$[0] % $[1]` даст `-1.2`
- При JSON `[1, 2, 3, 4]` запрос `lax $[*] + $[*]` завершится ошибкой, так как результат выражения `$[*]` - это `1, 2, 3, 4`, несколько чисел. Бинарная операция требует только одно число для каждого своего аргумента.

Булевские значения

В отличии от некоторых языков программирования, в JsonPath булевскими значениями считаются не только `true` (истина) и `false` (ложь), но и `null` (неопределенность).

JsonPath считает все значения, полученные из JSON документа, не булевскими. Например, запрос `! $.is_valid_user` (логическое отрицание, примененное к полю `is_valid_user`), является синтаксически неверным поскольку поле `is_valid_user` не является булевским значением (даже если по факту в нем хранится `true` или `false`). Правильный способ написать такой запрос - это явно использовать сравнение с булевским значением, например `$.is_valid_user == false`.

Логические операции

JsonPath поддерживает некоторые логические операции для работы с булевскими значениями.

Аргументы всех логических операций должны являться единственным булевским значением.
Все логические операции возвращают булевское значение.

Логическое отрицание !

Таблица истинности:

`x`	`!x`
`true`	`false`
`false`	`true`
`null`	`null`

Логическое И &&

Таблица истинности, первый столбец это левый аргумент, первая строка это правый аргумент, каждая клетка это результат применения логического И с левым и правым аргументами:

`&&`	`true`	`false`	`null`
`true`	`true`	`false`	`null`
`false`	`false`	`false`	`false`
`null`	`null`	`false`	`null`

Логическое ИЛИ ||

Таблица истинности, первый столбец это левый аргумент, первая строка это правый аргумент, каждая клетка это результат применения логического ИЛИ с левым и правым аргументами:

`||`	`true`	`false`	`null`
`true`	`true`	`true`	`true`
`false`	`true`	`false`	`null`
`null`	`true`	`null`	`null`

Примеры

- `! (true == true)`, результат `false`
- `(true == true) && (true == false)`, результат `false`
- `(true == true) || (true == false)`, результат `true`

Операторы сравнения

JsonPath реализует операторы сравнения для значений:

- Равенство, `==`
- Неравенство, `!=` и `<>`
- Меньше и меньше либо равно, `<` и `<=`
- Больше и больше либо равно, `>` и `>=`

Все операторы сравнения возвращают булевское значение. Оба аргумента оператора поддерживают наличие нескольких значений.

Если при вычислении аргументов оператора возникла ошибка, оператор возвращает `null`. При этом выполнение JsonPath запроса продолжается.

Для каждого из аргументов производится автоматическая распаковка массивов. После этого для каждой пары, где первый элемент берется из последовательности левого аргумента, а второй элемент из последовательности правого аргумента:

1. Выполняется сравнение элементов пары
2. Если при сравнении возникла ошибка, устанавливается флаг `ERROR`
3. Если результат сравнения это истина, устанавливается флаг `FOUND`
4. Если один из флагов `ERROR` или `FOUND` установлен и запрос исполняется в `lax` режиме, больше никакие пары не рассматриваются

Если после рассмотрения пар:

1. Установлен флаг `ERROR`, оператор возвращает `null`
2. Установлен флаг `FOUND`, оператор возвращает `true`
3. Иначе оператор возвращает `false`

Можно сказать что данный алгоритм рассматривает все пары из декартового произведения левого и правого аргумента, пытаясь найти ту, сравнение которой вернет истину.

Сравнение элементов в паре производится по следующим правилам:

1. Если левый или правый аргумент являются массивом или объектом, сравнение завершается с ошибкой
2. `null == null` возвращает истину
3. Во всех остальных случаях если один из аргументов `null`, возвращается ложь
4. Если левый и правый аргумент разных типов, сравнение завершается с ошибкой
5. Строки сравниваются побайтово
6. `true` считается больше `false`
7. Числа сравниваются с точностью `1e-20`

Пример

Для примера рассмотрим JSON документ

```
{
    "left": [1, 2],
    "right": [4, "Inaros"]
}
```

и разберем по шагам выполнение запроса `lax $.left < $.right`:

1. Автоматическая распаковка массивов в левом и правом аргументе. В качестве левого аргумента получаем последовательность `1, 2`, в качестве правого `4, "Iranos"`
2. Рассматриваем пару `(1, 4)`. Сравнение проходит успешно, `1 < 4` это истина. Устанавливаем флаг `FOUND`
3. Поскольку выполнение происходит в `lax` режиме и установлен флаг `FOUND`, больше никакие пары мы не рассматриваем
4. Поскольку установлен флаг `FOUND`, оператор возвращает истину

Разберем тот же запрос, но в другом режиме выполнения: `strict $.left < $.right`:

1. Автоматическая распаковка массивов в левом и правом аргументе. В качестве левого аргумента получаем последовательность `1, 2`, в качестве правого `4, "Iranos"`
2. Рассматриваем пару `(1, 4)`. Сравнение проходит успешно, `1 < 4` это истина. Устанавливаем флаг `FOUND`
3. Рассматриваем пару `(2, 4)`. Сравнение проходит успешно, `2 < 4` это истина. Устанавливаем флаг `FOUND`
4. Рассматриваем пару `(1, "Iranos")`. Сравнение завершается ошибкой (число нельзя сравнить со строкой). Устанавливаем флаг `ERROR`
5. Рассматриваем пару `(2, "Iranos")`. Сравнение завершается ошибкой (число нельзя сравнить со строкой). Устанавливаем флаг `ERROR`
6. Поскольку установлен флаг `ERROR`, оператор возвращает `null`

Предикаты

JsonPath поддерживает предикаты - выражения, возвращающие булевское значение, и проверяющие некоторое условие. Их можно использовать, например, в фильтрах.

`like_regex`

Предикат `like_regex` позволяет проверить строку на соответствие регулярному выражению. Синтаксис регулярных выражений такой же как в Hyperscan UDF и REGEXP.

Синтаксис

```
<expression> like_regex <regexp string> [flag <flag string>]
```

Где:

1. `<expression>` - это JsonPath выражение, содержащее строки, которые следует проверить на соответствие регулярному выражению
2. `<regexp string>` - это строка, содержащая регулярное выражение
3. `flag <flag string>` - это опциональная секция, в которой `<flag string>` является строкой с флагами для исполнения регулярного выражения

Поддерживаемые флаги:

- `i` - отключение чувствительности к регистру

Исполнение

Перед проверкой производится автоматическая распаковка массивов во входной последовательности.

После этого для каждого элемента входной последовательности:

1. Выполняется проверка на соответствие элемента регулярному выражению
2. Если элемент не является строкой, устанавливается флаг `ERROR`
3. Если результат проверки это истина, устанавливается флаг `FOUND`
4. Если один из флагов `ERROR` или `FOUND` установлен и запрос исполняется в `lax` режиме, больше никакие пары не рассматриваются

Если после рассмотрения пар:

1. Установлен флаг `ERROR`, предикат возвращает `null`
2. Установлен флаг `FOUND`, предикат возвращает `true`
3. Иначе предикат возвращает `false`

Примеры

1. `"123456" like_regex "^[0-9]+$"` возвращает `true`
2. `"123abcd456" like_regex "^[0-9]+$"` возвращает `false`
3. `"Naomi Nagata" like_regex "nag"` возвращает `false`
4. `"Naomi Nagata" like_regex "nag" flag "i"` возвращает `true`

`starts with`

Предикат `starts with` позволяет проверить является ли одна строка префиксом другой.

Синтаксис

```
<string expression> starts with <prefix expression>
```

Где:

1. `<string expression>` - это JsonPath выражение, содержащее строку, которую нужно проверить
2. `<prefix expression>` - это JsonPath выражение, содержащее строку-префикс

То есть предикат будет проверять что `<string expression>` начинается со строки `<prefix expression>`.

Исполнение

Первый аргумент предиката должен быть единственной строкой.

Второй аргумент предиката должен быть последовательностью (возможно, из нескольких) строк.

Для каждого элемента из последовательности строк-префиксов:

1. Выполняется проверка "является ли элемент префиксом входной строки"
2. Если элемент не является строкой, устанавливается флаг `ERROR`
3. Если результат проверки это истина, устанавливается флаг `FOUND`
4. Если один из флагов `ERROR` или `FOUND` установлен и запрос исполняется в `lax` режиме, больше никакие пары не рассматриваются

Если после рассмотрения пар:

1. Установлен флаг `ERROR`, предикат возвращает `null`
2. Установлен флаг `FOUND`, предикат возвращает `true`
3. Иначе предикат возвращает `false`

Примеры

1. `"James Holden" starts with "James"` возвращает `true`
2. `"James Holden" starts with "Amos"` возвращает `false`

`exists`

Предикат `exists` позволяет проверить, возвращает ли JsonPath выражение хотя бы один элемент.

Синтаксис

```
exists (<expression>)
```

Где `<expression>` - это JsonPath выражение, которое нужно проверить. Скобки вокруг выражения обязательны.

Исполнение

1. Исполняется переданное JsonPath выражение
2. Если в результате исполнения возникла ошибка, предикат возвращает `null`
3. Если в результате исполнения была получена пустая последовательность, предикат возвращает `false`
4. Иначе предикат возвращает `true`

Примеры

Рассмотрим JSON документ:

```
{
    "profile": {
        "name": "Josephus",
        "surname": "Miller"
    }
}
```

1. `exists ($.profile.name)` возвращает `true`
2. `exists ($.friends.profile.name)` возвращает `false`
3. `strict exists ($.friends.profile.name)` возвращает `null`, потому что в `strict` режиме обращение к несуществующим ключам объекта это ошибка

`is unknown`

Предикат `is unknown` позволяет проверить, является ли булевское значение `null`.

Синтаксис

```
(<expression>) is unknown
```

Где `<expression>` - это JsonPath выражение, которое нужно проверить. Допускаются только выражения, возвращающие булевское значение. Скобки вокруг выражения обязательны.

Исполнение

1. Если переданное выражение возвращает `null`, предикат возвращает `true`
2. Иначе предикат возвращает `false`

Примеры

1. `(1 == 2) is unknown` возвращает `false`. Выражение `1 == 2` вернуло `false`, что не является `null`

2. `(1 == "string") is unknown` возвращает `true`. Выражение `1 == "string"` вернуло `null`, поскольку в JsonPath строки и числа не сравнимы

Фильтры

JsonPath позволяет фильтровать значения, полученные в ходе выполнения запроса.

Выражение в фильтре должно возвращать булевское значение.
Перед фильтрацией производится автоматическая распаковка массивов во входной последовательности.

Для каждого элемента входной последовательности:

1. Значение текущего фильтруемого объекта `@` становится равным текущему элементу входной последовательности
2. Исполняется выражение в фильтре
3. Если в ходе выполнения выражения возникла ошибка, текущий элемент входной последовательности пропускается
4. Если результат исполнения выражения это единственное значение `true`, текущий элемент добавляется в результат фильтра

Пример

Пусть у нас есть JSON документ, описывающий друзей пользователя

```
{
    "friends": [
        {
            "name": "James Holden",
            "age": 35,
            "money": 500
        },
        {
            "name": "Naomi Nagata",
            "age": 30,
            "money": 345
        }
    ]
}
```

и мы хотим получить с помощью JsonPath запроса друзей которые старше 32 лет. Для этого можно написать следующий запрос:

```
$.friends ? (@.age > 32)
```

Разберем запрос по частям:

- `$.friends` - обращение к массиву `friends` в JSON документе
- `? ( ... )` - синтаксис фильтра. Выражение внутри скобок называется предикатом
- `@` - обращение к текущему фильтруемому объекту. В нашем примере это объект, описывающий друга пользователя
- `@.age` - обращение к полю `age` текущего фильтруемого объекта
- `@.age > 32` - сравнение поля `age` со значением 32. В результате выполнения запроса останутся только те значения, для которых данный предикат вернул истину

В результате выполнения этого запроса будет получен только первый друг из массива друзей пользователя.

Как и многие другие операторы языка JsonPath, фильтры можно выстраивать в цепочки. Разберем более сложный запрос, который отбирает имена друзей которые старше 20 лет и которые имеют меньше чем 400 единиц валюты:

```
$.friends ? (@.age > 20) ? (@.money < 400) . name
```

Разберем запрос по частям:

- `$.friends` - обращение к массиву `friends` в JSON документе
- `? (@.age > 20)` - первый фильтр. Поскольку все друзья старше 20, он просто вернет все элементы массива `friends`
- `? (@.money < 400)` - второй фильтр. Вернет только второй элемент массива `friends`, поскольку только у него поле `money` имеет значение меньше 400
- `.name` - обращение к полю `name` у отфильтрованных объектов

В результате выполнения этого запроса будет получена последовательность из одного элемента - `"Naomi Nagata"`.

На практике рекомендуется объединять несколько фильтров в один если есть такая возможность. Рассмотренный запрос эквивалентен `$.friends ? (@.age > 20 && @.money < 400) . name`.

Методы

JsonPath поддерживает методы - функции, которые преобразуют одни последовательности значений в другие. Синтаксис вызова метода похож на обращение к ключу объекта:

```
$.friends.size()
```

Также как и обращение к ключам объекта, вызовы методов можно выстраивать в цепочки:

```
$.numbers.double().floor()
```

`type`

Метод `type` возвращает строчку с названием типа переданного значения.

Для каждого элемента входной последовательности метод добавляет строчку в выходную последовательность в соответствии с табличкой:

Тип значения	Строка с названием типа
Null	`"null"`
Булевское значение	`"boolean"`
Число	`"number"`
Строка	`"string"`
Массив	`"array"`
Объект	`"object"`

Примеры

1. `"Naomi".type()` возвращает `"string"`
2. `false.type()` возвращает `"boolean"`

`size`

Метод `size` возвращает размер массива.

Для каждого элемента входной последовательности метод добавляет в выходную последовательность:

1. Размер массива, если тип элемента это массив
2. Для всех остальных типов (включая объекты) `1`

Примеры

Рассмотрим JSON документ:

```
{
    "array": [1, 2, 3],
    "object": {
        "a": 1,
        "b": 2
    },
    "scalar": "string"
}
```

И запросы к нему:

1. `$.array.size()` возвращает `3`
2. `$.object.size()` возвращает `1`
3. `$.scalar.size()` возвращает `1`

`double`

Метод `double` конвертирует строки в числа.

Перед выполнением производится автоматическая распаковка массивов во входной последовательности.

Все элементы во входной последовательности должны быть строками, которые содержат числа в десятичной записи. Допускается указание дробной части и экспоненты.

Примеры

1. `"125".double()` возвращает `125`
2. `"125.456".double()` возвращает `125.456`
3. `"125.456e-3".double()` возвращает `0.125456`

`ceiling`

Метод `ceiling` округляет числа вверх.

Перед выполнением производится автоматическая распаковка массивов во входной последовательности.

Все элементы во входной последовательности должны быть числами.

Примеры

1. `(1.3).ceiling()` возвращает `2`
2. `(1.8).ceiling()` возвращает `2`
3. `(1.5).ceiling()` возвращает `2`
4. `(1.0).ceiling()` возвращает `1`

`floor`

Метод `floor` округляет числа вниз.

Перед выполнением производится автоматическая распаковка массивов во входной последовательности.

Все элементы во входной последовательности должны быть числами.

Примеры

1. `(1.3).floor()` возвращает `1`
2. `(1.8).floor()` возвращает `1`

3. `(1.5).floor()` возвращает `1`
4. `(1.0).floor()` возвращает `1`

`abs`

Метод `abs` вычисляет абсолютное значение числа (убирает знак).

Перед выполнением производится автоматическая распаковка массивов во входной последовательности.

Все элементы во входной последовательности должны быть числами.

Примеры

1. `(0.0).abs()` возвращает `0`
2. `(1.0).abs()` возвращает `1`
3. `(-1.0).abs()` возвращает `1`

`keyvalue`

Метод `keyvalue` конвертирует объект в последовательность пар ключ-значение.

Перед выполнением производится автоматическая распаковка массивов во входной последовательности.

Все элементы во входной последовательности должны быть объектами.

Для каждого элемента входной последовательности:

1. Рассматривается каждая пара ключ-значение в элементе
2. Для каждой такой пары формируется объект с ключами `name` и `value`
3. `name` хранит строку с названием ключа из пары
4. `value` хранит значение из пары
5. Все объекты для этого элемента добавляются в выходную последовательность

Примеры

Рассмотрим JSON документ:

```
{
    "name": "Chrisjen",
    "surname": "Avasarala",
    "age": 70
}
```

Для него запрос `$.keyvalue()` вернет последовательность:

```
{
    "name": "age",
    "value": 70
},
{
    "name": "name",
    "value": "Chrisjen"
},
{
    "name": "surname",
    "value": "Avasarala"
}
```

Переменные

Функции использующие JsonPath могут передавать значения внутрь запроса, они называются переменными. Чтобы обратиться к переменной, нужно написать символ `$` и название переменной: `$variable`.

Пример

Пусть переменная `planet` равна

```
{
    "name": "Mars",
    "gravity": 0.376
}
```

Тогда запрос `strict $planet.name` даст `"Mars"`.

В отличии от многих языков программирования, JsonPath не поддерживает создание новых переменных или изменение существующих.

Общие аргументы

Все функции для работы с JSON принимают:

1. JSON значение (может быть произвольным выражением типа `Json` или `Json?`)
2. JsonPath запрос (должен быть явно указан строковым литералом; параметр запроса или другое выражение здесь использовать нельзя)
3. **(Опционально)** `PASSING` секция

Важно

Сам JsonPath запрос нельзя передать как параметр YQL-запроса, в том числе как параметр, заданный через SDK. `JsonPath` не является типом данных YQL. Поэтому ни `$path`, ни `CAST($path AS JsonPath)` нельзя использовать как второй аргумент JSON-функции.

Чтобы передать динамические значения в фиксированный JsonPath запрос, используйте секцию `PASSING`. Если приложению нужно выбирать разные пути, выбирайте шаблон запроса из заданного приложением списка разрешённых вариантов. Не подставляйте недоверенный путь в текст YQL-запроса.

PASSING секция

Позволяет передавать значения в JsonPath запрос в качестве переменных.
В качестве выражения в этой секции можно использовать параметр YQL-запроса:

```
DECLARE $increment AS Int64;

$json = CAST(@@{"catalog":{"price":10}}@@ AS Json);

SELECT JSON_VALUE(
    $json,
    "$.catalog.price + $value"
    PASSING $increment AS value
    RETURNING Int64
);
-- При $increment = 5 результат равен 15.
```

Синтаксис

```
PASSING
    <expression 1> AS <variable name 1>,
    <expression 2> AS <variable name 2>,
    ...
```

`<expression>` может быть следующих типов:

- Числа, `Date`, `DateTime` и `Timestamp` (будет произведен `CAST` в тип `Double` перед передачей в JsonPath)
- `Utf8`, `Bool` и `Json`

Имя переменной `<variable name>` можно указать несколькими способами:

- Как SQL имя, например `variable`
- В кавычках, например `"variable"`

Пример

```
JSON_VALUE(
    $json,
    "$.timestamp - $Now + $Hour"
    PASSING
        24 * 60 as Hour,
        CurrentUtcTimestamp() as "Now"
)
```

JSON_EXISTS

Функция `JSON_EXISTS` позволяет проверить, удовлетворяет ли JSON значение указанному JsonPath.

Синтаксис

```
JSON_EXISTS(
    <JSON expression>,
    <JsonPath query>,
    [<PASSING clause>]
    [{TRUE | FALSE | UNKNOWN | ERROR} ON ERROR]
)
```

Возвращаемое значение: `Bool?`

Значения по умолчанию: если `ON ERROR` секция не указана, используется `FALSE ON ERROR`

Поведение:

1. Если `<JSON expression>` это `NULL` или пустой `Json?`, то возвращается пустой `Bool?`
2. Если при выполнении JsonPath возникла ошибка, то возвращаемое значение зависит от `ON ERROR` секции:
   - `TRUE` - вернуть `True`
   - `FALSE` - вернуть `False`
   - `UNKNOWN` - вернуть пустой `Bool?`
   - `ERROR` - завершить весь запрос с ошибкой
3. Если результат выполнения JsonPath - это одно или несколько значений, возвращается `True`
4. Иначе возвращается `False`

Примеры

```
$json = CAST(@@{
    "title": "Rocinante",
    "crew": [
        "James Holden",
        "Naomi Nagata",
        "Alex Kamai",
        "Amos Burton"
    ]
}@@ as Json);

SELECT
    JSON_EXISTS($json, "$.title"), -- True
    JSON_EXISTS($json, "$.crew[*]"), -- True
    JSON_EXISTS($json, "$.nonexistent"); -- False, так как JsonPath вернет пустой результат

SELECT
    -- Ошибка JsonPath, вернет False, так как по умолчанию используется FALSE ON ERROR
    JSON_EXISTS($json, "strict $.nonexistent");

SELECT
    -- Ошибка JsonPath, весь YQL запрос завершится ошибкой
    JSON_EXISTS($json, "strict $.nonexistent" ERROR ON ERROR);
```

JSON_VALUE

Функция `JSON_VALUE` позволяет извлечь из JSON скалярное значение (все что не массив и не объект).

Синтаксис

```
JSON_VALUE(
    <JSON expression>,
    <JsonPath query>,
    [<PASSING clause>]
    [RETURNING <type>]
    [{ERROR | NULL | DEFAULT <expr>} ON EMPTY]
    [{ERROR | NULL | DEFAULT <expr>} ON ERROR]
)
```

Возвращаемое значение: `<type>?`

Значения по умолчанию:

1. Если `ON EMPTY` секция не указана, используется `NULL ON EMPTY`
2. Если `ON ERROR` секция не указана, используется `NULL ON ERROR`
3. Если `RETURNING` секция не указана, в качестве `<type>` используется `Utf8`

Поведение:

1. Если `<JSON expression>` это `NULL` или пустой `Json?`, то возвращается пустой `<type>?`
2. Если возникла ошибка, то возвращаемое значение зависит от `ON ERROR` секции:
   - `NULL` - вернуть пустой `<type>?`
   - `ERROR` - завершить весь запрос с ошибкой
   - `DEFAULT <expr>` - вернуть `<expr>`, предварительно сделав `CAST` в тип `<type>?`. Если `CAST` не удастся, весь запрос завершится с ошибкой
3. Если в результате выполнения JsonPath получился пустой результат, то возвращаемое значение зависит от `ON EMPTY` секции:
   - `NULL` - вернуть пустой `<type>?`
   - `ERROR` - завершить весь запрос с ошибкой
   - `DEFAULT <expr>` - вернуть `<expr>`, предварительно сделав `CAST` в тип `<type>?`. Если `CAST` не удастся, то поведение соответствует `ON ERROR` секции
4. Если результат выполнения JsonPath - это одно значение, то:
   - Если `RETURNING` секция не была указана, значение конвертируется в `Utf8`
   - Иначе производится `CAST` значения в `<type>`. Если `CAST` не удастся, то поведение соответствует `ON ERROR` секции. При этом значение из JSON должно "соответствовать" типу `<type>`.
5. Вернуть полученный результат

Определим соответствие JSON типов YQL типам:

- JSON Number - числовые типы, `Date`, `DateTime` и `Timestamp`
- JSON Bool - `Bool`
- JSON String - `Utf8` и `String`

Ошибками при выполнении `JSON_VALUE` считаются:

- Ошибки во время вычисления JsonPath
- Результат выполнения JsonPath - это несколько значений или не скалярное значение
- Тип полученного из JSON значения не совпадает с ожидаемым

`RETURNING` секция поддерживает типы чисел, `Date`, `DateTime`, `Timestamp`, `Utf8`, `String` и `Bool`.

Примеры

```
$json = CAST(@@{
    "friends": [
        {
            "name": "James Holden",
            "age": 35
        },
        {
            "name": "Naomi Nagata",
            "age": 30
        }
    ]
}@@ as Json);

SELECT
    JSON_VALUE($json, "$.friends[0].age"), -- "35" (тип Utf8?)
    JSON_VALUE($json, "$.friends[0].age" RETURNING Uint64), -- 35 (тип Uint64?)
    JSON_VALUE($json, "$.friends[0].age" RETURNING Utf8); -- пустой Utf8?, так как произошла ошибка. Тип JSON
значения Number не соответствует строковому типу Utf8

SELECT
    -- "empty" (тип String?)
    JSON_VALUE(
        $json,
        "$.friends[50].name"
        RETURNING String
        DEFAULT "empty" ON EMPTY
    );

SELECT
    -- 20 (тип Uint64?). Результат выполнения JsonPath пустой, но значение
    -- по умолчанию из секции ON EMPTY не может быть приведено к Uint64.
    -- Поэтому используется значение из ON ERROR
    JSON_VALUE(
        $json,
        "$.friends[50].age"
        RETURNING Uint64
        DEFAULT -1 ON EMPTY
        DEFAULT 20 ON ERROR
    );
```

JSON_QUERY

Функция `JSON_QUERY` позволяет извлекать из JSON массивы и объекты.

Синтаксис

```
JSON_QUERY(
    <JSON expression>,
    <JsonPath query>,
    [<PASSING clause>]
    [WITHOUT [ARRAY] | WITH [CONDITIONAL | UNCONDITIONAL] [ARRAY] WRAPPER]
    [{ERROR | NULL | EMPTY ARRAY | EMPTY OBJECT} ON EMPTY]
    [{ERROR | NULL | EMPTY ARRAY | EMPTY OBJECT} ON ERROR]
)
```

Возвращаемое значение `Json?`

Значения по умолчанию

1. Если `ON EMPTY` секция не указана, используется `NULL ON EMPTY`
2. Если `ON ERROR` секция не указана, используется `NULL ON ERROR`
3. Если `WRAPPER` секция не указана, используется `WITHOUT WRAPPER`
4. Если указана секция `WITH WRAPPER`, но не указано `CONDITIONAL` или `UNCONDITIONAL`, используется `UNCONDITIONAL`

Поведение:

> **Примечание**
>
> Указывать секции `WITH ... WRAPPER` и `ON EMPTY` одновременно нельзя

1. Если `<JSON expression>` это `NULL` или пустой `Json?`, то возвращается пустой `Json?`
2. Если указана `WRAPPER` секция, то:
   - `WITHOUT WRAPPER` или `WITHOUT ARRAY WRAPPER` - никак не преобразовывать результат выполнения JsonPath
   - `WITH UNCONDITIONAL WRAPPER` или `WITH UNCONDITIONAL ARRAY WRAPPER` - обернуть результат выполнения JsonPath в массив
   - `WITH CONDITIONAL WRAPPER` или `WITH CONDITIONAL ARRAY WRAPPER` - обернуть результат выполнения JsonPath в массив если он не является единственным массивом или объектом
3. Если в результате выполнения JsonPath получился пустой результат, то возвращаемое значение зависит от `ON EMPTY` секции:
   - `NULL` - вернуть пустой `Json?`
   - `ERROR` - завершить весь запрос с ошибкой

- `EMPTY ARRAY` - вернуть пустой JSON массив, `[]`
- `EMPTY OBJECT` - вернуть пустой JSON объект, `{}`

4. Если возникла ошибка, то возвращаемое значение зависит от `ON ERROR` секции:
   - `NULL` - вернуть пустой `Json?`
   - `ERROR` - завершить весь запрос с ошибкой
   - `EMPTY ARRAY` - вернуть пустой JSON массив, `[]`
   - `EMPTY OBJECT` - вернуть пустой JSON объект, `{}`
5. Вернуть полученный результат

Ошибками при выполнении `JSON_QUERY` считаются:

- Ошибки во время вычисления JsonPath
- Результат выполнения JsonPath - это несколько значений (даже после применения секции `WRAPPER`) или скалярное значение

Примеры

```
$json = CAST(@@{
    "friends": [
        {
            "name": "James Holden",
            "age": 35
        },
        {
            "name": "Naomi Nagata",
            "age": 30
        }
    ]
}@@ as Json);

SELECT
    JSON_QUERY($json, "$.friends[0]"); -- {"name": "James Holden", "age": 35}

SELECT
    JSON_QUERY($json, "$.friends.name" WITH UNCONDITIONAL WRAPPER); -- ["James Holden", "Naomi Nagata"]

SELECT
    JSON_QUERY($json, "$.friends[0]" WITH CONDITIONAL WRAPPER), -- {"name": "James Holden", "age": 35}
    JSON_QUERY($json, "$.friends.name" WITH CONDITIONAL WRAPPER); -- ["James Holden", "Naomi Nagata"]
```

Смотрите также

- Доступ к значениям в JSON с помощью YQL
- Изменение JSON с помощью YQL

Функции встроенных C++ библиотек

Многие прикладные функции, которые с одной стороны слишком специфичны, чтобы стать частью ядра YQL, а с другой — могут быть полезны широкому кругу пользователей, доступны через встроенные C++ библиотеки.

- DateTime
- Digest
- Histogram
- Hyperscan
- Ip
- Knn
- Math
- Pcre
- Pire
- Re2
- Roaring
- String
- Unicode
- Url
- Yson

DateTime

Модуль DateTime предоставляет набор функций для работы с датами и временем в YDB. Они поддерживают все доступные типы данных временных меток и интервалов как с базовым диапазоном допустимых значений, так и с расширенным. Кроме того, эти функции используют собственное внутреннее представление значений, в котором каждое значение разбивается на отдельные компоненты. Это представление описано ниже.

Самые простые операции с временными метками и интервалами можно выполнять и без использования модуля DateTime. Например, преобразование между строками и временными метками по ISO 8601 доступно через `CAST`, изменить точность временной метки также можно с помощью `CAST`, а между временными метками и интервалами работают арифметические операторы. Модуль DateTime предназначен для задач, где этих базовых операций недостаточно и требуются более сложные преобразования.

Внутренний формат представления

Большинство функций модуля DateTime оперируют не исходными временными метками и интервалами, а внутренним представлением, основанным на специальном типе данных ресурс. Оно представляет собой указатель на структуру данных в памяти, где компоненты времени хранятся раздельно. Работа с ресурсом происходит исключительно через функции модуля DateTime. Типовой сценарий работы с этим модулем (примеры см. в конце статьи) следующий:

1. Вызвать функцию `Split`, чтобы получить из исходной временной метки внутреннее представление, либо `Parse...`, чтобы получить его из строки.
2. При необходимости вызвать функции, модифицирующие это внутреннее представление нужным образом, например `Shift...`, `StartOf...` или `Update...`.
3. Вызвать одну из функций для создания нового значения на основе внутреннего представления, которое уже можно сохранять в таблицы или возвращать из запроса:
   - `Make...`, чтобы создать из внутреннего представления новую временную метку (не обязательно той же точности, что и исходная);
   - `Format`, чтобы получить строковое представление;
   - `Get...`, чтобы получить отдельные компоненты.

В YDB поддерживается два диапазона допустимых значений временных меток: базовый и расширенный. В модуле DateTime им соответствуют два отдельных внутренних представления: `Resource<TM>` для базовых типов данных и `Resource<TM64>` для типов данных с расширенным диапазоном допустимых значений.

`Resource<TM>` и `Resource<TM64>` содержат следующие компоненты дат:

- Year (для `Resource<TM>` 12 бит, беззнаковый; для `Resource<TM64>` 19 бит, знаковый);
- Month (4 бита);
- Day (5 бит);
- Hour (5 бит);
- Minute (6 бит);
- Second (6 бит);
- Microsecond (20 бит);
- TimezoneId (16 бит);
- DayOfYear (9 бит) — день от начала года;
- WeekOfYear (6 бит) — неделя от начала года, 1 января всегда относится к первой неделе;
- WeekOfYearIso8601 (6 бит) — неделя года согласно ISO 8601 (первой неделей считается та, в которой 4 января);
- DayOfWeek (3 бита) — день недели.

Если таймзона не GMT, то в компонентах хранится локальное время в соответствующей таймзоне.

Split

Преобразование из простого типа YDB во внутреннее представление модуля DateTime. Преобразование всегда выполняется успешно, при условии что на вход функции подаётся требуемое значение.

Список функций

-

```
DateTime::Split(Date/TzDate/Datetime/TzDatetime/Timestamp/TzTimestamp{Flags:AutoMap})
-> Resource<TM>;
```

-

```
DateTime::Split(Date32/TzDate32/Datetime64/TzDatetime64/Timestamp64/TzTimestamp64{Flags:AutoMap})
-> Resource<TM64>.
```

Функции, принимающие на вход `Resource<TM>` или `Resource<TM64>`, могут быть вызваны непосредственно от простого типа даты/времени. В этом случае будет сделано неявное преобразование через вызов соответствующей функции `Split`.

Make...

Сборка простого типа YDB из внутреннего представления модуля DateTime. Преобразование всегда выполняется успешно, если на вход функции подаётся требуемое значение.

Список функций

- `DateTime::MakeDate(Resource<TM>{Flags:AutoMap}) -> Date`
- `DateTime::MakeDate32(Resource<TM64>{Flags:AutoMap}) -> Date32`;
- `DateTime::MakeTzDate32(Resource<TM64>{Flags:AutoMap}) -> TzDate32`;
- `DateTime::MakeDatetime(Resource<TM>{Flags:AutoMap}) -> Datetime`;
- `DateTime::MakeTzDatetime(Resource<TM>{Flags:AutoMap}) -> TzDatetime`;
- `DateTime::MakeDatetime64(Resource<TM64>{Flags:AutoMap}) -> Datetime64`;

- `DateTime::MakeTzDatetime64(Resource<TM64>{Flags:AutoMap}) -> TzDatetime64`;
- `DateTime::MakeTimestamp(Resource<TM>{Flags:AutoMap}) -> Timestamp`;
- `DateTime::MakeTzTimestamp(Resource<TM>{Flags:AutoMap}) -> TzTimestamp`;
- `DateTime::MakeTimestamp64(Resource<TM64>{Flags:AutoMap}) -> Timestamp64`;
- `DateTime::MakeTzTimestamp64(Resource<TM64>{Flags:AutoMap}) -> TzTimestamp64`.

Примеры

```
SELECT
    DateTime::MakeTimestamp(DateTime::Split(Datetime("2019-01-01T15:30:00Z"))),
      -- 2019-01-01T15:30:00.000000Z
    DateTime::MakeDate(Datetime("2019-01-01T15:30:00Z")),
      -- 2019-01-01
    DateTime::MakeTimestamp(DateTime::Split(TzDatetime("2019-01-01T00:00:00,Europe/Moscow"))),
      -- 2018-12-31T21:00:00Z (конвертация в UTC)
    DateTime::MakeDate(TzDatetime("2019-01-01T12:00:00,GMT"))
      -- 2019-01-01 (Datetime -> Date с неявным Split)
```

Get...

Получение компоненты внутреннего представления.

Список функций

- `DateTime::GetYear(Resource<TM>{Flags:AutoMap}) -> Uint16`;
- `DateTime::GetYear(Resource<TM64>{Flags:AutoMap}) -> Int32`;
- `DateTime::GetDayOfYear(Resource<TM>/Resource<TM64>{Flags:AutoMap}) -> Uint16`;
- `DateTime::GetMonth(Resource<TM>/Resource<TM64>{Flags:AutoMap}) -> Uint8`;
- `DateTime::GetMonthName(Resource<TM>/Resource<TM64>{Flags:AutoMap}) -> String`;
- `DateTime::GetWeekOfYear(Resource<TM>/Resource<TM64>{Flags:AutoMap}) -> Uint8`;
- `DateTime::GetWeekOfYearIso8601(Resource<TM>/Resource<TM64>{Flags:AutoMap}) -> Uint8`;
- `DateTime::GetDayOfMonth(Resource<TM>/Resource<TM64>{Flags:AutoMap}) -> Uint8`;
- `DateTime::GetDayOfWeek(Resource<TM>/Resource<TM64>{Flags:AutoMap}) -> Uint8`;
- `DateTime::GetDayOfWeekName(Resource<TM>/Resource<TM64>{Flags:AutoMap}) -> String`;
- `DateTime::GetHour(Resource<TM>/Resource<TM64>{Flags:AutoMap}) -> Uint8`;
- `DateTime::GetMinute(Resource<TM>/Resource<TM64>{Flags:AutoMap}) -> Uint8`;
- `DateTime::GetSecond(Resource<TM>/Resource<TM64>{Flags:AutoMap}) -> Uint8`;
- `DateTime::GetMillisecondOfSecond(Resource<TM>/Resource<TM64>{Flags:AutoMap}) -> Uint32`;
- `DateTime::GetMicrosecondOfSecond(Resource<TM>/Resource<TM64>{Flags:AutoMap}) -> Uint32`;
- `DateTime::GetTimezoneId(Resource<TM>/Resource<TM64>{Flags:AutoMap}) -> Uint16`;
- `DateTime::GetTimezoneName(Resource<TM>/Resource<TM64>{Flags:AutoMap}) -> String`.

Примеры

```
$tm = DateTime::Split(TzDatetime("2019-01-09T00:00:00,Europe/Moscow"));

SELECT
    DateTime::GetDayOfMonth($tm) as Day, -- 9
    DateTime::GetMonthName($tm) as Month, -- "January"
    DateTime::GetYear($tm) as Year, -- 2019
    DateTime::GetTimezoneName($tm) as TzName, -- "Europe/Moscow"
    DateTime::GetDayOfWeekName($tm) as WeekDay; -- "Wednesday"
```

Update

Обновление одной или нескольких компонент во внутреннем представлении. Возвращает либо обновлённую копию, либо `NULL`, если после обновления получается некорректная дата или возникают другие противоречия.

Список функций

```
DateTime::Update(Resource<TM>{Flags:AutoMap}, [ Year:Uint16?, Month:Uint8?, Day:Uint8?, Hour:Uint8?, Minute:U
int8?, Second:Uint8?, Microsecond:Uint32?, Timezone:String? ]) -> Resource<TM>?
DateTime::Update(Resource<TM64>{Flags:AutoMap}, [ Year:Int32?, Month:Uint8?, Day:Uint8?, Hour:Uint8?, Minute:
Uint8?, Second:Uint8?, Microsecond:Uint32?, Timezone:String? ]) -> Resource<TM64>?
```

Примеры

```
$tm = DateTime::Split(Timestamp("2019-01-01T01:02:03.456789Z"));

SELECT
    DateTime::MakeDate(DateTime::Update($tm, 2012)), -- 2012-01-01
    DateTime::MakeDate(DateTime::Update($tm, 2000, 6, 6)), -- 2000-06-06
    DateTime::MakeDate(DateTime::Update($tm, NULL, 2, 30)), -- NULL (30 февраля)
    DateTime::MakeDatetime(DateTime::Update($tm, NULL, NULL, 31)), -- 2019-01-31T01:02:03Z
    DateTime::MakeDatetime(DateTime::Update($tm, 15 as Hour, 30 as Minute)), -- 2019-01-01T15:30:03Z
    DateTime::MakeTimestamp(DateTime::Update($tm, 999999 as Microsecond)), -- 2019-01-01T01:02:03.999999Z
    DateTime::MakeTimestamp(DateTime::Update($tm, "Europe/Moscow" as Timezone)), -- 2018-12-31T22:02:03.45678
9Z (конвертация в UTC)
    DateTime::MakeTzTimestamp(DateTime::Update($tm, "Europe/Moscow" as Timezone)); -- 2019-01-01T01:02:03.456
789,Europe/Moscow
```

From...

Получение Timestamp из количества секунд/миллисекунд/микросекунд от начала эпохи в UTC. При выходе за границы Timestamp возвращается NULL.

Список функций

- `DateTime::FromSeconds(Uint32{Flags:AutoMap}) -> Timestamp`;
- `DateTime::FromSeconds64(Int64{Flags:AutoMap}) -> Timestamp64`;
- `DateTime::FromMilliseconds(Uint64{Flags:AutoMap}) -> Timestamp`;
- `DateTime::FromMilliseconds64(Int64{Flags:AutoMap}) -> Timestamp64`;
- `DateTime::FromMicroseconds(Uint64{Flags:AutoMap}) -> Timestamp`;
- `DateTime::FromMicroseconds64(Int64{Flags:AutoMap}) -> Timestamp64`.

To...

Получение количества секунд/миллисекунд/микросекунд от начала эпохи в UTC из простого типа.

Список функций

- `DateTime::ToSeconds(Date/Datetime/Timestamp/TzDate/TzDatetime/TzTimestamp{Flags:AutoMap}) -> Uint32`;
-

```
DateTime::ToSeconds(
Date32/Datetime64/Timestamp64/TzDate32/TzDatetime64/TzTimestamp64{Flags:AutoMap})
-> Int64;
```

-

```
DateTime::ToMilliseconds(Date/Datetime/Timestamp/TzDate/TzDatetime/TzTimestamp{Flags:AutoMap})
-> Uint64;
```

-

```
DateTime::ToMilliseconds(
Date32/Datetime64/Timestamp64/TzDate32/TzDatetime64/TzTimestamp64{Flags:AutoMap})
-> Int64;
```

-

```
DateTime::ToMicroseconds(Date/Datetime/Timestamp/TzDate/TzDatetime/TzTimestamp{Flags:AutoMap})
-> Uint64;
```

-

```
DateTime::ToMicroseconds(
Date32/Datetime64/Timestamp64/TzDate32/TzDatetime64/TzTimestamp64{Flags:AutoMap})
-> Int64.
```

Примеры

```
SELECT
    DateTime::FromSeconds(1546304523), -- 2019-01-01T01:02:03.000000Z
    DateTime::ToMicroseconds(Timestamp("2019-01-01T01:02:03.456789Z")); -- 1546304523456789
```

Interval...

Преобразования между `Interval` и различными единицами измерения времени.

Список функций

- `DateTime::ToDays(Interval{Flags:AutoMap}) -> Int32`;
- `DateTime::ToDays(Interval64{Flags:AutoMap}) -> Int32`;
- `DateTime::ToHours(Interval{Flags:AutoMap}) -> Int32`;
- `DateTime::ToHours(Interval64{Flags:AutoMap}) -> Int64`;
- `DateTime::ToMinutes(Interval{Flags:AutoMap}) -> Int32`;
- `DateTime::ToMinutes(Interval64{Flags:AutoMap}) -> Int64`;
- `DateTime::ToSeconds(Interval{Flags:AutoMap}) -> Int64`;
- `DateTime::ToSeconds(Interval64{Flags:AutoMap}) -> Int64`;
- `DateTime::ToMilliseconds(Interval{Flags:AutoMap}) -> Int64`;
- `DateTime::ToMilliseconds(Interval64{Flags:AutoMap}) -> Int64`;
- `DateTime::ToMicroseconds(Interval{Flags:AutoMap}) -> Int64`;
- `DateTime::ToMicroseconds(Interval64{Flags:AutoMap}) -> Int64`;
- `DateTime::IntervalFromDays(Int32{Flags:AutoMap}) -> Interval`;
- `DateTime::Interval64FromDays(Int32{Flags:AutoMap}) -> Interval64`;
- `DateTime::IntervalFromHours(Int32{Flags:AutoMap}) -> Interval`;
- `DateTime::Interval64FromHours(Int64{Flags:AutoMap}) -> Interval64`;
- `DateTime::IntervalFromMinutes(Int32{Flags:AutoMap}) -> Interval`;

- `DateTime::Interval64FromMinutes(Int64{Flags:AutoMap}) -> Interval64`;
- `DateTime::IntervalFromSeconds(Int64{Flags:AutoMap}) -> Interval`;
- `DateTime::Interval64FromSeconds(Int64{Flags:AutoMap}) -> Interval64`;
- `DateTime::IntervalFromMilliseconds(Int64{Flags:AutoMap}) -> Interval`;
- `DateTime::Interval64FromMilliseconds(Int64{Flags:AutoMap}) -> Interval64`;
- `DateTime::IntervalFromMicroseconds(Int64{Flags:AutoMap}) -> Interval`;
- `DateTime::Interval64FromMicroseconds(Int64{Flags:AutoMap}) -> Interval64`.

`AddTimezone` никак не влияет на вывод `ToSeconds()`, поскольку `ToSeconds()` всегда возвращают время в таймзоне GMT.

Interval также можно создавать из строкового литерала в формате ISO 8601.

> **Важно**
>
> На данный момент поддерживаются только литералы для детерминированных интервалов. Все размерности больше недели — `M` (месяц) и `Y` (год) — не являются фиксированными, поскольку их продолжительность определяется только внешним контекстом, календарём.

Примеры

```
SELECT
    DateTime::ToDays(Interval("PT3000M")), -- 2
    DateTime::IntervalFromSeconds(1000000), -- 11 days 13 hours 46 minutes 40 seconds
    DateTime::ToDays(CAST('2018-01-01' as date) - CAST('2017-12-31' as date)); --1
```

StartOf... / EndOf... / TimeOfDay

Получить начало (конец) периода, содержащего дату/время. При некорректном результате возвращается NULL. Если таймзона не GMT, то начало (конец) периода будет в указанной временной зоне.

Список функций

- `DateTime::StartOfYear(Resource<TM>{Flags:AutoMap}) -> Resource<TM>?`;
- `DateTime::StartOfYear(Resource<TM64>{Flags:AutoMap}) -> Resource<TM64>?`;
- `DateTime::EndOfYear(Resource<TM>{Flags:AutoMap}) -> Resource<TM>?`;
- `DateTime::EndOfYear(Resource<TM64>{Flags:AutoMap}) -> Resource<TM64>?`;
- `DateTime::StartOfQuarter(Resource<TM>{Flags:AutoMap}) -> Resource<TM>?`;
- `DateTime::StartOfQuarter(Resource<TM64>{Flags:AutoMap}) -> Resource<TM64>?`;
- `DateTime::EndOfQuarter(Resource<TM>{Flags:AutoMap}) -> Resource<TM>?`;
- `DateTime::EndOfQuarter(Resource<TM64>{Flags:AutoMap}) -> Resource<TM64>?`;
- `DateTime::StartOfMonth(Resource<TM>{Flags:AutoMap}) -> Resource<TM>?`;
- `DateTime::StartOfMonth(Resource<TM64>{Flags:AutoMap}) -> Resource<TM64>?`;
- `DateTime::EndOfMonth(Resource<TM>{Flags:AutoMap}) -> Resource<TM>?`;
- `DateTime::EndOfMonth(Resource<TM64>{Flags:AutoMap}) -> Resource<TM64>?`;
- `DateTime::StartOfWeek(Resource<TM>{Flags:AutoMap}) -> Resource<TM>?`;
- `DateTime::StartOfWeek(Resource<TM64>{Flags:AutoMap}) -> Resource<TM64>?`;
- `DateTime::EndOfWeek(Resource<TM>{Flags:AutoMap}) -> Resource<TM>?`;
- `DateTime::EndOfWeek(Resource<TM64>{Flags:AutoMap}) -> Resource<TM64>?`;
- `DateTime::StartOfDay(Resource<TM>{Flags:AutoMap}) -> Resource<TM>?`;
- `DateTime::StartOfDay(Resource<TM64>{Flags:AutoMap}) -> Resource<TM64>?`;
- `DateTime::EndOfDay(Resource<TM>{Flags:AutoMap}) -> Resource<TM>?`;
- `DateTime::EndOfDay(Resource<TM64>{Flags:AutoMap}) -> Resource<TM64>?`;
- `DateTime::StartOf(Resource<TM>{Flags:AutoMap}, Interval{Flags:AutoMap}) -> Resource<TM>?`;
- `DateTime::StartOf(Resource<TM64>{Flags:AutoMap}, Interval64{Flags:AutoMap}) -> Resource<TM64>?`;
- `DateTime::EndOf(Resource<TM>{Flags:AutoMap}, Interval{Flags:AutoMap}) -> Resource<TM>?`;
- `DateTime::EndOf(Resource<TM64>{Flags:AutoMap}, Interval64{Flags:AutoMap}) -> Resource<TM64>?`.

Функции `StartOf` / `EndOf` предназначены для группировки в пределах суток по произвольному периоду. Результат отличается от входного значения только компонентами времени. Период более суток трактуется как сутки (эквивалентно `StartOfDay` / `EndOfDay`). Если в сутках не содержится целого числа периодов, производится округление к ближайшему времени от начала суток, кратному указанному периоду. При нулевом интервале выход совпадает со входом. Отрицательный интервал трактуется как положительный.

Функции `EndOf...` предназначены для получения последнего момента времени, находящегося в том же выбранном интервале, что и заданный.

Поведение функций с периодами больше дня отличается от поведения одноимённых функций в старой библиотеке. Компоненты времени всегда обнуляются (это логично, поскольку эти функции в основном используются для группировки по периоду). Отдельно существует возможность выделить время в пределах суток:

- `DateTime::TimeOfDay(Resource<TM>{Flags:AutoMap}) -> Interval`;
- `DateTime::TimeOfDay(Resource<TM64>{Flags:AutoMap}) -> Interval64`.

Примеры

```
SELECT
    DateTime::MakeDate(DateTime::StartOfYear(Date("2019-06-06"))),
      -- 2019-01-01 (неявный Split здесь и дальше)
    DateTime::MakeDatetime(DateTime::StartOfQuarter(Datetime("2019-06-06T01:02:03Z"))),
```

```
    -- 2019-04-01T00:00:00Z (компоненты времени обнулены)
  DateTime::MakeDate(DateTime::StartOfMonth(Timestamp("2019-06-06T01:02:03.456789Z"))),
    -- 2019-06-01
  DateTime::MakeDate(DateTime::StartOfWeek(Date("1970-01-01"))),
    -- NULL (начало эпохи - четверг, начало недели - 1969-12-29, выход за границы)
  DateTime::MakeTimestamp(DateTime::StartOfWeek(Date("2019-01-01"))),
    -- 2018-12-31T00:00:00Z
  DateTime::MakeDatetime(DateTime::StartOfDay(Datetime("2019-06-06T01:02:03Z"))),
    -- 2019-06-06T00:00:00Z
  DateTime::MakeTzDatetime(DateTime::StartOfDay(TzDatetime("1970-01-01T05:00:00,Europe/Moscow"))),
    -- NULL (в GMT выход за эпоху)
  DateTime::MakeTzTimestamp(DateTime::StartOfDay(TzTimestamp("1970-01-02T05:00:00.000000,Europe/Moscow"))),
    -- 1970-01-02T00:00:00,Europe/Moscow (начало дня по Москве)
  DateTime::MakeDatetime(DateTime::StartOf(Datetime("2019-06-06T23:45:00Z"), Interval("PT7H"))),
    -- 2019-06-06T21:00:00Z
  DateTime::MakeDatetime(DateTime::StartOf(Datetime("2019-06-06T23:45:00Z"), Interval("PT20M"))),
    -- 2019-06-06T23:40:00Z
  DateTime::TimeOfDay(Timestamp("2019-02-14T01:02:03.456789Z"));
    -- 1 hour 2 minutes 3 seconds 456789 microseconds
```

Shift...

Прибавить/вычесть заданное количество единиц к компоненте во внутреннем представлении и обновить остальные поля. Возвращает либо обновлённую копию, либо NULL, если после обновления получается некорректная дата или возникают другие противоречия.

Список функций

- `DateTime::ShiftYears(Resource<TM>{Flags:AutoMap}, Int32) -> Resource<TM>?`;
- `DateTime::ShiftYears(Resource<TM64>{Flags:AutoMap}, Int32) -> Resource<TM64>?`;
- `DateTime::ShiftQuarters(Resource<TM>{Flags:AutoMap}, Int32) -> Resource<TM>?`;
- `DateTime::ShiftQuarters(Resource<TM64>{Flags:AutoMap}, Int32) -> Resource<TM64>?`;
- `DateTime::ShiftMonths(Resource<TM>{Flags:AutoMap}, Int32) -> Resource<TM>?`;
- `DateTime::ShiftMonths(Resource<TM64>{Flags:AutoMap}, Int32) -> Resource<TM64>?`.

Если в результате номер дня в месяце превышает максимально возможный, то в поле `Day` будет записан последний день месяца,
время при этом не изменится (см. примеры).

Примеры

```
$tm1 = DateTime::Split(Datetime("2019-01-31T01:01:01Z"));
$tm2 = DateTime::Split(TzDatetime("2049-05-20T12:34:50,Europe/Moscow"));

SELECT
    DateTime::MakeDate(DateTime::ShiftYears($tm1, 10)), -- 2029-01-31T01:01:01
    DateTime::MakeDate(DateTime::ShiftYears($tm2, -10000)), -- NULL (выход за границы)
    DateTime::MakeDate(DateTime::ShiftQuarters($tm2, 0)), -- 2049-05-20T12:34:50,Europe/Moscow
    DateTime::MakeDate(DateTime::ShiftQuarters($tm1, -3)), -- 2018-04-30T01:01:01
    DateTime::MakeDate(DateTime::ShiftMonths($tm1, 1)), -- 2019-02-28T01:01:01
    DateTime::MakeDate(DateTime::ShiftMonths($tm1, -35)), -- 2016-02-29T01:01:01
```

Format

Получить строковое представление момента времени, используя произвольную строку форматирования.

Список функций

-

```
DateTime::Format(String, alwaysWriteFractionalSeconds:Bool?)
-> (Resource<TM64>{Flags:AutoMap}) -> String
```

Поддерживается множество спецификаторов для строки форматирования:

- `%%` - символ %;
- `%Y` — год, 1–6 цифр и знак для дат до н.э.;
- `%m` — месяц, 2 цифры;
- `%d` — день, 2 цифры;
- `%H` - час 2 цифры;
- `%M` - минуты 2 цифры;
- `%S` — секунды, 2 цифры или `XX.XXXXXX`, если есть микросекунды (и только если флаг `alwaysWriteFractionalSeconds` не установлен в `True`);
- `%z` — `+hhmm` или `-hhmm`;
- `%Z` - IANA имя таймзоны;
- `%b` - короткое трехбуквенное английское название месяца (Jan);
- `%B` - полное английское название месяца (January).

Все остальные символы строки форматирования переносятся без изменений.

Примеры

```
$format = DateTime::Format("%Y-%m-%d %H:%M:%S %Z");

SELECT
```

```
    $format(DateTime::Split(TzDatetime("2019-01-01T01:02:03,Europe/Moscow")));
      -- "2019-01-01 01:02:03 Europe/Moscow"
```

Parse/Parse64

Парсит строку во внутреннее представление с использованием произвольной строки форматирования. Для незаполненных полей используются значения по умолчанию. При возникновении ошибок возвращается `NULL`.

Список функций

- `DateTime::Parse(String) -> (String{Flags:AutoMap}) -> Resource<TM>?`;
- `DateTime::Parse64(String) -> (String{Flags:AutoMap}) -> Resource<TM64>?`.

Реализованные спецификаторы:

- `%%` - символ %;
- `%Y` — год, 4 цифры — при использовании `Parse`, или 1–6 цифр и знак для дат до н.э. — при использовании `Parse64`;
- `%m` — месяц, 2 цифры;
- `%d` — день, 2 цифры;
- `%H` — час, 2 цифры;
- `%M` — минуты, 2 цифры;
- `%S` — секунды, могут включать микросекунды в форматах от `XX` до `XX.XXXXXX`;
- `%Z` — имя таймзоны в формате IANA (например, GMT);
- `%b` — короткое трёхбуквенное регистронезависимое английское название месяца (например, Jan).
- `%B` - полное регистронезависимое английское название месяца (January).

Примеры

```
$parse1 = DateTime::Parse("%H:%M:%S");
$parse2 = DateTime::Parse("%S");
$parse3 = DateTime::Parse("%m/%d/%Y");
$parse4 = DateTime::Parse("%Z");
$parse5 = DateTime::Parse64("%m/%d/%Y");

SELECT
    DateTime::MakeDatetime($parse1("01:02:03")), -- 1970-01-01T01:02:03Z
    DateTime::MakeTimestamp($parse2("12.3456")), -- 1970-01-01T00:00:12.345600Z
    DateTime::MakeTimestamp($parse3("02/30/2000")), -- NULL (Feb 30)
    DateTime::MakeTimestamp($parse4("Canada/Central")), -- 1970-01-01T06:00:00Z (конвертация в UTC)
    DateTime::MakeTimestamp64($parse5("02/10/1931")), -- 1931-02-10T00:00:00Z (конвертация в UTC)
```

Parse конкретных форматов

Для распространённых форматов есть врапперы вокруг соответствующих методов util. Можно получить только TM с компонентами в UTC таймзоне.

Список функций

- `DateTime::ParseRfc822(String{Flags:AutoMap}) -> Resource<TM>?`;
- `DateTime::ParseIso8601(String{Flags:AutoMap}) -> Resource<TM>?`;
- `DateTime::ParseHttp(String{Flags:AutoMap}) -> Resource<TM>?`;
- `DateTime::ParseX509(String{Flags:AutoMap}) -> Resource<TM>?`.

Примеры

```
SELECT
    DateTime::MakeTimestamp(DateTime::ParseRfc822("Fri, 4 Mar 2005 19:34:45 EST")),
      -- 2005-03-05T00:34:45Z
    DateTime::MakeTimestamp(DateTime::ParseIso8601("2009-02-14T02:31:30+0300")),
      -- 2009-02-13T23:31:30Z
    DateTime::MakeTimestamp(DateTime::ParseHttp("Sunday, 06-Nov-94 08:49:37 GMT")),
      -- 1994-11-06T08:49:37Z
    DateTime::MakeTimestamp(DateTime::ParseX509("20091014165533Z"))
      -- 2009-10-14T16:55:33Z
```

Типовые сценарии

Преобразования между строками и секундами

Преобразование строковой даты (в таймзоне Москвы) в секунды (в таймзоне GMT):

```
$datetime_parse = DateTime::Parse("%Y-%m-%d %H:%M:%S");
$datetime_parse_tz = DateTime::Parse("%Y-%m-%d %H:%M:%S %Z");

SELECT
    DateTime::ToSeconds(TzDatetime("2019-09-16T00:00:00,Europe/Moscow")) AS md_us1, -- 1568581200
    DateTime::ToSeconds(DateTime::MakeDatetime($datetime_parse_tz("2019-09-16 00:00:00" || " Europe/Mosco
w"))), -- 1568581200
    DateTime::ToSeconds(DateTime::MakeDatetime(DateTime::Update($datetime_parse("2019-09-16 00:00:00"), "Euro
pe/Moscow" AS Timezone))), -- 1568581200
    -- НЕПРАВИЛЬНО (Date импортирует время как GMT, а AddTimezone никак не влияет на ToSeconds, которая всегд
а возвращает время в таймзоне GMT)
    DateTime::ToSeconds(AddTimezone(Date("2019-09-16"), 'Europe/Moscow')) AS md_us2, -- 1568592000
```

Преобразование строковой даты (в таймзоне Москвы) в секунды (в таймзоне Москвы). Поскольку `DateTime::ToSeconds()` экспортирует только GMT, придется временно забыть о таймзонах и работать только в GMT (выглядит это так, как будто временно мы считаем, что в Москве таймзона GMT):

```
$date_parse = DateTime::Parse("%Y-%m-%d");
$datetime_parse = DateTime::Parse("%Y-%m-%d %H:%M:%S");
$datetime_parse_tz = DateTime::Parse("%Y-%m-%d %H:%M:%S %Z");

SELECT
    DateTime::ToSeconds(Datetime("2019-09-16T00:00:00Z")) AS md_ms1, -- 1568592000
    DateTime::ToSeconds(Date("2019-09-16")) AS md_ms2, -- 1568592000
    DateTime::ToSeconds(DateTime::MakeDatetime($date_parse("2019-09-16"))) AS md_ms3, -- 1568592000
    DateTime::ToSeconds(DateTime::MakeDatetime($datetime_parse("2019-09-16 00:00:00"))) AS md_ms4, -- 1568592
000
    DateTime::ToSeconds(DateTime::MakeDatetime($datetime_parse_tz("2019-09-16 00:00:00 GMT"))) AS md_ms5, --
1568592000

    -- НЕПРАВИЛЬНО (импортирует время в таймзоне Москвы, а RemoveTimezone никак не влияет на ToSeconds)
    DateTime::ToSeconds(RemoveTimezone(TzDatetime("2019-09-16T00:00:00,Europe/Moscow"))) AS md_ms6, -- 156858
1200
    DateTime::ToSeconds(DateTime::MakeDatetime($datetime_parse_tz("2019-09-16 00:00:00 Europe/Moscow"))) AS m
d_ms7 -- 1568581200
```

Преобразование секунд (в таймзоне GMT) в строковую дату (в таймзоне Москвы):

```
$date_format = DateTime::Format("%Y-%m-%d %H:%M:%S %Z");
SELECT
    $date_format(AddTimezone(DateTime::FromSeconds(1568592000), 'Europe/Moscow')) -- "2019-09-16 03:00:00 Eur
ope/Moscow"
```

Преобразование секунд (в таймзоне Москвы) в строковую дату (в таймзоне Москвы). Здесь таймзона `%Z` выводится для справки - обычно выводить ее не нужно, потому что она будет равна "GMT" и может сбить с толку.

```
$date_format = DateTime::Format("%Y-%m-%d %H:%M:%S %Z");
SELECT
    $date_format(DateTime::FromSeconds(1568592000)) -- "2019-09-16 00:00:00 GMT"
```

Преобразование секунд (в таймзоне GMT) в трехбуквенное название дня недели (в таймзоне Москвы):

```
SELECT
    SUBSTRING(DateTime::GetDayOfWeekName(AddTimezone(DateTime::FromSeconds(1568581200), "Europe/Moscow")), 0,
3) -- "Mon"
```

Форматирование даты и времени

Обычно для форматирования времени используется отдельное именованное выражение, но можно обойтись и без него:

```
$date_format = DateTime::Format("%Y-%m-%d %H:%M:%S %Z");

SELECT

   -- Вариант с именованным выражением

   $date_format(AddTimezone(DateTime::FromSeconds(1568592000), 'Europe/Moscow')),

   -- Вариант без именованного выражения

   DateTime::Format("%Y-%m-%d %H:%M:%S %Z")
       (AddTimezone(DateTime::FromSeconds(1568592000), 'Europe/Moscow'))
;
```

Преобразование типов

Так можно преобразовывать только константы:

```
SELECT
    TzDatetime("2019-09-16T00:00:00,Europe/Moscow"), -- 2019-09-16T00:00:00,Europe/Moscow
    Date("2019-09-16") -- 2019-09-16
```

А так - константу, именованное выражение или поле таблицы:

```
SELECT
CAST("2019-09-16T00:00:00,Europe/Moscow" AS TzDatetime), -- 2019-09-16T00:00:00,Europe/Moscow
    CAST("2019-09-16" AS Date) -- 2019-09-16
```

Преобразование времени в дату

`CAST` в `Date` или `TzDate` дает такую дату в таймзоне GMT, в которую происходит полночь по локальному времени (например, для московского времени 2019-10-22 00:00:00, будет возвращена дата 2019-10-21). Для получения даты в локальной таймзоне можно использовать DateTime::Format.

```
$x = Datetime("2019-10-21T21:00:00Z");
SELECT
```

```
    AddTimezone($x, "Europe/Moscow"), -- 2019-10-22T00:00:00,Europe/Moscow
    CAST($x as TzDate), -- 2019-10-21,GMT
    CAST(AddTimezone($x, "Europe/Moscow") as TzDate), -- 2019-10-21,Europe/Moscow
    CAST(AddTimezone($x, "Europe/Moscow") as Date), -- 2019-10-21
  DateTime::Format("%Y-%m-%d %Z")(AddTimezone($x, "Europe/Moscow")), -- 2019-10-22 Europe/Moscow
```

Стоит отметить, что некоторые значения в типах TzDatetime и TzTimestamp, имеющие положительное смещение таймзоны, не могут быть преобразованы в тип TzDate. Рассмотрим следующий пример:

```
SELECT CAST(TzDatetime("1970-01-01T23:59:59,Europe/Moscow") as TzDate);
/* Fatal: Timestamp 1970-01-01T23:59:59.000000,Europe/Moscow cannot be casted to TzDate */
```

Не существует такого значения, начиная с Unix epoch, которым можно представить полночь 01.01.1970 для московского времени. В результате, такое преобразование считается невозможным и генерирует ошибку времени выполнения.

В то же время, значения, имеющие отрицательное смещение таймзоны, возвращают корректный результат:

```
SELECT CAST(TzDatetime("1970-01-01T23:59:59,America/Los_Angeles") as TzDate);
/* 1970-01-01,America/Los_Angeles */
```

Летнее время

Обратите внимание, что летнее время зависит от года:

```
SELECT
    RemoveTimezone(TzDatetime("2019-09-16T10:00:00,Europe/Moscow")) as DST1, -- 2019-09-16T07:00:00Z
    RemoveTimezone(TzDatetime("2008-12-03T10:00:00,Europe/Moscow")) as DST2, -- 2008-12-03T07:00:00Z
    RemoveTimezone(TzDatetime("2008-07-03T10:00:00,Europe/Moscow")) as DST3, -- 2008-07-03T06:00:00Z (DST)
```

Digest

Модуль Digest вычисляют хеши (aka fingerprint'ы) от исходных сообщений. Хеши представляют собой (обычно) некоторую строку небольшого / фиксированного размера. Фактический размер хеша зависит от используемого алгоритма. Результат представляет собой последовательность бит.
Модуль содержит как криптографические, так и некриптографические функции.

Важным свойством алгоритмов является то, что хеш, вероятно, изменится, если сообщение каким-то образом изменится. Другим свойством является то, что функции дайджеста являются односторонними функциями, то есть должно быть трудно найти сообщение, соответствующее некоторому заданному дайджесту.

Список функций

- `Digest::Crc32c(String{Flags::AutoMap}) -> Uint32`
- `Digest::Crc64(String{Flags::AutoMap}, [Init:Uint64?]) -> Uint64`
- `Digest::Fnv32(String{Flags::AutoMap}, [Init:Uint32?]) -> Uint32`
- `Digest::Fnv64(String{Flags::AutoMap}, [Init:Uint64?]) -> Uint64`
- `Digest::MurMurHash(String{Flags:AutoMap}, [Init:Uint64?]) -> Uint64`
- `Digest::MurMurHash32(String{Flags:AutoMap}, [Init:Uint32?]) -> Uint32`
- `Digest::MurMurHash2A(String{Flags:AutoMap}, [Init:Uint64?]) -> Uint64`
- `Digest::MurMurHash2A32(String{Flags:AutoMap}, [Init:Uint32?]) -> Uint32`
- `Digest::CityHash(String{Flags:AutoMap}, [Init:Uint64?]) -> Uint64`
- `Digest::CityHash128(String{Flags:AutoMap}) -> Tuple<Uint64,Uint64>`

Функции для видов хешей, которые поддерживают значение-параметр для инициализации (seed), принимают его опциональным именованным аргументом `Init`.

CityHash функция для байтовой строки с результатом типа uint128. Результат представлен как пара из двух uint64 чисел `<low, high>`

```
SELECT Digest::CityHash128("Who set this ancient quarrel new abroach?"); -- (11765163929838407746,2460323961016211789)
```

- `Digest::NumericHash(Uint64{Flags:AutoMap}) -> Uint64`

```
SELECT Digest::NumericHash(123456789); -- 1734215268924325803
```

- `Digest::Md5Hex(String{Flags:AutoMap}) -> String`
- `Digest::Md5Raw(String{Flags:AutoMap}) -> String`
- `Digest::Md5HalfMix(String{Flags:AutoMap}) -> Uint64`

MD5 - широко распространенный, 128-битный алгоритм хеширования.

Md5Hex - возращает хеш в виде ASCII строки закодировавнной в 16 битной кодировке

Md5Raw - возращает хеш в виде байтовой строки

Md5HalfMix - вариант огрубления MD5 до размера Uint64

```
SELECT
    Digest::Md5Hex("Who set this ancient quarrel new abroach?"), -- "644e98bae764871650f2d93e14c6488d"
    Digest::Md5Raw("Who set this ancient quarrel new abroach?"), -- Binary String: 64 4e 98 ba e7 64 87 16 50 f2 d9 3e 14 c6 48 8d
    Digest::Md5HalfMix("Who set this ancient quarrel new abroach?"); -- 17555822562955248004
```

- `Digest::Argon2(string:String{Flags:AutoMap}, salt:String{Flags:AutoMap}) -> String`

Argon2 - функция формирования ключа, направленная на высокую скорость заполнения памяти и эффективное использование нескольких вычислительных блоков.

Параметры:

- string - исходная строка
- salt - строка которая будет использоавться в качестве соли для хеш функции

```
SELECT Digest::Argon2("Who set this ancient quarrel new abroach?", "zcIvVcuHEIL8"); -- Binary String: fa 50 34 d3 c3 23 a4 de 22 c7 7c e1 9c 65 64 88 25 b3 59 75 c5 b8 8c 73 da 88 eb 79 31 70 e8 f1
SELECT Digest::Argon2("Who set this ancient quarrel new abroach?", "M78P42R8HA=="); -- Binary String: d2 0e f1 3e 72 5a e9 32 65 ed 28 4b 12 1f 39 70 e5 10 aa 1a 15 67 6d 96 5d e8 19 b3 bd d5 04 e9
```

- `Digest::Blake2B(string:String{Flags:AutoMap},[key:String?]) -> String`

BLAKE2 — криптографическая хеш-функция, создана как альтернатива MD5 и SHA-1 алгоритмам, в которых были найдены уязвимости.

Параметры:

- string - исходная строка
- key - ключ, используемый для шифроавния исходной строки (может быть использован как разделяемый секрет между источником и приемником)

```
SELECT Digest::Blake2B("Who set this ancient quarrel new abroach?"); -- Binary String: 62 21 91 d8 11 5a da ad 5e 7c 86 47 41 02 7f 8f a8 a6 82 07 47 d8 f8 30 ab b4 c3 00 db 9c 24 2f
```

- `Digest::SipHash(low:Uint64,high:Uint64,string:String{Flags:AutoMap}) -> Uint64`

Функция для хеширования исходного сообщения (`string`) с помощью 128 битного ключа. Ключ представлен парой uint64 чисел low, high

```
SELECT Digest::SipHash(0,0,"Who set this ancient quarrel new abroach?"); -- 14605466535756698285
```

-

```
Digest::HighwayHash(key0:Uint64,key1:Uint64,key2:Uint64,key3:Uint64,
string:String{Flags:AutoMap}) -> Uint64
```

Функция для хеширования исходного сообщения (`string`) с ключом длиной в 256 бит.

Ключ формируется как:

- key0 - первые 8 байт ключа
- key1 - последующие 8 байт ключа
- key2 - последующие 8 байт ключа
- key3 - последние 8 байт ключа

- `Digest::FarmHashFingerprint(source:Uint64{Flags:AutoMap}) -> Uint64`
- `Digest::FarmHashFingerprint2(low:Uint64{Flags:AutoMap}, high:Uint64{Flags:AutoMap}) -> Uint64`

FarmHash функция для 128 битного числа. 128 битное число получается путем объединения битов из двух uit64 чисел.

- `Digest::FarmHashFingerprint32(string:String{Flags:AutoMap}) -> Uint32`
- `Digest::FarmHashFingerprint64(string:String{Flags:AutoMap}) -> Uint64`
- `Digest::FarmHashFingerprint128(string:String{Flags:AutoMap}) -> Tuple<Uint64,Uint64>`

FarmHash функция для байтовой строки с результатом типа uint128. Результат представлен как пара из двух uint64 чисел `<low, high>`

```
SELECT Digest::FarmHashFingerprint2(31880,6990065); -- 237693065644851126
SELECT Digest::FarmHashFingerprint128("Who set this ancient quarrel new abroach?"); -- (17165761740714960035,
5559728965407786337)
```

- `Digest::SuperFastHash(String{Flags:AutoMap}) -> Uint32`
- `Digest::Sha1(String{Flags:AutoMap}) -> String`
- `Digest::Sha256(String{Flags:AutoMap}) -> String`
- `Digest::IntHash64(Uint64{Flags:AutoMap}) -> Uint64`
- `Digest::XXH3(String{Flags:AutoMap}) -> Uint64`
- `Digest::XXH3_128(String{Flags:AutoMap}) -> Tuple<Uint64,Uint64>`

XXH3 - некриптографическая хеш-функция из семейста xxxHash. XXH3_128 генерит 128 битный хеш, результат представлен как пара из двух uint64 чисел `<low, high>`

```
SELECT Digest::XXH3_128("Who set this ancient quarrel new abroach?"); -- (17117571879768798812, 1428260025880
4776266)
```

Histogram

Набор вспомогательных функций для агрегатной функции HISTOGRAM. В описании сигнатур ниже под HistogramStruct подразумевается результат работы агрегатной функции `HISTOGRAM`, `LinearHistogram` или `LogarithmicHistogram`, который является структурой определенного вида.

Список функций

- `Histogram::Print(HistogramStruct{Flags:AutoMap}, Byte?) -> String`
- `Histogram::Normalize(HistogramStruct{Flags:AutoMap}, [Double?]) -> HistogramStruct` - во втором аргументе желаемая площадь гистограммы, по умолчанию 100.
- `Histogram::ToCumulativeDistributionFunction(HistogramStruct{Flags:AutoMap}) -> HistogramStruct`
- `Histogram::GetSumAboveBound(HistogramStruct{Flags:AutoMap}, Double) -> Double`
- `Histogram::GetSumBelowBound(HistogramStruct{Flags:AutoMap}, Double) -> Double`
- `Histogram::GetSumInRange(HistogramStruct{Flags:AutoMap}, Double, Double) -> Double`
- `Histogram::CalcUpperBound(HistogramStruct{Flags:AutoMap}, Double) -> Double`
- `Histogram::CalcLowerBound(HistogramStruct{Flags:AutoMap}, Double) -> Double`
- `Histogram::CalcUpperBoundSafe(HistogramStruct{Flags:AutoMap}, Double) -> Double`
- `Histogram::CalcLowerBoundSafe(HistogramStruct{Flags:AutoMap}, Double) -> Double`

У `Histogram::Print` есть опциональный числовой аргумент, который задает максимальную длину столбцов гистограммы (в символах, так как гистограмма рисуется в технике ASCII-арт). Значение по умолчанию — 25. Данная функция предназначена в первую очередь для просмотра гистограмм в консоли.

Hyperscan

Hyperscan является opensource библиотекой для поиска по регулярным выражениям, разработанной компанией Intel.

Библиотека имеет 4 реализации с использованием разных наборов процессорных инструкций (SSE3, SSE4.2, AVX2 и AVX512), среди которых автоматически выбирается нужная в соответствии с текущим процессором.

По умолчанию все функции работают в однобайтовом режиме, но если регулярное выражение является валидной UTF-8 строкой, но не является валидной ASCII строкой, — автоматически включается режим UTF-8.

Список функций

- `Hyperscan::Grep(pattern:String) -> (string:String?) -> Bool`
- `Hyperscan::Match(pattern:String) -> (string:String?) -> Bool`
- `Hyperscan::BacktrackingGrep(pattern:String) -> (string:String?) -> Bool`
- `Hyperscan::BacktrackingMatch(pattern:String) -> (string:String?) -> Bool`
- `Hyperscan::MultiGrep(pattern:String) -> (string:String?) -> Tuple<Bool, Bool, ...>`
- `Hyperscan::MultiMatch(pattern:String) -> (string:String?) -> Tuple<Bool, Bool, ...>`
- `Hyperscan::Capture(pattern:String) -> (string:String?) -> String?`
- `Hyperscan::Replace(pattern:String) -> (string:String?, replacement:String) -> String?`

Синтаксис вызова

При вызове напрямую, чтобы избежать компиляции регулярного выражения на каждой строке таблицы, необходимо обернуть вызов функции в именованное выражение:

```
$re = Hyperscan::Grep("\\d+");      -- создаем вызываемое значение для проверки конкретного регулярного выраж
ения
SELECT * FROM table WHERE $re(key); -- используем его для фильтрации таблицы
```

Обратите внимание на экранирование спецсимволов в регулярном выражении. Второй слеш нужен, так как все стандартные строковые литералы в SQL могут принимать C-escaped строки, а последовательность `\d` не является валидной последовательностью, и даже если бы являлась — не приводила бы к ожидаемому эффекту поиска чисел.

Есть возможность отключить чувствительность к регистру (то есть включить case-insensitive режим), указав в начале регулярного выражения флаг `(?i)`.

Grep

Проверяет совпадение регулярного выражения с **частью строки** (произвольной подстрокой).

Match

Проверяет совпадение регулярного выражения **со строкой целиком**.

Чтобы получить результат аналогичный `Grep` (где учитывается совпадение с подстрокой), нужно обрамлять регулярное выражение с обеих сторон в `.*` (`.*foo.*` вместо `foo`). Однако, с точки зрения читабельности кода, обычно лучше поменять функцию.

BacktrackingGrep / BacktrackingMatch

По принципу работы данные функции полностью совпадают с одноимёнными функциями без префикса `Backtracking`, но поддерживают более широкий ассортимент регулярных выражений. Это происходит за счет того, что если конкретное регулярное выражение в полном объёме не поддерживается Hyperscan, то библиотека переключается в режим предварительной фильтрации (prefilter). В этом случае она отвечает не «да» или «нет», а «точно нет» или «может быть да». Ответы «может быть да» затем автоматически перепроверяются с помощью медленной, но более функциональной библиотеки libpcre.

MultiGrep / MultiMatch

Библиотека Hyperscan предоставляет возможность за один проход по тексту проверить несколько регулярных выражений и получить по каждому из них отдельный ответ.

Однако, если необходимо проверить совпадение строки с любым из перечисленных выражений (результаты объединялись бы через «или»), то эффективнее сделать одно регулярное выражение, объединив части с помощью оператора `|`, и использовать для поиска обычный `Grep` или `Match`.

При вызове функций `MultiGrep` / `MultiMatch` регулярные выражения передаются по одному на строку с использованием многострочных строковых литералов:

Пример

```
$multi_match = Hyperscan::MultiMatch(@@a.*
.*x.*
.*axa.*@@);

SELECT
    $multi_match("a") AS a,     -- (true, false, false)
    $multi_match("axa") AS axa; -- (true, true, true)
```

Capture и Replace

`Hyperscan::Capture` при совпадении строки с указанным регулярным выражением возвращает последнюю подстроку, совпавшую с регулярным выражением. `Hyperscan::Replace` заменяет все вхождения указанного регулярного выражения на заданную строку.

В библиотеке Hyperscan отсутствует развитая функциональность для подобных операций, так что `Hyperscan::Capture` и `Hyperscan::Replace` хоть и реализованы для единообразия, но для сколько-либо нетривиальных поисков и замен лучше использовать одноимённые функции из библиотеки Re2:

- Re2::Capture;
- Re2::Replace.

Пример использования

```
$value = "xaaxaaXaa";

$match = Hyperscan::Match("a.*");
$grep = Hyperscan::Grep("axa");
$insensitive_grep = Hyperscan::Grep("(?i)axaa$");
$multi_match = Hyperscan::MultiMatch(@@a.*
.*a.*
.*a
.*axa.*@@);

$capture = Hyperscan::Capture(".*a{2}.*");
$capture_many = Hyperscan::Capture(".*x(a+).*");
$replace = Hyperscan::Replace("xa");

SELECT
    $match($value) AS match,                        -- false
    $grep($value) AS grep,                          -- true
    $insensitive_grep($value) AS insensitive_grep,  -- true
    $multi_match($value) AS multi_match,            -- (false, true, true, true)
    $multi_match($value).0 AS some_multi_match,     -- false
    $capture($value) AS capture,                    -- "xaa"
    $capture_many($value) AS capture_many,          -- "xa"
    $replace($value, "b") AS replace                -- "babaXaa"
;
```

Ip

В модуле `Ip` поддерживаются как IPv4, так и IPv6 адреса. По умолчанию они представляются в виде бинарных строк длиной 4 и 16 байт, соответственно.

Список функций

- `Ip::FromString(String{Flags:AutoMap}) -> String?` - из человекочитаемого представления в бинарное
- `Ip::SubnetFromString(String{Flags:AutoMap}) -> String?` - из человекочитаемого представления подсети в бинарное
- `Ip::ToString(String{Flags:AutoMap}) -> String?` - из бинарного представления в человекочитаемое
- `Ip::ToString(String{Flags:AutoMap}) -> String?` - из бинарного представления подсети в человекочитаемое
- `Ip::IsIPv4(String?) -> Bool`
- `Ip::IsIPv6(String?) -> Bool`
- `Ip::IsEmbeddedIPv4(String?) -> Bool`
- `Ip::ConvertToIPv6(String{Flags:AutoMap}) -> String` - IPv6 остается без изменений, а IPv4 становится embedded в IPv6
- `Ip::GetSubnet(String{Flags:AutoMap}, [Uint8?]) -> String` - во втором аргументе размер подсети, по умолчанию 24 для IPv4 и 64 для IPv6
- `Ip::GetSubnetByMask(String{Flags:AutoMap}, String{Flags:AutoMap}) -> String` - во втором аргументе битовая маска подсети
- `Ip::SubnetMatch(String{Flags:AutoMap}, String{Flags:AutoMap}) -> Bool` - в первом аргументе подсеть, во втором аргументе подсеть или адрес

Примеры

```
SELECT Ip::IsEmbeddedIPv4(
  Ip::FromString("::ffff:77.75.155.3")
); -- true

SELECT
  Ip::ToString(
    Ip::GetSubnet(
      Ip::FromString("213.180.193.3")
    )
  ); -- "213.180.193.0"

SELECT
  Ip::SubnetMatch(
    Ip::SubnetFromString("192.168.0.1/16"),
    Ip::FromString("192.168.1.14"),
  ); -- true

SELECT
  Ip::ToString(
    Ip::GetSubnetByMask(
      Ip::FromString("192.168.0.1"),
      Ip::FromString("255.255.0.0")
    )
  ); -- "192.168.0.0"
```

KNN

Введение

Одним из частных случаев векторного поиска является задача k-NN, где требуется найти `k` ближайших точек к точке запроса. Это может быть полезно в различных приложениях, таких как классификация изображений, рекомендательные системы и многое другое.

Решения задачи k-NN разбивается на два крупных подкласса методов: точные и приближенные.

Точный метод

В основе точного метода лежит вычисление расстояния от вектора запроса до всех векторов в наборе данных. Этот алгоритм, также известный как наивный подход или метод грубой силы, имеет время выполнения `O(dn)`, где `n` — количество векторов в наборе данных, а `d` — их размерность.

Точный векторный поиск лучше использовать, если полный перебор искомых векторов происходит за приемлемое время. В том числе, когда их можно предварительно отфильтровать по некоторому условию, например, по идентификатору пользователя. В таких случаях точный метод может работать быстрее, чем текущая реализация векторных индексов.

Основные преимущества:

- отсутствие необходимости в дополнительных структурах данных, таких как специализированные векторные индексы;
- полная поддержка транзакций, в том числе в режиме строгой согласованности;
- мгновенное применение операций модификации данных: вставка, обновление, удаление.

Приближенные методы

Приближенные методы могут избегать полного перебора исходных данных или снижать точность векторов. Это позволяет значительно ускорить процесс поиска, хотя и может привести к некоторому снижению качества результатов.

Скалярное квантование — это метод уменьшения точности координат с сохранением размерности векторов.

YDB поддерживает векторный поиск по векторам типов `Float`, Float16, BFloat16, `Int8`, `Uint8`, `Bit`. Следовательно, возможно применение скалярного квантования для преобразования данных из `Float` в любой из этих типов.

Скалярное квантование сокращает время, необходимое для чтения и записи данных, за счёт уменьшения числа байт. Например, при квантовании из `Float` в `Bit` объём координат сокращается в 32 раза без учёта служебных данных сериализации.

Приближенный векторный поиск без индекса использует очень простую дополнительную структуру данных - множество векторов с другим квантованием. Это позволяет использовать простой алгоритм поиска: сначала грубый предварительный поиск по сжатым векторам, а затем уточнять результаты по исходным векторам.

Основные преимущества:

- полная поддержка транзакций, в том числе в режиме строгой согласованности;
- мгновенное применение операций модификации данных: вставка, обновление, удаление.

> **Примечание**
>
> Рекомендуется проводить замеры, обеспечивает ли такое преобразование достаточную точность/полноту.

Типы данных

В математике для хранения точек используется вектор вещественных или целых чисел.
В этом модуле вектора представлены типом данных `String`, который является бинарным сериализованным представлением вектора.

Функции

Функции работы с векторами реализовываются в виде пользовательских функций (UDF) в модуле `Knn`.

Функции преобразования вектора в бинарное представление

Функции преобразования нужны для сериализации векторов во внутреннее бинарное представление и обратно.

Все функции сериализации упаковывают возвращаемые данные типа `String` в Tagged тип.

Бинарное представление вектора можно сохранить в YDB колонку.

> **Примечание**
>
> В настоящий момент YDB не поддерживает хранение `Tagged` типов и поэтому перед сохранением бинарного представления векторов нужно извлечь `String` с помощью функции Untag.

Сигнатуры функций

```
Knn::ToBinaryStringFloat(List<Float>{Flags:AutoMap})->Tagged<String, "FloatVector">
Knn::ToBinaryStringFloat16(List<Float>{Flags:AutoMap})->Tagged<String, "Float16Vector">
Knn::ToBinaryStringBFloat16(List<Float>{Flags:AutoMap})->Tagged<String, "BFloat16Vector">
Knn::ToBinaryStringUint8(List<Uint8>{Flags:AutoMap})->Tagged<String, "Uint8Vector">
Knn::ToBinaryStringInt8(List<Int8>{Flags:AutoMap})->Tagged<String, "Int8Vector">
Knn::ToBinaryStringBit(List<Double>{Flags:AutoMap})->Tagged<String, "BitVector">
Knn::ToBinaryStringBit(List<Float>{Flags:AutoMap})->Tagged<String, "BitVector">
Knn::ToBinaryStringBit(List<Uint8>{Flags:AutoMap})->Tagged<String, "BitVector">
Knn::ToBinaryStringBit(List<Int8>{Flags:AutoMap})->Tagged<String, "BitVector">
Knn::FloatFromBinaryString(String{Flags:AutoMap})->List<Float>?
```

Формат сериализации

Функции сериализации векторных данных преобразуют массив элементов в байтовую строку следующего формата:

- **Основная часть** — непрерывный массив элементов (knn-serializer.h)
- **Тип** — 1 байт в конце строки, обозначающий тип данных (knn-defines.h):
  - `1` — `Float` (4 байта на элемент);
  - `2` — `Uint8` (1 байт на элемент);
  - `3` — `Int8` (1 байт на элемент);
  - `4` — `Float16` (2 байта на элемент, IEEE-754 binary16);
  - `5` — `BFloat16` (2 байта на элемент, bfloat16);
  - `10` — `Bit` (1 бит на элемент).

Например, вектор из 5 элементов типа `Float` сериализуется в строку длиной 21 байт: 4 байта × 5 элементов (основная часть) + 1 байт (тип) = 21 байт.

Формат `Bit` содержит дополнительный байт перед маркером типа с количеством неиспользованных битов в последнем байте данных.

Детали имплементации

`ToBinaryStringFloat16` и `ToBinaryStringBFloat16` округляют координаты `Float` до ближайшего представимого значения, при равенстве расстояний — до чётного. `FloatFromBinaryString` преобразует их обратно в `Float`. Для векторных индексов с такими форматами укажите `vector_type=float16` или `vector_type=bfloat16`.

`ToBinaryStringBit` преобразует в `1` все координаты, которые больше `0`. Остальные координаты преобразуются в `0`.

Функции расстояния и сходства

Функции расстояния и сходства принимают на вход два сериализованных вектора (`String`) и возвращают расстояние/сходство между ними.

> **Примечание**
>
> Функции расстояния возвращают малое значение для близких векторов, функции сходства возвращают большие значения для близких векторов. Это следует учитывать в порядке сортировки.

Функции сходства:

- скалярное произведение `InnerProductSimilarity` (сумма произведений координат)
- косинусное сходство `CosineSimilarity` (скалярное произведение разделенное на произведение длин векторов)

Функции расстояния:

- косинусное расстояние `CosineDistance` (1 - косинусное сходство)
- манхэттенское расстояние `ManhattanDistance`, также известно как `L1 distance` (сумма модулей покоординатной разности)
- Евклидово расстояние `EuclideanDistance`, также известно как `L2 distance` (корень суммы квадратов покоординатной разности)

Сигнатуры функций

```
Knn::InnerProductSimilarity(String{Flags:AutoMap}, String{Flags:AutoMap})->Float?
Knn::CosineSimilarity(String{Flags:AutoMap}, String{Flags:AutoMap})->Float?
Knn::CosineDistance(String{Flags:AutoMap}, String{Flags:AutoMap})->Float?
Knn::ManhattanDistance(String{Flags:AutoMap}, String{Flags:AutoMap})->Float?
Knn::EuclideanDistance(String{Flags:AutoMap}, String{Flags:AutoMap})->Float?
```

В случае отличающихся длины или формата, эти функции возвращают `NULL`.

> **Примечание**
>
> Все функции расстояния и сходства поддерживают перегрузки с аргументами одного из типов `Tagged<String, "FloatVector">`, `Tagged<String, "Float16Vector">`, `Tagged<String, "BFloat16Vector">`, `Tagged<String, "Uint8Vector">`, `Tagged<String, "Int8Vector">`, `Tagged<String, "BitVector">`.
>
> Если оба аргумента `Tagged`, то значение тега должно совпадать, иначе запрос завершится с ошибкой.
>
> Пример:
>
> ```
> Error: Failed to find UDF function: Knn.CosineDistance, reason: Error: Module: Knn, function: Cosi
> neDistance, error: Arguments should have same tags, but 'FloatVector' is not equal to 'Uint8Vecto
> r'
> ```

Примеры точного поиска

Создание таблицы

```
CREATE TABLE Facts (
    id Uint64,        -- Id of fact
    user Utf8,        -- User name
    fact Utf8,        -- Human-readable description of a user fact
    embedding String, -- Binary representation of embedding vector (result of Knn::ToBinaryStringFloat)
```

```
    PRIMARY KEY (id)
);
```

Добавление векторов

```
$vector = [1.f, 2.f, 3.f, 4.f];
UPSERT INTO Facts (id, user, fact, embedding)
VALUES (123, "Williams", "Full name is John Williams", Untag(Knn::ToBinaryStringFloat($vector), "FloatVecto
r"));
```

Точный поиск K ближайших векторов

```
$K = 10;
$TargetEmbedding = Knn::ToBinaryStringFloat([1.2f, 2.3f, 3.4f, 4.5f]);

SELECT * FROM Facts
WHERE user="Williams"
ORDER BY Knn::CosineDistance(embedding, $TargetEmbedding)
LIMIT $K;
```

Точный поиск векторов, находящихся в радиусе R

```
$R = 0.1f;
$TargetEmbedding = Knn::ToBinaryStringFloat([1.2f, 2.3f, 3.4f, 4.5f]);

SELECT * FROM Facts
WHERE Knn::CosineDistance(embedding, $TargetEmbedding) < $R;
```

Примеры приближенного поиска

Данный пример отличается от примера с точным поиском использованием битового квантования.
Это позволяет сначала делать грубый предварительный поиск по колонке `embedding_bit`, а затем уточнять результаты по основной колонке с векторами `embedding`.

Создание таблицы

```
CREATE TABLE Facts (
    id Uint64,            -- Id of fact
    user Utf8,            -- User name
    fact Utf8,            -- Human-readable description of a user fact
    embedding String,     -- Binary representation of embedding vector (result of Knn::ToBinaryStringFloat)
    embedding_bit String, -- Binary representation of embedding vector (result of Knn::ToBinaryStringBit)
    PRIMARY KEY (id)
);
```

Добавление векторов

```
$vector = [1.f, 2.f, 3.f, 4.f];
UPSERT INTO Facts (id, user, fact, embedding, embedding_bit)
VALUES (123, "Williams", "Full name is John Williams", Untag(Knn::ToBinaryStringFloat($vector), "FloatVecto
r"), Untag(Knn::ToBinaryStringBit($vector), "BitVector"));
```

Скалярное квантование

ML модель может выполнять квантование или это можно сделать вручную с помощью YQL.

Ниже приведен пример квантования в YQL.

Float -> Int8

```
$MapInt8 = ($x) -> {
    $min = -5.0f;
    $max =  5.0f;
    $range = $max - $min;
  RETURN CAST(Math::Round(IF($x < $min, -127, IF($x > $max, 127, ($x / $range) * 255))) As Int8)
};

$FloatList = [-1.2f, 2.3f, 3.4f, -4.7f];
SELECT ListMap($FloatList, $MapInt8);
```

Приближенный поиск K ближайших векторов: битовое квантование

Алгоритм приближенного поиска:

- производится приближенный поиск с использованием битового квантования;
- получается приближенный список векторов;
- в этом списке производим поиск без использования квантования.

```
$K = 10;
$Target = [1.2f, 2.3f, 3.4f, 4.5f];
$TargetEmbeddingBit = Knn::ToBinaryStringBit($Target);
$TargetEmbeddingFloat = Knn::ToBinaryStringFloat($Target);

$Ids = SELECT id FROM Facts
```

```
ORDER BY Knn::CosineDistance(embedding_bit, $TargetEmbeddingBit)
LIMIT $K * 10;

SELECT * FROM Facts
WHERE id IN $Ids
ORDER BY Knn::CosineDistance(embedding, $TargetEmbeddingFloat)
LIMIT $K;
```

Math

Набор обёрток вокруг функций из библиотеки libm, а также утилит Яндекс.

Константы

Список функций

- `Math::Pi() -> Double`
- `Math::E() -> Double`
- `Math::Eps() -> Double`

Примеры

```
SELECT Math::Pi();  -- 3.141592654
SELECT Math::E();   -- 2.718281828
SELECT Math::Eps(); -- 2.220446049250313e-16
```

(Double) -> Bool

Список функций

- `Math::IsInf(Double{Flags:AutoMap}) -> Bool`
- `Math::IsNaN(Double{Flags:AutoMap}) -> Bool`
- `Math::IsFinite(Double{Flags:AutoMap}) -> Bool`

Примеры

```
SELECT Math::IsNaN(0.0/0.0);    -- true
SELECT Math::IsFinite(1.0/0.0); -- false
```

(Double) -> Double

Список функций

- `Math::Abs(Double{Flags:AutoMap}) -> Double`
- `Math::Acos(Double{Flags:AutoMap}) -> Double`
- `Math::Asin(Double{Flags:AutoMap}) -> Double`
- `Math::Asinh(Double{Flags:AutoMap}) -> Double`
- `Math::Atan(Double{Flags:AutoMap}) -> Double`
- `Math::Cbrt(Double{Flags:AutoMap}) -> Double`
- `Math::Ceil(Double{Flags:AutoMap}) -> Double`
- `Math::Cos(Double{Flags:AutoMap}) -> Double`
- `Math::Cosh(Double{Flags:AutoMap}) -> Double`
- `Math::Erf(Double{Flags:AutoMap}) -> Double`
- `Math::ErfInv(Double{Flags:AutoMap}) -> Double`
- `Math::ErfcInv(Double{Flags:AutoMap}) -> Double`
- `Math::Exp(Double{Flags:AutoMap}) -> Double`
- `Math::Exp2(Double{Flags:AutoMap}) -> Double`
- `Math::Fabs(Double{Flags:AutoMap}) -> Double`
- `Math::Floor(Double{Flags:AutoMap}) -> Double`
- `Math::Lgamma(Double{Flags:AutoMap}) -> Double`
- `Math::Rint(Double{Flags:AutoMap}) -> Double`
- `Math::Sigmoid(Double{Flags:AutoMap}) -> Double`
- `Math::Sin(Double{Flags:AutoMap}) -> Double`
- `Math::Sinh(Double{Flags:AutoMap}) -> Double`
- `Math::Sqrt(Double{Flags:AutoMap}) -> Double`
- `Math::Tan(Double{Flags:AutoMap}) -> Double`
- `Math::Tanh(Double{Flags:AutoMap}) -> Double`
- `Math::Tgamma(Double{Flags:AutoMap}) -> Double`
- `Math::Trunc(Double{Flags:AutoMap}) -> Double`
- `Math::Log(Double{Flags:AutoMap}) -> Double`
- `Math::Log2(Double{Flags:AutoMap}) -> Double`
- `Math::Log10(Double{Flags:AutoMap}) -> Double`

Примеры

```
SELECT Math::Sqrt(256);     -- 16
SELECT Math::Trunc(1.2345); -- 1
```

(Double, Double) -> Double

Список функций

- `Math::Atan2(Double{Flags:AutoMap}, Double{Flags:AutoMap}) -> Double`
- `Math::Fmod(Double{Flags:AutoMap}, Double{Flags:AutoMap}) -> Double`
- `Math::Hypot(Double{Flags:AutoMap}, Double{Flags:AutoMap}) -> Double`

- `Math::Pow(Double{Flags:AutoMap}, Double{Flags:AutoMap}) -> Double`
- `Math::Remainder(Double{Flags:AutoMap}, Double{Flags:AutoMap}) -> Double`

Примеры

```
SELECT Math::Atan2(1, 0);       -- 1.570796327
SELECT Math::Remainder(2.1, 2); -- 0.1
```

(Double, Int32) -> Double

Список функций

- `Math::Ldexp(Double{Flags:AutoMap}, Int32{Flags:AutoMap}) -> Double`
- `Math::Round(Double{Flags:AutoMap}, [Int32?]) -> Double` - во втором аргументе указывается степень 10, до которой округляем (отрицательная для знаков после запятой и положительная для округления до десятков—тысяч—миллионов); по умолчанию 0

Примеры

```
SELECT Math::Pow(2, 10);        -- 1024
SELECT Math::Round(1.2345, -2); -- 1.23
```

(Double, Double, [Double?]) -> Bool

Список функций

- `Math::FuzzyEquals(Double{Flags:AutoMap}, Double{Flags:AutoMap}, [Double?]) -> Bool` - сравнивает два Double на нахождение внутри окрестности, задаваемой третьим аргументом; по умолчанию 1.0e-13. Окрестность указывается в относительных единицах от минимального по модулю аргумента.

> **Внимание**
>
> Если один из сравниваемых аргументов равен `0.0`, то функция всегда возвращает `false`.

Примеры

```
SELECT Math::FuzzyEquals(1.01, 1.0, 0.05); -- true
```

Функции взятия остатка

Список функций

- `Math::Mod(Int64{Flags:AutoMap}, Int64) -> Int64?`
- `Math::Rem(Int64{Flags:AutoMap}, Int64) -> Int64?`

Ведут себя аналогично встроенному оператору % в случае неотрицательных аргументов. Различия заметны при отрицательных аргументах:

- Math::Mod сохраняет знак второго аргумента (делителя).
- Math::Rem сохраняет знак первого аргумента (делимого).

Функции возвращают null, если делитель равен нулю.

Примеры

```
SELECT Math::Mod(-1, 7);        -- 6
SELECT Math::Rem(-1, 7);        -- -1
```

Функции округления до целого числа в заданном режиме

Список функций

- `Math::RoundDownward() -> Tagged<Uint32, MathRoundingMode>` -- rounding towards negative infinity
- `Math::RoundToNearest() -> Tagged<Uint32, MathRoundingMode>` -- rounding towards nearest representable value
- `Math::RoundTowardZero() -> Tagged<Uint32, MathRoundingMode>` -- rounding towards zero
- `Math::RoundUpward() -> Tagged<Uint32, MathRoundingMode>` -- rounding towards positive infinity
- `Math::NearbyInt(AutoMap<Double>, Tagged<Uint32, MathRoundingMode>) -> Optional<Int64>`

Функция `Math::NearbyInt` округляет первый аргумент до целого числа в соответсвии с режимом, заданным вторым аргументом.

Если результат выходит за пределы 64-битного целого числа, возращается NULL.

Примеры

```
SELECT Math::NearbyInt(1.5, Math::RoundDownward()); -- 1
SELECT Math::NearbyInt(1.5, Math::RoundToNearest()); -- 2
SELECT Math::NearbyInt(2.5, Math::RoundToNearest()); -- 2
SELECT Math::NearbyInt(1.5, Math::RoundTowardZero()); -- 1
SELECT Math::NearbyInt(1.5, Math::RoundUpward()); -- 2
SELECT Math::NearbyInt(-1.5, Math::RoundDownward()); -- -2
SELECT Math::NearbyInt(-1.5, Math::RoundToNearest()); -- -2
SELECT Math::NearbyInt(-2.5, Math::RoundToNearest()); -- -2
```

```
SELECT Math::NearbyInt(-1.5, Math::RoundTowardZero()); -- -1
SELECT Math::NearbyInt(-1.5, Math::RoundUpward()); -- -1
```

Pcre

Библиотека Pcre на данный момент является алиасом к Hyperscan.

На данный момент доступны:

- Hyperscan (Intel)
- Pire (Яндекс)
- Re2 (Google)

Все три модуля предоставляют примерно одинаковый набор функций с идентичным интерфейсом, что позволяет переключаться между ними с минимальными изменениями запроса.

HyperScan внутри имеет несколько реализаций с использованием разных наборов процессорных инструкций, среди которых автоматически выбирается нужная в соответствии с текущим процессором. В Hyperscan также доступны отдельные функции с backtracking (возможность сослаться на предыдущую найденную часть строки), которые реализованы через гибридное использование двух библиотек Hyperscan и libpcre.

Pire (Perl Incompatible Regular Expressions) это разработанная в Яндексе очень быстрая библиотека регулярных выражений. На нижнем уровне она просматривает входную строку один раз, подряд, без откатов, и тратит (на x86 и x86_64) по 5 инструкций на символ. Но с 2011-2013 года эта библиотека практически не развивается и как намекает название («i» расшифровывается как incompatible), возможно потребуется адаптировать сами выражения.

Hyperscan и Pire рассчитаны в первую очередь под Grep и Match.

Модуль Re2 использует google::RE2, где реализован широкий ассортимент возможностей (см. официальную документацию). Основной плюс библиотеки Re2 — развитый функционал по Capture и Replace, если вам нужны эти функции, то рекомендуется пользоваться именно ей.

Pire

Список функций

- `Pire::Grep(pattern:String) -> (string:String?) -> Bool`
- `Pire::Match(pattern:String) -> (string:String?) -> Bool`
- `Pire::MultiGrep(pattern:String) -> (string:String?) -> Tuple<Bool, Bool, ...>`
- `Pire::MultiMatch(pattern:String) -> (string:String?) -> Tuple<Bool, Bool, ...>`
- `Pire::Capture(pattern:String) -> (string:String?) -> String?`
- `Pire::Replace(pattern:String) -> (string:String?, replacement:String) -> String?`

Одной из опций для поиска по регулярным выражениям в YQL является библиотека Pire (Perl Incompatible Regular Expressions). Это разработанная в Яндексе очень быстрая библиотека регулярных выражений: на нижнем уровне она просматривает входную строку один раз, подряд, без откатов, и тратит (на x86 и x86_64) по 5 инструкций на символ.

Скорость работы достигается за счет некоторых разумных ограничений:

- Библиотека Pire ориентирована прежде всего на проверку на совпадение строки с регулярным выражением (Match).
- Возможность вернуть совпавшую подстроку тоже поддерживается (Capture), но с ограничениями (возвращает совпадение только с одной группой).

По умолчанию все функции работают в однобайтовом режиме, но если регулярное выражение является валидной UTF-8 строкой, но не является валидной ASCII строкой, — автоматически включается режим UTF-8.

Чтобы включить Unicode-режим, можно вставить в выражение один символ за пределами ASCII с оператором `?`, например `\\w+я?`.

Синтаксис вызова

Чтобы избежать компиляции регулярного выражения на каждой строке таблицы, необходимо обернуть вызов функции в именованное выражение:

```
$re = Pire::Grep("\\d+"); -- создаем вызываемое значение для проверки конкретного регулярного выражения
SELECT * FROM table WHERE $re(key); -- используем его для фильтрации таблицы
```

Внимание

При экранировании спецсимволов в регулярном выражении нужен второй слеш, так как все стандартные строковые литералы в SQL могут принимать C-escaped строки, а последовательность `\d` не является валидной последовательностью, и даже если бы являлась — не приводила бы к ожидаемому эффекту поиска чисел.

Есть возможность отключить чувствительность к регистру (то есть включить case-insensitive режим), указав в начале регулярного выражения флаг `(?i)`.

Примеры

```
$value = "xaaxaaxaa";
$match = Pire::Match("a.*");
$grep = Pire::Grep("axa");
$insensitive_grep = Pire::Grep("(?i)axa");
$multi_match = Pire::MultiMatch(@@a.*
.*a.*
.*a
.*axa.*@@);
$capture = Pire::Capture(".*x(a).*");
$capture_many = Pire::Capture(".*x(a+).*");
$replace = Pire::Replace(".*x(a).*");

SELECT
  $match($value) AS match,                        -- false
  $grep($value) AS grep,                          -- true
  $insensitive_grep($value) AS insensitive_grep,  -- true
  $multi_match($value) AS multi_match,            -- (false, true, true, true)
  $multi_match($value).0 AS some_multi_match,     -- false
  $capture($value) AS capture,                    -- "a"
  $capture_many($value) AS capture_many,          -- "aa"
  $replace($value, "b") AS replace;               -- "xaaxaaxba"
```

Grep

Проверяет совпадение регулярного выражения с **частью строки** (произвольной подстрокой).

Match

Проверяет совпадение регулярного выражения **со строкой целиком**.
Чтобы получить результат, аналогичный `Grep` (где учитывается совпадение с подстрокой), нужно обернуть регулярное выражение с обоих сторон в `.*`, например `.*foo.*` вместо `foo`.

MultiGrep / MultiMatch

Библиотека Pire предоставляет возможность за один проход по тексту проверить несколько регулярных выражений и получить по каждому из них отдельный ответ.
С помощью функций MultiGrep/MultiMatch можно оптимизировать скорость выполнения запроса, но это нужно делать осмотрительно, поскольку размер используемого для проверки конечного автомата растет экспоненциально с числом регулярных выражений:

- Если вас интересует совпадение строки с любым из перечисленных выражений (результаты объединяются через «или»), то намного эффективнее сделать одно регулярное выражение, объединив части с помощью оператора `|`, и использовать для поиска обычный Grep или Match.
- В библиотеке Pire установлен лимит на размер конечного автомата (в YQL используется значение этого лимита, установленное по умолчанию в библиотеке). Если лимит превышен, при запуске запроса вернется ошибка `Failed to glue up regexes, probably the finite state machine appeared to be too large`.

При вызове функций MultiGrep/MultiMatch регулярные выражения передаются по одному на строку с использованием многострочных строковых литералов:

Примеры

```
$multi_match = Pire::MultiMatch(@@a.*
.*x.*
.*axa.*@@);

SELECT
    $multi_match("a") AS a,      -- (true, false, false)
    $multi_match("axa") AS axa;  -- (true, true, true)
```

Capture

При совпадении строки с указанным регулярным выражением возвращает подстроку, совпавшую с группой, отмеченной в регулярном выражении круглыми скобками.
Capture работает НЕ в жадном режиме, то есть возвращается самая короткая подстрока из возможных.

> **Внимание**
>
> В выражении должна быть ровно **одна** группа, обозначенная круглыми скобками. В случае отсутствия совпадения возвращается `NULL` (пустой Optional).

Если описанные выше ограничения и особенности по каким-либо причинам выглядят неприемлемыми, рекомендуется рассмотреть возможность использования Re2::Capture.

Replace

В библиотеке Pire отсутствует функция замены по регулярному выражению. Функция `Pire::Replace` в YQL представляет собой упрощенную эмуляцию, реализованную с помощью `Capture`. Может работать некорректно, если найденная подстрока встречается в исходной более одного раза.

Как правило, лучше воспользоваться Re2::Replace вместо неё.

PostgreSQL UDF

YQL предоставляет возможность доступа к функциям и типам данных PostgreSQL.

Имена PostgreSQL типов в YQL получаются добавлением префикса `Pg` к исходному имени типа.
Например `PgVarchar`, `PgInt4`, `PgText`. Имена pg типов (как и вообще всех типов) в YQL являются case-insensitive. На данный момент поддерживаются все простые типы данных из PostgreSQL, а также массивы.

Если исходный тип является типом массива (в PostgreSQL такие типы начинаются с подчеркивания: `_int4` - массив 32-битных целых), то имя типа в YQL тоже начинается с подчеркивания – `_PgInt4`.

Литералы

Строковые и числовые литералы Pg типов можно создавать с помощью специальных суффиксов (аналогично простым строковым и числовым литералам).

Целочисленные литералы

Суффикс	Тип	Комментарий
`p`	`PgInt4`	32-битное знаковое целое (в PostgreSQL нет беззнаковых типов)
`ps`	`PgInt2`	16-битное знаковое целое
`pi`	`PgInt4`	
`pb`	`PgInt8`	64-битное знаковое цело
`pn`	`PgNumeric`	знаковое целое произвольной точности (до 131072 цифр)

Литералы с плавающей точкой

Суффикс	Тип	Комментарий
`p`	`PgFloat8`	число с плавающей точкой (64 бит double)
`pf4`	`PgFloat4`	число с плавающей точкой (32 бит float)
`pf8`	`PgFloat8`	
`pn`	`PgNumeric`	число с плавающей точкой произвольной точности (до 131072 цифр перед запятой, до 16383 цифр после запятой)

Строковые литералы

Суффикс	Тип	Комментарий
`p`	`PgText`	текстовая строка
`pt`	`PgText`	
`pv`	`PgVarchar`	текстовая строка
`pb`	`PgBytea`	бинарная строка

Внимание

Значения строковых/числовых литералов (т.е. то что идет перед суффиксом) должны быть валидной строкой/числом с точки зрения YQL.
В частности, должны соблюдаться правила эскейпинга YQL, а не PostgreSQL.

Пример:

```
SELECT
    1234p,       -- pgint4
    0x123pb,     -- pgint8
    "тест"pt,    -- pgtext
    123e-1000pn; -- pgnumeric
;
```

Литерал массива

Для построения литерала массива используется функция `PgArray`:

```
SELECT
    PgArray(1p, NULL ,2p) -- {1,NULL,2}, тип _int4
;
```

Конструктор литералов произвольного типа

Литералы всех типов (в том числе и Pg типов) могут создаваться с помощью конструктора литералов со следующей сигнатурой:
`Имя_типа(<строковая константа>)`.

Напрмер:

```
DECLARE $foo AS String;
SELECT
    PgInt4("1234"), -- то же что и 1234p
    PgInt4(1234),   -- в качестве аргумента можно использовать литеральные константы
    PgInt4($foo),   -- и declare параметры
    PgBool(true),
    PgInt8(1234),
    PgDate("1932-01-07"),
;
```

Также поддерживается встроенная функция `PgConst` со следующей сигнатурой: `PgConst(<строковое значение>, <тип>)`. Такой способ более удобен для кодогенерации.

Например:

```
SELECT
    PgConst("1234", PgInt4), -- то же что и 1234p
    PgConst("true", PgBool)
;
```

Для некоторых типов в функции `PgConst` можно указать дополнительные модификаторы. Возможные модификаторы для типа `pginterval` перечислены в документации PostgreSQL.

```
SELECT
    PgConst(90, pginterval, "day"), -- 90 days
    PgConst(13.45, pgnumeric, 10, 1); -- 13.5
;
```

Операторы

Операторы PostgreSQL (унарные и бинарные) доступны через встроенную функцию `PgOp(<оператор>, <операнды>)`:

```
SELECT
    PgOp("*", 123456789987654321pn, 9999999999999999999pn), --  12345678998765432099876543210012345679
    PgOp('|/', 10000.0p), -- 100.0p (квадратный корень)
    PgOp("-", 1p), -- -1p
    -1p,           -- унарный минус для литералов работает и без PgOp
;
```

Оператор приведения типа

Для приведения значения одного Pg типа к другому используется встроенная функция `PgCast(<исходное значение>, <желаемый тип>)`:

```
SELECT
    PgCast(123p, PgText), -- преобразуем число в строку
;
```

При преобразовании из строковых Pg типов в некоторые целевые типы можно указать дополнительные модификаторы. Возможные модификаторы для типа `pginterval` перечислены в документации.

```
SELECT
    PgCast('90'p, pginterval, "day"), -- 90 days
    PgCast('13.45'p, pgnumeric, 10, 1); -- 13.5
;
```

Преобразование значений Pg типов в значения YQL типов и обратно

Для некоторых Pg типов возможна конвертация в YQL типы и обратно. Конвертация осуществляется с помощью встроенных функций
`FromPg(<значение Pg типа>)` и `ToPg(<значение YQL типа>)`:

```
SELECT
    FromPg("тест"pt), -- Just(Utf8("тест")) - pg типы всегда nullable
    ToPg(123.45), -- 123.45pf8
;
```

Список псевдонимов типов PostgreSQL при их использовании в YQL

Ниже приведены типы данных YQL, соответствующие им логические типы PostgreSQL и названия типов PostgreSQL при их использовании в YQL:

YQL	PostgreSQL	Название PostgreSQL-типа в YQL
`Bool`	`bool`	`pgbool`
`Int8`	`int2`	`pgint2`
`Uint8`	`int2`	`pgint2`
`Int16`	`int2`	`pgint2`

`Uint16`	`int4`	`pgint4`
`Int32`	`int4`	`pgint4`
`Uint32`	`int8`	`pgint8`
`Int64`	`int8`	`pgint8`
`Uint64`	`numeric`	`pgnumeric`
`Float`	`float4`	`pgfloat4`
`Double`	`float8`	`pgfloat8`
`String`	`bytea`	`pgbytea`
`Utf8`	`text`	`pgtext`
`Yson`	`bytea`	`pgbytea`
`Json`	`json`	`pgjson`
`Uuid`	`uuid`	`pguuid`
`JsonDocument`	`jsonb`	`pgjsonb`
`Date`	`date`	`pgdate`
`Datetime`	`timestamp`	`pgtimestamp`
`Timestamp`	`timestamp`	`pgtimestamp`
`Interval`	`interval`	`pginterval`
`TzDate`	`text`	`pgtext`
`TzDatetime`	`text`	`pgtext`
`TzTimestamp`	`text`	`pgtext`
`Date32`	`date`	`pgdate`
`Datetime64`	`timestamp`	`pgtimestamp`
`Timestamp64`	`timestamp`	`pgtimestamp`
`Interval64`	`interval`	`pginterval`
`TzDate32`	`text`	`pgtext`
`TzDatetime64`	`text`	`pgtext`
`TzTimestamp64`	`text`	`pgtext`
`Decimal`	`numeric`	`pgnumeric`
`DyNumber`	`numeric`	`pgnumeric`

Таблица соответствия типов `ToPg`

Таблица соответствия типов данных YQL и PostgreSQL при использовании функции `ToPg`:

YQL	PostgreSQL	Название PostgreSQL-типа в YDB
`Bool`	`bool`	`pgbool`
`Int8`	`int2`	`pgint2`
`Uint8`	`int2`	`pgint2`
`Int16`	`int2`	`pgint2`
`Uint16`	`int4`	`pgint4`
`Int32`	`int4`	`pgint4`
`Uint32`	`int8`	`pgint8`
`Int64`	`int8`	`pgint8`
`Uint64`	`numeric`	`pgnumeric`
`Float`	`float4`	`pgfloat4`
`Double`	`float8`	`pgfloat8`
`String`	`bytea`	`pgbytea`
`Utf8`	`text`	`pgtext`
`Yson`	`bytea`	`pgbytea`

`Json`	`json`	`pgjson`
`Uuid`	`uuid`	`pguuid`
`JsonDocument`	`jsonb`	`pgjsonb`
`Date`	`date`	`pgdate`
`Datetime`	`timestamp`	`pgtimestamp`
`Timestamp`	`timestamp`	`pgtimestamp`
`Interval`	`interval`	`pginterval`
`TzDate`	`text`	`pgtext`
`TzDatetime`	`text`	`pgtext`
`TzTimestamp`	`text`	`pgtext`
`Date32`	`date`	`pgdate`
`Datetime64`	`timestamp`	`pgtimestamp`
`Timestamp64`	`timestamp`	`pgtimestamp`
`Interval64`	`interval`	`pginterval`
`TzDate32`	`text`	`pgtext`
`TzDatetime64`	`text`	`pgtext`
`TzTimestamp64`	`text`	`pgtext`
`Decimal`	`numeric`	`pgnumeric`
`DyNumber`	`numeric`	`pgnumeric`

Таблица соответствия типов `FromPg`

Таблица соответствия типов данных PostgreSQL и YQL при использовании функции `FromPg`:

PostgreSQL	YQL
`bool`	`bool`
`int2`	`Int16`
`int4`	`Int32`
`int8`	`Int64`
`float4`	`Float`
`float8`	`Double`
`bytea`	`String`
`varchar`	`Utf8`
`text`	`Utf8`
`cstring`	`Utf8`
`uuid`	`Uuid`
`date`	`Date32`
`timestamp`	`Timestamp64`

Вызов PostgreSQL функций

Чтобы вызвать PostgreSQL функцию, необходимо добавить префикс `Pg::` к ее имени:

```
SELECT
    Pg::extract('isodow'p,PgCast('1961-04-12'p,PgDate)), -- 3pn (среда) - работа с датами до 1970 года
    Pg::generate_series(1p,5p), -- [1p,2p,3p,4p,5p] - для функций-генераторов возвращается ленивый список
;
```

Существует также альтернативный способ вызова функций через встроенную функцию `PgCall(<имя функции>, <операнды>)`:

```
SELECT
    PgCall('lower', 'Test'p), -- 'test'p
;
```

При вызове функции, возвращающей набор `pgrecord`, можно распаковать результат в список структур, используя функцию `PgRangeCall(<имя функции>, <операнды>)`:

```
SELECT * FROM
    AS_TABLE(PgRangeCall("json_each", pgjson('{"a":"foo", "b":"bar"}')));
    --- 'a'p,pgjson('"foo"')
    --- 'b'p,pgjson('"bar"')
;
```

Вызов агрегационных PostgreSQL функций

Чтобы вызвать агрегационную PostgreSQL функцию, необходимо добавить префикс `Pg::` к ее имени:

```
SELECT
Pg::string_agg(x,',','p)
FROM (VALUES ('a'p),('b'p),('c'p)) as a(x); -- 'a,b,c'p

SELECT
Pg::string_agg(x,',','p) OVER (ORDER BY x)
FROM (VALUES ('a'p),('b'p),('c'p)) as a(x); -- 'a'p,'a,b'p,'a,b,c'p
;
```

Также можно использовать агрегационную PostgreSQL функцию для построения фабрики агрегационных функций с последующим применением в `AGGREGATE_BY`:

```
$agg_max = AggregationFactory("Pg::max");

SELECT
AGGREGATE_BY(x,$agg_max)
FROM (VALUES ('a'p),('b'p),('c'p)) as a(x); -- 'c'p

SELECT
AGGREGATE_BY(x,$agg_max) OVER (ORDER BY x),
FROM (VALUES ('a'p),('b'p),('c'p)) as a(x); -- 'a'p,'b'p,'c'p
```

В этом случае вызов `AggregationFactory` принимает только имя функции с префиксом `Pg::`, а все аргументы функции передаются в `AGGREGATE_BY`.

Если в агрегационной функции не один аргумент, а ноль или два и более, необходимо использовать кортеж при вызове `AGGREGATE_BY`:

```
$agg_string_agg = AggregationFactory("Pg::string_agg");

SELECT
AGGREGATE_BY((x,',','p),$agg_string_agg)
FROM (VALUES ('a'p),('b'p),('c'p)) as a(x); -- 'a,b,c'p

SELECT
AGGREGATE_BY((x,',','p),$agg_string_agg) OVER (ORDER BY x)
FROM (VALUES ('a'p),('b'p),('c'p)) as a(x); -- 'a'p,'a,b'p,'a,b,c'p
```

Внимание

Не поддерживается режим `DISTINCT` над аргументами при вызове агрегационных PostgreSQL функций, а также использование `MULTI_AGGREGATE_BY`.

Логические операции

Для выполнения логических операций используются функции `PgAnd`, `PgOr`, `PgNot`:

```
SELECT
    PgAnd(PgBool(true), PgBool(true)), -- PgBool(true)
    PgOr(PgBool(false), null), -- PgCast(null, pgbool)
    PgNot(PgBool(true)), -- PgBool(false)
;
```

Roaring

Введение

Битовые множества, также называемые битовыми картами (bitmaps), обычно используются в качестве быстрых структур данных. К сожалению, они могут потреблять слишком много памяти. Чтобы уменьшить занимаемый объём, применяется сжатие.

Roaring bitmaps — это сжатые битовые карты, которые, как правило, превосходят обычные алгоритмы сжатия битовых карт, такие как WAH, EWAH или Concise. В некоторых случаях Roaring bitmaps могут быть в сотни раз быстрее, а также часто обеспечивают значительно лучшее сжатие. Они могут быть даже быстрее, чем несжатые битовые карты.

Реализация

Работать с Roaring bitmap в YDB можно с помощью набора пользовательских функций UDF в модуле `Roaring`. Они предоставляют возможность работать с 32-битными Roaring bitmap. Для этого данные должны быть сериализованы в формате 32-битных битовых масок, описанном в спецификации. Это можно сделать с помощью функции, доступной в библиотеке Roaring bitmap. Такие библиотеки есть для многих языков, например, для Go.

Сериализованную битовую маску приложение может сохранить в колонке с типом `String`.

Для работы с Roaring bitmap данные из строкового типа необходимо десериализовать в тип Resource<roaring_bitmap>. Для сохранения нужно выполнить обратную операцию. После этого приложение сможет прочитать обновлённую битовую маску из YDB и десериализовать её уже на своей стороне.

Доступные методы

```
Roaring::Deserialize(String{Flags:AutoMap})->Resource<roaring_bitmap>
Roaring::FromUint32List(List<Uint32>{Flags:AutoMap})->Resource<roaring_bitmap>
Roaring::Serialize(Resource<roaring_bitmap>{Flags:AutoMap})->String
Roaring::Uint32List(Resource<roaring_bitmap>{Flags:AutoMap})->List<Uint32>

Roaring::Cardinality(Resource<roaring_bitmap>{Flags:AutoMap})->Uint32

Roaring::Or(Resource<roaring_bitmap>{Flags:AutoMap}, Resource<roaring_bitmap>{Flags:AutoMap})->Resource<roari
ng_bitmap>
Roaring::OrWithBinary(Resource<roaring_bitmap>{Flags:AutoMap}, String{Flags:AutoMap})->Resource<roaring_bitma
p>

Roaring::And(Resource<roaring_bitmap>{Flags:AutoMap}, Resource<roaring_bitmap>{Flags:AutoMap})->Resource<roar
ing_bitmap>
Roaring::AndWithBinary(Resource<roaring_bitmap>{Flags:AutoMap}, String{Flags:AutoMap})->Resource<roaring_bitm
ap>

Roaring::AndNot(Resource<roaring_bitmap>{Flags:AutoMap}, Resource<roaring_bitmap>{Flags:AutoMap})->Resource<r
oaring_bitmap>
Roaring::AndNotWithBinary(Resource<roaring_bitmap>{Flags:AutoMap}, String{Flags:AutoMap})->Resource<roaring_b
itmap>

Roaring::RunOptimize(Resource<roaring_bitmap>{Flags:AutoMap})->Resource<roaring_bitmap>
```

Сериализация и десериализация

Для создания `Resource<roaring_bitmap>` доступны две функции: `Deserialize` и `FromUint32List`. Вторая функция позволяет создавать Roaring bitmap из списков беззнаковых целых чисел, то есть без необходимости использовать библиотечный код Roaring bitmaps для создания бинарного представления.

YDB не сохраняет данные с типом `Resource`, поэтому созданную битовую маску необходимо преобразовать в бинарное представление с помощью метода `Serialize`.

Чтобы использовать полученную битовую маску, например, в условии `WHERE`, предусмотрен метод `Uint32List`, который возвращает список беззнаковых целых чисел из `Resource<roaring_bitmap>`.

Битовые операции

В настоящий момент поддерживаются три модифицирующие бинарные операции с битовыми масками:

- `Or`
- `And`
- `AndNot`

Модифицирующие операции означают, что они изменяют `Resource<roaring_bitmap>`, переданный в первом аргументе. У каждой из этих операций есть версия с суффиксом `WithBinary`, которая позволяет работать с бинарным представлением, не прибегая к его десериализации в тип `Resource<roaring_bitmap>`. Реализация этих методов всё равно выполняет десериализацию для выполнения операции, но не создаёт промежуточный `Resource`, что позволяет экономить ресурсы.

Прочие операции

Для получения кардинальности (количества бит, установленных в 1) предусмотрена функция `Cardinality`.

После построения или модификации битовой маски её можно оптимизировать с помощью метода `RunOptimize`. Это связано с тем, что внутренний формат Roaring bitmap может использовать контейнеры с более эффективным представлением для разных последовательностей бит.

Примеры

```
$b = Roaring::FromUint32List(AsList(42));
$b = Roaring::Or($b, Roaring::FromUint32List(AsList(56)));
```

```
SELECT Roaring::Uint32List($b) AS `Or`; -- [42, 56]
```

```
$b1 = Roaring::FromUint32List(AsList(10, 567, 42));
$b2 = Roaring::FromUint32List(AsList(42));

$b2ser = Roaring::Serialize($b2); -- результат можно сохранить в колонку с типом String

SELECT Roaring::Cardinality(Roaring::AndWithBinary($b1, $b2ser)) AS Cardinality; -- 1

SELECT Roaring::Uint32List(Roaring::And($b1, $b2)) AS `And`; -- [42]
SELECT Roaring::Uint32List(Roaring::AndWithBinary($b1, $b2ser)) AS AndWithBinary; -- [42]
```

```
$b1 = Roaring::FromUint32List(AsList(10, 567, 42));
$b2 = Roaring::FromUint32List(AsList(42));

$b2ser = Roaring::Serialize($b2); -- результат можно сохранить в колонку с типом String

SELECT Roaring::Cardinality(Roaring::AndNotWithBinary($b1, $b2ser)) AS Cardinality; -- 2

SELECT Roaring::Uint32List(Roaring::AndNot($b1, $b2)) AS AndNot; -- [10,567]
SELECT Roaring::Uint32List(Roaring::AndNotWithBinary($b1, $b2ser)) AS AndNotWithBinary; -- [10,567]
```

Re2

Список функций

```
Re2::Grep(pattern:String, options:Struct<...>?) -> (string:String?) -> Bool
Re2::Match(pattern:String, options:Struct<...>?) -> (string:String?) -> Bool
Re2::Capture(pattern:String, options:Struct<...>?) -> (string:String?) -> Struct<_1:String?,foo:String?,...>
Re2::FindAndConsume(pattern:String, options:Struct<...>?) -> (string:String?) -> List<String>
Re2::Replace(pattern:String, options:Struct<...>?) -> (string:String?, replacement:String) -> String?
Re2::Count(pattern:String, options:Struct<...>?) -> (string:String?) -> Uint32
Re2::Options([CaseSensitive:Bool?,DotNl:Bool?,Literal:Bool?,LogErrors:Bool?,LongestMatch:Bool?,MaxMem:Uint6
4?,NeverCapture:Bool?,NeverNl:Bool?,OneLine:Bool?,PerlClasses:Bool?,PosixSyntax:Bool?,Utf8:Bool?,WordBoundar
y:Bool?]) -> Struct<CaseSensitive:Bool,DotNl:Bool,Literal:Bool,LogErrors:Bool,LongestMatch:Bool,MaxMem:Uint6
4,NeverCapture:Bool,NeverNl:Bool,OneLine:Bool,PerlClasses:Bool,PosixSyntax:Bool,Utf8:Bool,WordBoundary:Bool>
```

Модуль Re2 реализует поддержку регулярных выражений на основе google::RE2, где предоставляется широкий ассортимент возможностей (см. официальную документацию).

По умолчанию UTF-8 режим включается автоматически, если регулярное выражение является валидной строкой в кодировке UTF-8, но не является валидной ASCII-строкой. Вручную настройками библиотеки re2 можно управлять с помощью передачи результата функции `Re2::Options` вторым аргументом другим функциям модуля, рядом с регулярным выражением.

Важно

Все обратные слеши в регулярных выражениях (если они записаны в строке с кавычками) нужно удваивать, так как стандартные строковые литералы в SQL рассматриваются как C-escaped строки. Также можно записывать регулярное выражение в форме raw строки `@@regexp@@` — в этом случае удвоение слешей не требуется.

Примеры

```
$value = "xaaxaaxaa";
$options = Re2::Options(false AS CaseSensitive);
$match = Re2::Match("[ax]+\\d");
$grep = Re2::Grep("a.*");
$capture = Re2::Capture(".*(?P<foo>xa?)(a{2,}).*");
$replace = Re2::Replace("x(a+)x");
$count = Re2::Count("a", $options);

SELECT
  $match($value) AS match,               -- false
  $grep($value) AS grep,                 -- true
  $capture($value) AS capture,           -- (_0: 'xaaxaaxaa', _1: 'aa', foo: 'x')
  $capture($value)._1 AS capture_member, -- "aa"
  $replace($value, "b\\1z") AS replace,  -- "baazaaxaa"
  $count($value) AS count;               -- 6
```

Re2::Grep / Re2::Match

Если вынести за скобки детали реализации и синтаксиса регулярных выражений, эти функции полностью аналогичны одноименным функциям из модуля Pire. При прочих равных и отсутствии каких-либо специфических предпочтений мы рекомендуем пользоваться `Pire::Grep / Pire::Match`.

Функцию `Re2::Grep` можно вызвать с помощью выражения `REGEXP` (см. описание базового синтаксиса выражений).

Например, следующие два запроса эквивалентны (в том числе по эффективности вычислений):

- `$grep = Re2::Grep("b+"); SELECT $grep("aaabccc");`
- `SELECT "aaabccc" REGEXP "b+";`

Re2::Capture

В отличие от Pire::Capture в `Re2::Capture` поддерживаются множественные и именованные группы захвата (capturing groups).
Тип результата: структура с полями типа `String?`.

- Каждое поле соответствует группе захвата с соответствующим именем.
- Для неименованных групп генерируются имена вида: `_1`, `_2` и т.д.
- В результат всегда включается поле `_0`, в котором доступна вся совпавшая с регулярным выражением подстрока.

Подробнее про работу со структурами в YQL см. в разделе про контейнеры.

Re2::FindAndConsume

Ищет все вхождения регулярного выражения в переданный текст и возвращает список значений, соответствующих обрамленной в круглые скобки части регулярного выражения для каждого вхождения.

Re2::Replace

Работает следующим образом:

- Во входной строке (первый аргумент) все непересекающиеся подстроки, совпавшие с регулярным выражением, заменяются на указанную строку (второй аргумент).
- В строке с заменой можно использовать содержимое групп захвата (capturing groups) из регулярного выражения с помощью ссылок вида: `\\1`, `\\2` и т.д. Ссылка `\\0` обозначает всю совпавшую с регулярным выражением подстроку.

Re2::Count

Возвращает количество совпавших с регулярным выражением непересекающихся подстрок во входной строке.

Re2::Options

Пояснения к параметрам Re2::Options из официального репозитория

Параметр	По умолчанию	Комментарий
CaseSensitive:Bool?	true	match is case-sensitive (regexp can override with (?i) unless in posix_syntax mode)
DotNl:Bool?	false	let `.` match `\n` (default)
Literal:Bool?	false	interpret string as literal, not regexp
LogErrors:Bool?	true	log syntax and execution errors to ERROR
LongestMatch:Bool?	false	search for longest match, not first match
MaxMem:Uint64?	—	(see below) approx. max memory footprint of RE2
NeverCapture:Bool?	false	parse all parens as non-capturing
NeverNl:Bool?	false	never match \n, even if it is in regexp
PosixSyntax:Bool?	false	restrict regexps to POSIX egrep syntax
Utf8:Bool?	true	text and pattern are UTF-8; otherwise Latin-1
The following options are only consulted when PosixSyntax == true. When PosixSyntax == false, these features are always enabled and cannot be turned off; to perform multi-line matching in that case, begin the regexp with (?m).		
PerlClasses:Bool?	false	allow Perl's \d \s \w \D \S \W
WordBoundary:Bool?	false	allow Perl's \b \B (word boundary and not)
OneLine:Bool?	false	^ and $ only match beginning and end of text

Не рекомендуется Re2::Options использовать в коде. Большинство параметров можно заменить на флаги регулярного выражения.

Пример использования флагов

```
$value = "Foo bar FOO"u;
-- включить режим без учета регистра
$capture = Re2::Capture(@@(?i)(foo)@@);

SELECT
    $capture($value) AS capture; -- ("_0": "Foo", "_1": "Foo")

$capture = Re2::Capture(@@(?i)(?P<foo>FOO).*(?P<bar>bar)@@);

SELECT
    $capture($value) AS capture; -- ("_0": "Foo bar", "bar": "bar", "foo": "Foo")
```

В обоих случаях слово FOO будет найдено. Применение raw строки @@regexp@@ позволяет не удваивать слеши.

String

Функции для работы с ASCII-строками.

Список функций

- `String::Base32Encode(string:String{Flags:AutoMap}) -> String`
- `String::Base32Decode(string:String) -> String?`
- `String::Base32StrictDecode(string:String) -> String?`
- `String::Base64Encode(string:String{Flags:AutoMap}) -> String`
- `String::Base64Decode(string:String) -> String?`
- `String::Base64StrictDecode(string:String) -> String?`
- `String::EscapeC(string:String{Flags:AutoMap}) -> String`
- `String::UnescapeC(string:String{Flags:AutoMap}) -> String`
- `String::HexEncode(string:String{Flags:AutoMap}) -> String`
- `String::HexDecode(string:String) -> String?`
- `String::EncodeHtml(string:String{Flags:AutoMap}) -> String`
- `String::DecodeHtml(string:String{Flags:AutoMap}) -> String`
- `String::CgiEscape(string:String{Flags:AutoMap}) -> String`
- `String::CgiUnescape(string:String{Flags:AutoMap}) -> String`

Кодирует или декодирует строку указанным образом.

Примеры

```
SELECT String::Base64Encode("YQL"); -- "WVFM"
SELECT String::Base64Decode("WVFM"); -- "YQL"
```

`String::Strip(string:String{Flags:AutoMap}) -> String`

Вырезает из строки крайние пробелы.

```
SELECT String::Strip("YQL ");       -- "YQL"
```

- `String::Collapse(string:String{Flags:AutoMap}) -> String`

  Заменяет множественные пробелы внутри строки одиночными.
- `String::CollapseText(string:String{Flags:AutoMap}, limit:Uint64) -> String`

  Укорачивает текст до указанного размера с добавлением троеточия.
- `String::Contains(string:String?, substring:String) -> Bool`

  Проверяет наличие подстроки в строке.
- `String::Find(string:String{Flags:AutoMap}, String, [Uint64?]) -> Int64` - Устаревшая: используйте встроенную функцию Find
- `String::ReverseFind(string:String{Flags:AutoMap}, String, [Uint64?]) -> Int64` - Устаревшая: используйте встроенную функцию RFind
- `String::Substring(string:String{Flags:AutoMap}, [Uint64?, Uint64?]) -> String` - Устаревшая: используйте встроенную функцию Substring
- `String::HasPrefix(string:String?, prefix:String) -> Bool` - Устаревшая: используйте встроенную функцию StartsWith
- `String::StartsWith(string:String?, prefix:String) -> Bool` - Устаревшая: используйте встроенную функцию StartsWith
- `String::HasSuffix(string:String?, suffix:String) -> Bool` - Устаревшая: используйте встроенную функцию EndsWith
- `String::EndsWith(string:String?, suffix:String) -> Bool` - Устаревшая: используйте встроенную функцию EndsWith
- `String::Reverse(string:String?) -> String?` - Устаревшая: используйте Unicode::Reverse

  Устаревшие функции, к использованию не рекомендуются.
- `String::HasPrefixIgnoreCase(string:String?, prefix:String) -> Bool`
- `String::StartsWithIgnoreCase(string:String?, prefix:String) -> Bool`
- `String::HasSuffixIgnoreCase(string:String?, suffix:String) -> Bool`
- `String::EndsWithIgnoreCase(string:String?, suffix:String) -> Bool`

  Проверяют наличие префикса или суффикса в строке без учёта регистра символов.
- `String::AsciiToLower(string:String{Flags:AutoMap}) -> String` - меняет только латинские символы. Для работы с другими алфавитами см. Unicode::ToLower
- `String::AsciiToUpper(string:String{Flags:AutoMap}) -> String` - меняет только латинские символы. Для работы с другими алфавитами см. Unicode::ToUpper
- `String::AsciiToTitle(string:String{Flags:AutoMap}) -> String` - меняет только латинские символы. Для работы с другими алфавитами см. Unicode::ToTitle

  Переводят регистр ascii символов строки к ВЕРХНЕМУ, нижнему или Заглавному виду.
- ```
  String::SplitToList(string:String?, delimeter:String,
  [ DelimeterString:Bool?, SkipEmpty:Bool?, Limit:Uint64? ]) -> List<String>
  ```
  ```

Разбивает строку на подстроки по разделителю.
`string` -- исходная строка
`delimeter` -- разделитель
Именованные параметры:

- `DelimeterString:Bool?` — считать разделитель строкой (true, по умолчанию) или набором символов "любой из" (false)
- `SkipEmpty:Bool?` - пропускать ли пустые строки в результате, по умолчанию false
- `Limit:Uint64?` - ограничение на число извлекаемых компонент, по умолчанию не ограничено; необработанный суффикс оригинальной строки возвращается последним элементом при превышении лимита

```
SELECT String::SplitToList("1,2,3,4,5,6,7", ",", 3 as Limit); -- ["1", "2", "3", "4,5,6,7"]
```

- `String::JoinFromList(strings:List<String>{Flags:AutoMap}, separator:String) -> String`

  Конкатенирует список строк через разделитель в единую строку.
- `String::ToByteList(string:String) -> List<Byte>`

  Разбивает строку на список байт.
- `String::FromByteList(bytes:List<Uint8>) -> String`

  Собирает список байт в строку.
- `String::ReplaceAll(input:String{Flags:AutoMap}, find:String, replacement:String) -> String`
- `String::ReplaceFirst(input:String{Flags:AutoMap}, find:String, replacement:String) -> String`
- `String::ReplaceLast(input:String{Flags:AutoMap}, find:String, replacement:String) -> String`

  Заменяют все/первое/последнее вхождения(е) строки `find` в `input` на `replacement`.
- `String::RemoveAll(input:String{Flags:AutoMap}, symbols:String) -> String`
- `String::RemoveFirst(input:String{Flags:AutoMap}, symbols:String) -> String`
- `String::RemoveLast(input:String{Flags:AutoMap}, symbols:String) -> String`

  Удаляют все/первое/последнее вхождения(е) символа в наборе `symbols` из `input`. Второй аргумент интерпретируется как неупорядоченный набор символов для удаления.
- `String::IsAscii(string:String{Flags:AutoMap}) -> Bool`

  Проверяет, является ли строка валидной ascii последовательностью.
- `String::IsAsciiSpace(string:String{Flags:AutoMap}) -> Bool`
- `String::IsAsciiUpper(string:String{Flags:AutoMap}) -> Bool`
- `String::IsAsciiLower(string:String{Flags:AutoMap}) -> Bool`
- `String::IsAsciiAlpha(string:String{Flags:AutoMap}) -> Bool`
- `String::IsAsciiAlnum(string:String{Flags:AutoMap}) -> Bool`
- `String::IsAsciiHex(string:String{Flags:AutoMap}) -> Bool`

  Проверяют, отвечает ли ascii строка указанному условию.
-

```
String::LevensteinDistance(stringOne:String{Flags:AutoMap},
stringTwo:String{Flags:AutoMap}) -> Uint64
```

Вычисляет расстояние Левенштейна для переданных строк.

- `String::LeftPad(string:String{Flags:AutoMap}, size:Uint64, filler:[String?]) -> String`
- `String::RightPad(string:String{Flags:AutoMap}, size:Uint64, filler:[String?]) -> String`

  Выравнивает текст до указанного размера с дополнением указанным символом или пробелами.
- `String::Hex(value:Uint64{Flags:AutoMap}) -> String`
- `String::SHex(value:Int64{Flags:AutoMap}) -> String`
- `String::Bin(value:Uint64{Flags:AutoMap}) -> String`
- `String::SBin(value:Int64{Flags:AutoMap}) -> String`
- `String::HexText(string:String{Flags:AutoMap}) -> String`
- `String::BinText(string:String{Flags:AutoMap}) -> String`
- `String::HumanReadableDuration(value:Uint64{Flags:AutoMap}) -> String`
- `String::HumanReadableQuantity(value:Uint64{Flags:AutoMap}) -> String`
- `String::HumanReadableBytes(value:Uint64{Flags:AutoMap}) -> String`
- `String::Prec(Double{Flags:AutoMap}, digits:Uint64) -> String`

  Распечатывает значение указанным образом.

Внимание

Функции из библиотеки String не поддерживают кириллицу и умеют работать только с ASCII символами. Для работы со строками в кодировке UTF-8 используйте функции из Unicode.

Unicode

Функции для работы с Unicode строками.

Список функций

- `Unicode::IsUtf(String) -> Bool`

  Проверяет является ли строка валидной utf-8 последовательностью. Например, строка `"\xF0"` не является валидной utf-8 последовательностью, а строка `"\xF0\x9F\x90\xB1"` корректно описывает utf-8 emoji с котиком.

- `Unicode::GetLength(Utf8{Flags:AutoMap}) -> Uint64`

  Возвращает длину utf-8 строки в символах (unicode code points). Суррогатные пары учитываются как один символ.

```
SELECT Unicode::GetLength("жніўня"); -- 6
```

- `Unicode::Find(string:Utf8{Flags:AutoMap}, subString:Utf8, [pos:Uint64?]) -> Uint64?`
- `Unicode::RFind(string:Utf8{Flags:AutoMap}, subString:Utf8, [pos:Uint64?]) -> Uint64?`

  Поиск первого(`RFind` - последнего) вхождения подстроки в строку начиная с позиции `pos`. Возвращает позицию первого символа от найденной подстроки, в случае неуспеха возвращается Null.

```
SE``LECT Unicode::Find("aaa", "bb"); -- Null
```

- `Unicode::Substring(string:Utf8{Flags:AutoMap}, from:Uint64?, len:Uint64?) -> Utf8`

  Возвращает подстроку от `string` начиная с символа `from` длиной в `len` символов. Если аргумент `len` опущен, то подстрока берется до конца исходной строки.
  В случае `from` больше длины исходной строки, возвращается пустая строка `""`

```
SELECT Unicode::Substring("0123456789abcdefghij", 10); -- "abcdefghij"
```

- Функции `Unicode::Normalize...` приводят переданную utf-8 строку в одну из нормальных форм:
  - `Unicode::Normalize(Utf8{Flags:AutoMap}) -> Utf8` -- NFC
  - `Unicode::NormalizeNFD(Utf8{Flags:AutoMap}) -> Utf8`
  - `Unicode::NormalizeNFC(Utf8{Flags:AutoMap}) -> Utf8`
  - `Unicode::NormalizeNFKD(Utf8{Flags:AutoMap}) -> Utf8`
  - `Unicode::NormalizeNFKC(Utf8{Flags:AutoMap}) -> Utf8`
- `Unicode::Translit(string:Utf8{Flags:AutoMap}, [lang:String?]) -> Utf8`

  Транслитерирует в латинский алфавит слова переданной строки, целиком состоящие из символов алфавита языка, переданного вторым аргументом. Если язык не указан, то транслитерация ведется с русского. Доступные языки: "kaz", "rus", "tur", "ukr".

```
SELECT Unicode::Translit("Тот уголок земли, где я провел"); -- "Tot ugolok zemli, gde ya provel"
```

- `Unicode::LevensteinDistance(stringA:Utf8{Flags:AutoMap}, stringB:Utf8{Flags:AutoMap}) -> Uint64`

  Вычисляет расстояние Левенштейна для переданных строк.

-

```
Unicode::Fold(Utf8{Flags:AutoMap},
[ Language:String?, DoLowerCase:Bool?, DoRenyxa:Bool?, DoSimpleCyr:Bool?,
FillOffset:Bool? ]) -> Utf8
```

Выполняет case folding для переданной строки.
Параметры:

- `Language` задается по тем же правилам, что и в `Unicode::Translit()`
- `DoLowerCase` приводит строку к нижнему регистру, по умолчанию `true`
- `DoRenyxa` приводить символы с диокрифами к аналогичным латинским символам, по умолчанию `true`
- `DoSimpleCyr` приводить кирилические символы с диокрифами к аналогичным латинским символам, по умолчанию `true`
- `FillOffset` параметр не используется

```
SELECT Unicode::Fold("Kongreßstraße",  false AS DoSimpleCyr, false AS DoRenyxa); -- "kongressstrasse"
SELECT Unicode::Fold("çypт"); -- "сурт"
SELECT Unicode::Fold("Eylül", "Turkish" AS Language); -- "eylul"
```

- `Unicode::ReplaceAll(input:Utf8{Flags:AutoMap}, find:Utf8, replacement:Utf8) -> Utf8`
- `Unicode::ReplaceFirst(input:Utf8{Flags:AutoMap}, find:Utf8, replacement:Utf8) -> Utf8`
- `Unicode::ReplaceLast(input:Utf8{Flags:AutoMap}, find:Utf8, replacement:Utf8) -> Utf8`

  Заменяет все/первое/последнее вхождения строки `find` в `input` на `replacement`.

- `Unicode::RemoveAll(input:Utf8{Flags:AutoMap}, symbols:Utf8) -> Utf8`
- `Unicode::RemoveFirst(input:Utf8{Flags:AutoMap}, symbols:Utf8) -> Utf8`
- `Unicode::RemoveLast(input:Utf8{Flags:AutoMap}, symbols:Utf8) -> Utf8`

  Удаляются все/первое/последнее вхождения символов в наборе `symbols` из `input`. Второй аргумент интерпретируется как неупорядоченный набор символов для удаления.

```
SELECT Unicode::ReplaceLast("absence", "enc", ""); -- "abse"
SELECT Unicode::RemoveAll("abandon", "an"); -- "bdo"
```

- `Unicode::ToCodePointList(Utf8{Flags:AutoMap}) -> List<Uint32>`

  Разбить строку на unicode'ую последовательность codepoint'ов.
- `Unicode::FromCodePointList(List<Uint32>{Flags:AutoMap}) -> Utf8`

  Сформировать unicode строку из codepoint'ов.

```
SELECT Unicode::ToCodePointList("Щавель"); -- [1065, 1072, 1074, 1077, 1083, 1100]
SELECT Unicode::FromCodePointList(AsList(99,111,100,101,32,112,111,105,110,116,115,32,99,111,110,118,101,114,
116,101,114)); -- "code points converter"
```

- `Unicode::Reverse(Utf8{Flags:AutoMap}) -> Utf8`

  Перевернуть строку.
- `Unicode::ToLower(Utf8{Flags:AutoMap}) -> Utf8`
- `Unicode::ToUpper(Utf8{Flags:AutoMap}) -> Utf8`
- `Unicode::ToTitle(Utf8{Flags:AutoMap}) -> Utf8`

  Привести регистр строки к ВЕРХНЕМУ, нижнему или Заглавному виду.
-

```
Unicode::SplitToList( string:Utf8?, separator:Utf8,
[ DelimeterString:Bool?,  SkipEmpty:Bool?, Limit:Uint64? ]) -> List<Utf8>
```

Разбиение строки на подстроки по разделителю.
`string` -- исходная строка
`separator` -- разделитель
Параметры:

- DelimeterString:Bool? — считать разделитель строкой (true, по умолчанию) или набором символов "любой из" (false)
- SkipEmpty:Bool? - пропускать ли пустые строки в результате, по умолчанию false
- Limit:Uint64? - ограничение на число извлекаемых компонент, по умолчанию не ограничено; необработанный суффикс оригинальной строки возвращается последним элементом при превышении лимита

- `Unicode::JoinFromList(List<Utf8>{Flags:AutoMap}, separator:Utf8) -> Utf8`

  Конкатенация списка строк через `separator` в единую строку.

```
SELECT Unicode::SplitToList("One, two, three, four, five", ", ", 2 AS Limit); -- ["One", "two", "three, four,
five"]
SELECT Unicode::JoinFromList(["One", "two", "three", "four", "five"], ";"); -- "One;two;three;four;five"
```

- `Unicode::ToUint64(string:Utf8{Flags:AutoMap}, [prefix:Uint16?]) -> Uint64`

  Конвертация из строки в число.
  Второй опциональный аргумент задает систему счисления, по умолчанию 0 - автоматическое определение по префиксу.
  Поддерживаемые префиксы : `0x(0X)` - base-16, `0` - base-8. Система по умолчанию - base-10.
  Знак `-` перед числом интерпретируется как в беззнаковой арифметике языка C, например `-0x1` -> UI64_MAX.
  В случае наличия в строке некорректных символов или выхода числа за границы ui64 функция завершается с ошибкой.
- `Unicode::TryToUint64(string:Utf8{Flags:AutoMap}, [prefix:Uint16?]) -> Uint64?`

  Аналогично функции `Unicode::ToUint64()`, но вместо ошибки возвращает `NULL`.

```
SELECT Unicode::ToUint64("77741"); -- 77741
SELECT Unicode::ToUint64("-77741"); -- 18446744073709473875
SELECT Unicode::TryToUint64("asdh831"); -- Null
```

- `Unicode::Strip(string:Utf8{Flags:AutoMap}) -> Utf8`

  Вырезает из строки крайние символы Unicode-категории Space.

```
SELECT Unicode::Strip("\u200bыкль\u2002"u); -- "ыкль"
```

- `Unicode::IsAscii(string:Utf8{Flags:AutoMap}) -> Bool`

  Проверяет, состоит ли utf-8 строка исключительно из символов ascii.
- `Unicode::IsSpace(string:Utf8{Flags:AutoMap}) -> Bool`
- `Unicode::IsUpper(string:Utf8{Flags:AutoMap}) -> Bool`
- `Unicode::IsLower(string:Utf8{Flags:AutoMap}) -> Bool`
- `Unicode::IsAlpha(string:Utf8{Flags:AutoMap}) -> Bool`
- `Unicode::IsAlnum(string:Utf8{Flags:AutoMap}) -> Bool`
- `Unicode::IsHex(string:Utf8{Flags:AutoMap}) -> Bool`

  Проверяют, отвечает ли utf-8 строка указанному условию.
- `Unicode::IsUnicodeSet(string:Utf8{Flags:AutoMap}, unicode_set:Utf8) -> Bool`

  Проверяет, состоит ли utf-8 строка `string` исключительно из символов, указанных в `unicode_set`. Символы в `unicode_set` нужно указывать в квадратных скобках.

```
SELECT Unicode::IsUnicodeSet("вaоaо"u, "[вао]"u); -- true
SELECT Unicode::IsUnicodeSet("вaоaо"u, "[ваб]"u); -- false
```

Url

Normalize

- `Url::Normalize(String) -> String?`

Нормализует URL удобным для Web-роботов образом: приводит hostname в нижний регистр, выкидывает фрагмент и т.п. Результат нормализации зависит только от самого URL. В процессе нормализации **НЕ** выполняются операции, зависящие от внешних данных: приведение по дублям, зеркалам и т.п.

Возвращаемое значение:

- нормализованный URL;
- `NULL`, если переданный строковый аргумент не удалось распарсить как URL.

Примеры

```
SELECT Url::Normalize("hTTp://wWw.yDb.TECH/"); -- "http://www.ydb.tech/"
SELECT Url::Normalize("http://ydb.tech#foo");      -- "http://ydb.tech/"
```

NormalizeWithDefaultHttpScheme

- `Url::NormalizeWithDefaultHttpScheme(String?) -> String?`

Выполняет нормализацию аналогично `Url::Normalize`, но подставляет схему `http://` в случае, если схемы нет.

Возвращаемое значение:

- нормализованный URL;
- исходный URL, если нормализация не удалась.

Примеры

```
SELECT Url::NormalizeWithDefaultHttpScheme("wWw.yDb.TECH");    -- "http://www.ydb.tech/"
SELECT Url::NormalizeWithDefaultHttpScheme("http://ydb.tech#foo"); -- "http://ydb.tech/"
```

Encode / Decode

Кодируют UTF-8 строку в urlencoded формат (`Url::Encode`) и обратно (`Url::Decode`).

Список функций

- `Url::Encode(String?) -> String?`
- `Url::Decode(String?) -> String?`

Примеры

```
SELECT Url::Decode("http://ydb.tech/%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0");
  -- "http://ydb.tech/страница"
SELECT Url::Encode("http://ydb.tech/страница");
  -- "http://ydb.tech/%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0"
```

Parse

Разбирает URL на составные части.

```
Url::Parse(Parse{Flags:AutoMap}) -> Struct< Frag: String?, Host: String?, ParseError: String?, Pass: String?,
Path: String?, Port: String?, Query: String?, Scheme: String?, User: String? >
```

Примеры

```
SELECT Url::Parse(
  "https://en.wikipedia.org/wiki/Isambard_Kingdom_Brunel?s=24&g=h-24#Great_Western_Railway");
/*
(
  "Frag": "Great_Western_Railway",
  "Host": "en.wikipedia.org",
  "ParseError": null,
  "Pass": null,
  "Path": "/wiki/Isambard_Kingdom_Brunel",
  "Port": null,
  "Query": "s=24&g=h-24",
  "Scheme": "https",
  "User": null
)
*/
```

Get...

Получение компонента URL.

Список функций

- `Url::GetScheme(String{Flags:AutoMap}) -> String`
- `Url::GetHost(String?) -> String?`
- `Url::GetHostPort(String?) -> String?`

- `Url::GetSchemeHost(String?) -> String?`
- `Url::GetSchemeHostPort(String?) -> String?`
- `Url::GetPort(String?) -> String?`
- `Url::GetTail(String?) -> String?` -- всё после хоста: path + query + fragment
- `Url::GetPath(String?) -> String?`
- `Url::GetFragment(String?) -> String?`
- `Url::GetCGIParam(String?, String) -> String?` -- второй параметр — имя нужного CGI параметра
- `Url::GetDomain(String?, Uint8) -> String?` -- второй параметр — необходимый уровень домена
- `Url::GetTLD(String{Flags:AutoMap}) -> String`
- `Url::IsKnownTLD(String{Flags:AutoMap}) -> Bool` -- зарегистрирован на iana.org.
- `Url::IsWellKnownTLD(String{Flags:AutoMap}) -> Bool` -- находится в небольшом whitelist из com, net, org, ru и пр.
- `Url::GetDomainLevel(String{Flags:AutoMap}) -> Uint64`
- `Url::GetSignificantDomain(String{Flags:AutoMap}, [List<String>?]) -> String`
  Возвращает домен второго уровня в большинстве случаев и домен третьего уровня для хостеймов вида: ***.XXX.YY, где XXX — одно из com, net, org, co, gov, edu. Этот список можно переопределить через опциональный второй аргумент
- `Url::GetOwner(String{Flags:AutoMap}) -> String`
  Возвращает домен, которым с наибольшей вероятностью владеет отдельный человек или организация. В отличие от Url::GetSignificantDomain работает по специальному разрешительному списку, и помимо доменов из серии ***.co.uk возвращает домен третьего уровня для, например, бесплатных хостингов и блогов, например something.livejournal.com

Примеры

```
SELECT Url::GetScheme("https://ydb.tech");           -- "https://"
SELECT Url::GetDomain("http://www.ydb.tech", 2); -- "ydb.tech"
```

Cut...

- `Url::CutScheme(String?) -> String?`
  Возвращает переданный URL уже без схемы (http://, https:// и т.п.).
- `Url::CutWWW(String?) -> String?`
  Возвращает переданный домен без префикса "www.", если он присутствовал.
- `Url::CutWWW2(String?) -> String?`
  Возвращает переданный домен без префикса "www.", "www2.", "wwww777." и тому подобных, если он присутствовал.
- `Url::CutQueryStringAndFragment(String{Flags:AutoMap}) -> String`
  Возращает копию переданного URL с удаленными всеми CGI параметрами и фрагментами ("?foo=bar" и/или "#baz").

Примеры

```
SELECT Url::CutScheme("http://www.ydb.tech"); -- "www.ydb.tech"
SELECT Url::CutWWW("www.ydb.tech");           -- "ydb.tech"
```

...Punycode...

Преобразования Punycode.

Список функций

- `Url::HostNameToPunycode(String{Flag:AutoMap}) -> String?`
- `Url::ForceHostNameToPunycode(String{Flag:AutoMap}) -> String`
- `Url::PunycodeToHostName(String{Flag:AutoMap}) -> String?`
- `Url::ForcePunycodeToHostName(String{Flag:AutoMap}) -> String`
- `Url::CanBePunycodeHostName(String{Flag:AutoMap}) -> Bool`

Примеры

```
SELECT Url::PunycodeToHostName("xn--80aniges7g.xn--j1aef"); -- "экзампл.ком"
```

...Query...

Преобразования Query.

Список функций

```
Url::QueryStringToList(String{Flag:AutoMap}, [
  KeepBlankValues:Bool?,  -- пустые значения в percent-encoded запросах интерпретируются как пустые строки; п
о умолчанию false
  Strict:Bool?,           -- если false - ошибки парсинга игнорируются, ошибочные поля пропускаются; по умолч
анию true
  MaxFields:Uint32?,      -- максимальное количество полей, при превышении кидается исключение; по умолчанию
Max<Uint32>
  Separator:String?       -- разделитель пар ключ-значение; по умолчанию '&'
]) -> List<Tuple<String, String>>
Url::QueryStringToDict(String{Flag:AutoMap}, [
  KeepBlankValues:Bool?,  -- пустые значения в percent-encoded запросах интерпретируются как пустые строки; п
о умолчанию false
  Strict:Bool?,           -- если false - ошибки парсинга игнорируются, ошибочные поля пропускаются; по умолч
анию true
  MaxFields:Uint32?,      -- максимальное количество полей, при превышении кидается исключение; по умолчанию
Max<Uint32>
  Separator:String?       -- разделитель пар ключ-значение; по умолчанию '&'
]) -> Dict<String, List<String>>
```

```
Url::BuildQueryString(Dict<String, List<String?>>{Flag:AutoMap}, [
  Separator:String?       -- разделитель пар ключ-значение; по умолчанию '&'
]) -> String
Url::BuildQueryString(Dict<String, String?>{Flag:AutoMap}, [
  Separator:String?       -- разделитель пар ключ-значение; по умолчанию '&'
]) -> String
Url::BuildQueryString(List<Tuple<String, String?>>{Flag:AutoMap}, [
  Separator:String?       -- разделитель пар ключ-значение; по умолчанию '&'
]) -> String
```

Примеры

```
SELECT Url::QueryStringToList("a=1&b=2&a=3");                     -- [("a", "1"), ("b", "2"), ("a", "3")]
SELECT Url::QueryStringToDict("a=1&b=2&a=3");                     -- {"b" : ["2"], "a" : ["1", "3"]}
SELECT Url::BuildQueryString([("a", "1"), ("a", "3"), ("b", "2")]); -- "a=1&a=3&b=2"
SELECT Url::BuildQueryString({"a" : "1", "b" : "2"});             -- "b=2&a=1"
SELECT Url::BuildQueryString({"a" : ["1", "3"], "b" : ["2", "4"]}); -- "b=2&b=4&a=1&a=3"
```

Yson

YSON — разработанный в Яндексе формат данных, похожий на JSON.

- Сходства с JSON:
  - не имеет строгой схемы;
  - помимо простых типов данных поддерживает словари и списки в произвольных комбинациях.
- Некоторые отличия от JSON:
  - Помимо текстового представления имеет и бинарное;
  - В текстовом представлении вместо запятых — точки с запятой, а вместо двоеточий — равно;
- Поддерживается концепция «атрибутов», то есть именованных свойств, которые могут быть присвоены узлу в дереве.

Особенности реализации и функциональность модуля:

- Наравне с YSON данный модуль поддерживает и стандартный JSON, что несколько расширяет область его применения.
- Работает с DOM представлением YSON в памяти, которое в терминах YQL передается между функциями как «ресурс» (см. описание специальных типов данных). Большинство функций модуля имеют семантику запроса на выполнение указанной операции с ресурсом и возвращают пустой optional, если операция не удалась из-за несоответствия фактического типа данных ожидаемому.
- Предоставляет несколько основных классов функций (полный список и подробное описание функций см. ниже):
  - `Yson::Parse***` — получение ресурса с DOM-объектом из сериализованных данных, все дальнейшие операции выполняются уже над полученным ресурсом;
  - `Yson::From` — получение ресурса с DOM-объектом из простых типов данных YQL или контейнеров (списков или словарей);
  - `Yson::ConvertTo***` — преобразовать ресурс к простым типам данных или контейнерам;
  - `Yson::Lookup***` — получение одного элемента списка или словаря с опциональным преобразованием в нужный тип данных;
  - `Yson::YPath***` — получение одного элемента дерева документа по указанному относительному пути с опциональным преобразованием в нужный тип данных;
  - `Yson::Serialize***` — получить из ресурса копию его данных, сериализованную в одном из форматов;
- Для удобства при передаче сериализованного Yson и Json в функции, ожидающие на входе ресурс с DOM-объектом, неявное преобразование через `Yson::Parse` или `Yson::ParseJson` происходит автоматически. Также в SQL синтаксисе оператор точки или квадратных скобок автоматически добавляет вызов `Yson::Lookup`. Для сериализации ресурса по-прежнему нужно вызывать `Yson::ConvertTo***` или `Yson::Serialize***`. Таким образом, например, получение элемента "foo" словаря из колонки mycolumn типа Yson в виде строки может выглядеть так: `SELECT Yson::ConvertToString(mycolumn["foo"]) FROM mytable;` или `SELECT Yson::ConvertToString(mycolumn.foo) FROM mytable;`. В варианте с точкой можно экранировать спецсимволы по общим правилам для индентификаторов.

Функции модуля стоит рассматривать как «кубики», из которых можно собирать разные конструкции, например:

- `Yson::Parse*** -> Yson::Serialize***` — конвертация из одного формата в другой;
- `Yson::Parse*** -> Yson::Lookup -> Yson::Serialize***` — извлечение значения указанного поддерева в исходном дереве YSON;
- `Yson::Parse*** -> Yson::ConvertToList -> ListMap -> Yson::Lookup***` — извлечение элементов по ключу из YSON списка.

Примеры

```
$node = Json(@@
  {"abc": {"def": 123, "ghi": "привет"}}
@@);
SELECT Yson::SerializeText($node.abc) AS `yson`;
-- {"def"=123;"ghi"="\xD0\xBF\xD1\x80\xD0\xB8\xD0\xB2\xD0\xB5\xD1\x82"}
```

```
$node = Yson(@@
  <a=z;x=y>[
    {abc=123; def=456};
    {abc=234; xyz=789};
  ]
@@);
$attrs = Yson::YPath($node, "/@");

SELECT
  ListMap(Yson::ConvertToList($node), ($x) -> { return Yson::LookupInt64($x, "abc") }) AS abcs,
  Yson::ConvertToStringDict($attrs) AS attrs,
  Yson::SerializePretty(Yson::Lookup($node, "7", Yson::Options(false AS Strict))) AS miss;

/*
- abcs: `[123; 234]`
- attrs: `{"a"="z";"x"="y"}`
- miss: `NULL`
*/
```

Yson::Parse...

```
Yson::Parse(Yson{Flags:AutoMap}) -> Resource<'Yson2.Node'>
Yson::ParseJson(Json{Flags:AutoMap}) -> Resource<'Yson2.Node'>
Yson::ParseJsonDecodeUtf8(Json{Flags:AutoMap}) -> Resource<'Yson2.Node'>

Yson::Parse(String{Flags:AutoMap}) -> Resource<'Yson2.Node'>? -- принимает YSON в любом формате
Yson::ParseJson(String{Flags:AutoMap}) -> Resource<'Yson2.Node'>?
Yson::ParseJsonDecodeUtf8(String{Flags:AutoMap}) -> Resource<'Yson2.Node'>?
```

Результат всех трёх функций является несериализуемым: его можно только передать на вход другой функции из библиотеки Yson, но нельзя сохранить в таблицу или вернуть на клиент в результате операции — попытка так сделать приведет к ошибке типизации. Также запрещено возвращать его за пределы подзапросов: если это требуется, то надо вызвать Yson::Serialize, а оптимизатор уберёт лишнюю сериализию и десериализацию, если материализация в конечном счёте не потребуется.

Примечание

Функция `Yson::ParseJsonDecodeUtf8` ожидает, что символы, выходящие за пределы ASCII, должны быть дополнительно заэкранированы.

Yson::From

```
Yson::From(T) -> Resource<'Yson2.Node'>
```

`Yson::From` является полиморфной функцией, преобразующей в Yson ресурс большинство примитивных типов данных и контейнеров (списки, словари, кортежи, структуры и т.п.). Тип исходного объекта должен быть совместим с Yson. Например, в ключах словарей допустимы только типы `String` или `Utf8`, а вот `String?` или `Utf8?` уже нет.

Пример

```
SELECT Yson::Serialize(Yson::From(TableRow())) FROM table1;
```

Yson::WithAttributes

```
Yson::WithAttributes(Resource<'Yson2.Node'>{Flags:AutoMap}, Resource<'Yson2.Node'>{Flags:AutoMap}) -> Resourc
e<'Yson2.Node'>?
```

Добавляет к узлу Yson (первый аргумент) атрибуты (второй аргумент). Атрибуты должны представлять из себя узел map.

Yson::Equals

```
Yson::Equals(Resource<'Yson2.Node'>{Flags:AutoMap}, Resource<'Yson2.Node'>{Flags:AutoMap}) -> Bool
```

Проверка деревьев в памяти на равенство, толерантная к исходному формату сериализации и порядку перечисления ключей в словарях.

Yson::GetHash

```
Yson::GetHash(Resource<'Yson2.Node'>{Flags:AutoMap}) -> Uint64
```

Вычисление 64-битного хэша от дерева объектов.

Yson::Is...

```
Yson::IsEntity(Resource<'Yson2.Node'>{Flags:AutoMap}) -> bool
Yson::IsString(Resource<'Yson2.Node'>{Flags:AutoMap}) -> bool
Yson::IsDouble(Resource<'Yson2.Node'>{Flags:AutoMap}) -> bool
Yson::IsUint64(Resource<'Yson2.Node'>{Flags:AutoMap}) -> bool
Yson::IsInt64(Resource<'Yson2.Node'>{Flags:AutoMap}) -> bool
Yson::IsBool(Resource<'Yson2.Node'>{Flags:AutoMap}) -> bool
Yson::IsList(Resource<'Yson2.Node'>{Flags:AutoMap}) -> bool
Yson::IsDict(Resource<'Yson2.Node'>{Flags:AutoMap}) -> bool
```

Проверка, что текущий узел имеет соответствующий тип. Entity это `#`.

Yson::GetLength

```
Yson::GetLength(Resource<'Yson2.Node'>{Flags:AutoMap}) -> Uint64?
```

Получение числа элементов в списке или словаре.

Yson::ConvertTo...

```
Yson::ConvertTo(Resource<'Yson2.Node'>{Flags:AutoMap}, Type<T>) -> T
Yson::ConvertToBool(Resource<'Yson2.Node'>{Flags:AutoMap}) -> Bool?
Yson::ConvertToInt64(Resource<'Yson2.Node'>{Flags:AutoMap}) -> Int64?
Yson::ConvertToUint64(Resource<'Yson2.Node'>{Flags:AutoMap}) -> Uint64?
Yson::ConvertToDouble(Resource<'Yson2.Node'>{Flags:AutoMap}) -> Double?
Yson::ConvertToString(Resource<'Yson2.Node'>{Flags:AutoMap}) -> String?
Yson::ConvertToList(Resource<'Yson2.Node'>{Flags:AutoMap}) -> List<Resource<'Yson2.Node'>>
Yson::ConvertToBoolList(Resource<'Yson2.Node'>{Flags:AutoMap}) -> List<Bool>
Yson::ConvertToInt64List(Resource<'Yson2.Node'>{Flags:AutoMap}) -> List<Int64>
Yson::ConvertToUint64List(Resource<'Yson2.Node'>{Flags:AutoMap}) -> List<Uint64>
Yson::ConvertToDoubleList(Resource<'Yson2.Node'>{Flags:AutoMap}) -> List<Double>
Yson::ConvertToStringList(Resource<'Yson2.Node'>{Flags:AutoMap}) -> List<String>
Yson::ConvertToDict(Resource<'Yson2.Node'>{Flags:AutoMap}) -> Dict<String,Resource<'Yson2.Node'>>
Yson::ConvertToBoolDict(Resource<'Yson2.Node'>{Flags:AutoMap}) -> Dict<String,Bool>
Yson::ConvertToInt64Dict(Resource<'Yson2.Node'>{Flags:AutoMap}) -> Dict<String,Int64>
Yson::ConvertToUint64Dict(Resource<'Yson2.Node'>{Flags:AutoMap}) -> Dict<String,Uint64>
```

```
Yson::ConvertToDoubleDict(Resource<'Yson2.Node'>{Flags:AutoMap}) -> Dict<String,Double>
Yson::ConvertToStringDict(Resource<'Yson2.Node'>{Flags:AutoMap}) -> Dict<String,String>
```

Важно

Данные функции по умолчанию не делают неявного приведения типов, то есть значение в аргументе должно в точности соответствовать вызываемой функции.

`Yson::ConvertTo` является полиморфной функцией, преобразующей Yson ресурс в указанный во втором аргументе тип данных с поддержкой вложенных контейнеров (списки, словари, кортежи, структуры и т.п.).

Пример

```
$data = Yson(@@{
    "name" = "Anya";
    "age" = 15u;
    "params" = {
        "ip" = "95.106.17.32";
        "last_time_on_site" = 0.5;
        "region" = 213;
        "user_agent" = "Mozilla/5.0"
    }
}@@);
SELECT Yson::ConvertTo($data,
    Struct<
        name: String,
        age: Uint32,
        params: Dict<String,Yson>
    >
);
```

Yson::Contains

```
Yson::Contains(Resource<'Yson2.Node'>{Flags:AutoMap}, String) -> Bool?
```

Проверяет наличие ключа в словаре. Если тип объекта map, то ищем среди ключей.
Если тип объекта список, то ключ должен быть десятичным числом - индексом в списке.

Yson::Lookup...

```
Yson::Lookup(Resource<'Yson2.Node'>{Flags:AutoMap}, String) -> Resource<'Yson2.Node'>?
Yson::LookupBool(Resource<'Yson2.Node'>{Flags:AutoMap}, String) -> Bool?
Yson::LookupInt64(Resource<'Yson2.Node'>{Flags:AutoMap}, String) -> Int64?
Yson::LookupUint64(Resource<'Yson2.Node'>{Flags:AutoMap}, String) -> Uint64?
Yson::LookupDouble(Resource<'Yson2.Node'>{Flags:AutoMap}, String) -> Double?
Yson::LookupString(Resource<'Yson2.Node'>{Flags:AutoMap}, String) -> String?
Yson::LookupDict(Resource<'Yson2.Node'>{Flags:AutoMap}, String) -> Dict<String,Resource<'Yson2.Node'>>?
Yson::LookupList(Resource<'Yson2.Node'>{Flags:AutoMap}, String) -> List<Resource<'Yson2.Node'>>?
```

Перечисленные выше функции представляют собой краткую форму записи для типичного сценария использования: `Yson::YPath` — переход в словарь на один уровень с последующим извлечением значения — `Yson::ConvertTo***`. Второй аргумент для всех перечисленных функций — имя ключа в словаре (в отличие от YPath, без префикса `/`) или индекс в списке (например, `7`). Упрощают запрос и дают небольшой выигрыш в скорости работы.

Yson::YPath

```
Yson::YPath(Resource<'Yson2.Node'>{Flags:AutoMap}, String) -> Resource<'Yson2.Node'>?
Yson::YPathBool(Resource<'Yson2.Node'>{Flags:AutoMap}, String) -> Bool?
Yson::YPathInt64(Resource<'Yson2.Node'>{Flags:AutoMap}, String) -> Int64?
Yson::YPathUint64(Resource<'Yson2.Node'>{Flags:AutoMap}, String) -> Uint64?
Yson::YPathDouble(Resource<'Yson2.Node'>{Flags:AutoMap}, String) -> Double?
Yson::YPathString(Resource<'Yson2.Node'>{Flags:AutoMap}, String) -> String?
Yson::YPathDict(Resource<'Yson2.Node'>{Flags:AutoMap}, String) -> Dict<String,Resource<'Yson2.Node'>>?
Yson::YPathList(Resource<'Yson2.Node'>{Flags:AutoMap}, String) -> List<Resource<'Yson2.Node'>>?
```

Позволяет по входному ресурсу и пути на языке YPath получить ресурс, указывающий на соответствующую пути часть исходного ресурса.

Yson::Attributes

```
Yson::Attributes(Resource<'Yson2.Node'>{Flags:AutoMap}) -> Dict<String,Resource<'Yson2.Node'>>
```

Получение всех атрибутов узла в виде словаря.

Yson::Serialize...

```
Yson::Serialize(Resource<'Yson2.Node'>{Flags:AutoMap}) -> Yson -- бинарное представление
Yson::SerializeText(Resource<'Yson2.Node'>{Flags:AutoMap}) -> Yson
Yson::SerializePretty(Resource<'Yson2.Node'>{Flags:AutoMap}) -> Yson -- чтобы увидеть именно текстовый резуль
тат, можно обернуть его в ToBytes(...)
```

Yson::SerializeJson

```
Yson::SerializeJson(Resource<'Yson2.Node'>{Flags:AutoMap}, [Resource<'Yson2.Options'>?, SkipMapEntity:Bool?,
EncodeUtf8:Bool?, WriteNanAsString:Bool?]) -> Json?
```

- `SkipMapEntity` отвечает за сериализацию значений в словарях, имеющих значение `#`. На значение атрибутов флаг не влияет. По умолчанию `false`.
- `EncodeUtf8` отвечает за экранирование символов, выходящих за пределы ASCII. По умолчанию `false`.
- `WriteNanAsString` разрешает сериализацию значений `NaN` и `Inf` в json в виде строк. По умолчанию `false`.

Типы данных `Yson` и `Json`, возвращаемые функциями сериализации, представляет собой частный случай строки, про которую известно, что в ней находятся данные в соответствующем формате (Yson/Json).

Yson::Options

```
Yson::Options([AutoConvert:Bool?, Strict:Bool?]) -> Resource<'Yson2.Options'>
```

Передаётся последним опциональным аргументом (который для краткости не указан) в методы `Parse...`, `ConvertTo...`, `Contains`, `Lookup...` и `YPath...`, которые принимают результат вызова `Yson::Options`. По умолчанию все поля `Yson::Options` выключены (false), а при включении (true) модифицируют поведение следующим образом:

- **AutoConvert** — если переданное в Yson значение не в точности соответствует типу данных результата, то значение будет по возможности сконвертировано. Например, `Yson::ConvertToInt64` в этом режиме будет делать Int64 даже из чисел типа Double.
- **Strict** — по умолчанию все функции из библиотеки Yson возвращают ошибку в случае проблем в ходе выполнения запроса (например, попытка парсинга строки не являющейся Yson/Json, или попытка поиска по ключу в скалярном типе, или запрошено преобразование в несовместимый тип данных, и т.п.), а если отключить строгий режим, то вместо ошибки в большинстве случаев будет возвращаться `NULL`. При преобразовании в словарь или список (`ConvertTo<Type>Dict` или `ConvertTo<Type>List`) плохие элементы будут выброшены из полученной коллекции.

Пример

```
$yson = @@{y = true; x = 5.5}@@y;
SELECT Yson::LookupBool($yson, "z"); --- null
SELECT Yson::LookupBool($yson, "y"); --- true

SELECT Yson::LookupInt64($yson, "x"); --- Ошибка
SELECT Yson::LookupInt64($yson, "x", Yson::Options(false as Strict)); --- null
SELECT Yson::LookupInt64($yson, "x", Yson::Options(true as AutoConvert)); --- 5

SELECT Yson::ConvertToBoolDict($yson); --- Ошибка
SELECT Yson::ConvertToBoolDict($yson, Yson::Options(false as Strict)); --- { "y": true }
SELECT Yson::ConvertToDoubleDict($yson, Yson::Options(false as Strict)); --- { "x": 5.5 }
```

Если во всём запросе требуется применять одинаковые значения настроек библиотеки Yson, то удобнее воспользоваться PRAGMA yson.AutoConvert; и/или PRAGMA yson.Strict;. Также эти `PRAGMA` являются единственным способом повлиять на неявные вызовы библиотеки Yson, которые возникают при работе с типами данных Yson/Json.

Смотрите также

- Доступ к значениям в JSON с помощью YQL
- Изменение JSON с помощью YQL

Структура планов запросов

Чтобы лучше понимать работу запроса, вы можете получить и проанализировать его план.
Структура плана запроса в YDB представляет собой граф, где каждый узел содержит информацию об операциях и таблицах.

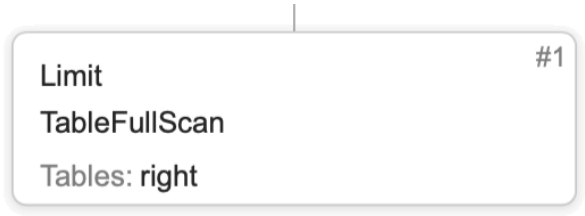
Ниже представлена справочная информация по типам узлов, а пример разбора плана запроса можно посмотреть в разделе План выполнения запроса.

Типы узлов

Stage

Стадия выполнения запроса.

Визуальное представление:



Стадия может содержать следующие операции:

TableFullScan

Полное сканирование таблицы. Важно иметь ввиду, что ресурсоёмкость данной операции пропорциональна размеру таблицы, поэтому по возможности ее стоит избегать.

Атрибут	Значение
Table	имя таблицы
ReadColumns	список читаемых колонок
ReadLimit	лимит на число прочитанных строк
Reverse	флаг, указывающий на порядок, в котором будут прочитаны строки, по умолчанию порядок прямой (от меньшего к большему), но в случае выставления флага в `true` порядок чтения будет обратным
Parallel	флаг, указывающий на то что чтение строк будет происходить паралленьно по шардам

TableRangeScan

Чтение строк таблицы по определённому диапазону значений первичного ключа.

Атрибут	Значение
Table	имя таблицы
ReadColumns	список читаемых колонок
ReadRange	диапазон ключей, по которому выполняется чтение
ReadLimit	лимит на число прочитанных строк
Reverse	флаг, указывающий на порядок, в котором будут прочитаны строки, по умолчанию порядок прямой (от меньшего к большему), но в случае выставления флага в `true` порядок чтения будет обратным
Parallel	флаг, указывающий на то что чтение строк будет происходить паралленьно по шардам

TablePointLookup

> **Важно**
>
> Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Чтение строк таблицы по конкретным значениям первичного ключа. Обратите внимание, что для этой операции необходимо указать все компоненты первичного ключа, чтение по префиксу ключа выполняется как `TableRangeScan`.

Атрибут	Значение
Table	имя таблицы
ReadColumns	список читаемых колонок

Upsert

Перезапись строк таблицы по соответствующим значениям первичного ключа, если они существовали, иначе добавление новых строк.

Атрибут	Значение
Table	имя таблицы

Columns	колонки, которые содержит строка

Delete

Удаление строк из таблицы.

Атрибут	Значение
Table	имя таблицы

Join

Объединение двух таблиц, в описании операции указана используемая стратегия (JoinDict или MapJoin).

- JoinDict - строятся словари для правой и левой частей, а затем выполняется их объединение по ключам
- MapJoin - для каждого элемента из левой части выполняется поиск по ключу в предварительно построенном словаре для правой части

Filter

Фильтрация строк, в результате остаются только те строки, для которых предикат оказался true.

Атрибут	Значение
Predicate	условие фильтрации
Limit	лимит на число строк

Aggregate

Группировка строк по значениям колонок с построением агрегатов.

Атрибут	Значение
GroupBy	колонки, по которым выполняется группировка
Aggregation	агрегатные функции

Sort

Сортировка строк.

Атрибут	Значение
SortBy	колонки, по которым выполняется сортировка

TopSort

Сортировка строк с применением лимита.

Атрибут	Значение
TopSortBy	колонки, по которым выполняется сортировка
Limit	лимит на число строк

Top

Взятие N элементов таким образом, что все они меньше или равны N+1 элемента, если бы вся последовательность была отсортирована.

Атрибут	Значение
TopBy	колонки, по которым будут взяты первые N строк
Limit	лимит на число строк

Limit

Ограничение на число строк.

Атрибут	Значение
Limit	величина лимита

Offset

Смещение, позволяющее пропустить первые N элементов заданного набора строк.

Атрибут	Значение
Offset	величина смещения

Union

Объединение результатов двух и более подзапросов в один набор.

Iterator

Итератор по заданному набору строк, как правило в качестве аргумента принимает precompute.

PartitionByKey

Партиционирование по ключу, как правило в качестве аргумента принимает precompute.

Connection

Зависимость по данным между стадиями.

Визуальное представление:



Каждая стадия исполняется в виде некоторого количества тасков. Например, читающая таблицу стадия может исполняться в N тасков, где N - это количество шардов таблицы. От того, какой connection используется, зависит как именно данные будут передаваться между стадиями. Далее будут описаны все возможные типы connection'ов, для их описания удобно ввести термины producer stage - стадия, отправляющая данные и consumer stage - стадия, принимающая данные.

UnionAll

Объединяет результаты всех тасков producer stage и отдаёт их как единый результат в единственный таск consumer stage.

Merge

Частный случай `UnionAll`, когда результаты producer stage отсортированы по заданному набору колонок, результат также отсортирован.

Broadcast

Рассылает результат единственного таска producer stage на все таски consumer stage.

Map

Реализует связи тасков стадий 1-к-1, то есть producer stage и consumer stage должны иметь одинаковое количество тасков.

HashShuffle

Рассылает результаты с тасков producer stage на таски consumer stage по некоторому правилу для заданных колонок. Правило жёстко описано в коде, но вот список колонок указывается в каждом connection'е независимо.

ResultSet

Результат выполнения запроса, возвращаемый клиенту.

Визуальное представление:



Precompute

Материализованный в памяти промежуточный результат.

Визуальное представление:



Стадии, использующие precompute, начинают выполняться только после завершения вычисления precompute.

Использование YDB UI

YDB предоставляет инструменты для мониторинга и определения состояния системы:

- YDB Monitoring — основной обозреватель кластера. Показывает состояние узлов, групп хранения.
- Hive web-viewer — интерфейс для взаимодействия с Hive.
- Interconnect overview — состояние соединений узлов кластера друг с другом.
- Логи — каждый компонент YDB пишет сообщения в логи разного уровня. По ним можно детектировать критические проблемы или разобраться в причинах неполадок.
- Графики — YDB собирает множество метрик, по которым можно определить состояние системы либо конкретного компонента.

Использование YDB UI

YDB предоставляет инструменты для мониторинга и определения состояния системы:

- YDB Monitoring — основной обозреватель кластера. Показывает состояние узлов, групп хранения.
- Hive web-viewer — интерфейс для взаимодействия с Hive.
- Interconnect overview — состояние соединений узлов кластера друг с другом.
- Логи — каждый компонент YDB пишет сообщения в логи разного уровня. По ним можно детектировать критические проблемы или разобраться в причинах неполадок.
- Графики — YDB собирает множество метрик, по которым можно определить состояние системы либо конкретного компонента.

YDB Monitoring

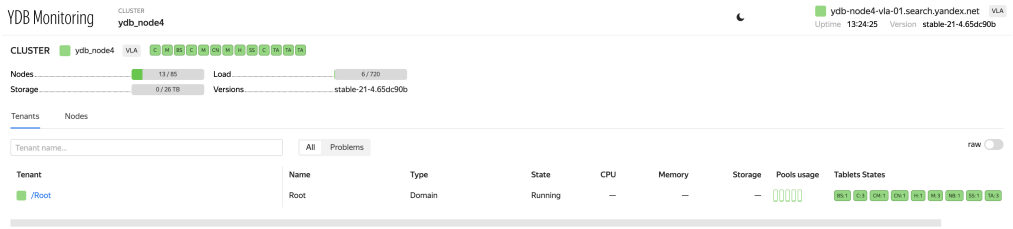
YDB Monitoring представлен набором web-страниц, отображающих состояние системы в разрезе по нескольким аспектам. На страницах приведены перечни компонент и их мгновенные параметры. Для многих компоненты приведен цветовой индикатор состояния перед названием.

Главная страница

Страница доступна по адресу:

```
http://<ендпоинт>:8765/monitoring/cluster
```

Примерный внешний вид страницы показан на скриншоте ниже.



В правом верхнем углу страницы расположена информация об узле сформировавшем текущую страницу:

- хост;
- время работы узла;
- версия ydb исполняемая на узле.

Ниже расположена краткая сводка информации о кластере:

- имя кластера, перечень запущенных таблеток;
- **Nodes** — количество узлов;
- **Load** — суммарная загрузка CPU на всех узлах;
- **Storage** — занято/всего доступно места на дисках;
- **Versions** — перечень версий YDB выполняемых на узлах кластера.

Далее следуют списки тенантов и узлов, которые переключаются вкладками Tenants и Nodes соответственно.

Список тенантов

YDB — мультиарендная СУБД, позволяющая изолировано обрабатывать запросы к базам различных пользователей, называемых тенантами. Каждая база данных принадлежит одному тенанту.

В списке тенантов для каждого тенанта приведены:

- **Tenant** — путь тенанта;
- **Name** — имя тенанта;
- **Type** — тип тенанта;
- **State** — состояние;
- **CPU** — загрузка CPU узлами тенанта;
- **Memory** — потребление ОЗУ узлами тенанта;
- **Storage** — оценка объема данных, хранимых тенантом;
- **Pools usage** — потребление CPU узлами в разбивке по внутренним пулам потоков;
- **Tablets States** — таблетки работающие в данном тенанте.

Тенант типа `Domain` обслуживает системные компоненты, необходимые для обеспечения работы всех тенантов. В него входят все storage узлы и системные таблетки. Тенант типа `Dedicated` выполняет обслуживание базы данных.

При нажатии на путь тенанта можно перейти на страницу тенанта.

Список узлов

В списке указаны все узлы кластера. Для каждого узла приведены:

- # — ID узла;
- **Host** — хост на котором располагается узел;
- **Endpoints** — прослушиваемые порты;
- **Version** — выполняемая версия YDB;
- **Uptime** — время работы узла;
- **Memory used** — используемый объем ОЗУ;
- **Memory limit** — установленное через cgroup ограничение потребления ОЗУ;
- **Pools usage** — потребление CPU в разбивке по внутренним пулам потоков;
- **Load average** — средняя загрузка хоста по CPU.

При нажатии на имя хоста, можно перейти на страницу узла.

Страница узла

Страница доступна по адресу:

```
http://<ендпоинт>:8765/monitoring/node/<node-id>/
```

Информация об узле представлена следующими блоками:

- **Pools** — потребление CPU в разбивке по внутренним пулам потоков, примерное предназначение пулов:

    - **System** — задачи критически важных системных компонентов;
    - **User** — пользовательские задачи, выполнение запросов таблетками;
    - **Batch** — длительные фоновые задачи;
    - **IO** — выполнение блокирующих операций ввода-вывода;
    - **IC** — обработка сетевого взаимодействия.

  Высокая загрузка пулов может быть причиной деградации производительности и увеличения времени отклика системы.
- **Common info** — базовая информация об узле:
    - **Version** — версия YDB;
    - **Uptime** — время работы узла;
    - **DC** — зона доступности в которой расположен узел;
    - **Rack** — идентификатор стойки в которой располагается узел.
- **Load average** — средняя загрузка CPU хоста за разные интервалы времени:
    - 1 минута;
    - 5 минут;
    - 15 минут.

На странице узла есть вкладки, Storage и Tablets содержащие список групп хранения и список таблеток соответственно.

Список групп хранения на узле

Для storage узлов список содержит группы хранения, хранящие данные на диске на данном узле. Для dynamic узлов показаны группы хранения принадлежащие тенанту к которому относится узел.

Группы хранения сгруппированы по пулам хранения. Каждый пул можно развернуть в список групп.

Для каждой группа приведены:

- ID группы хранения;
- индикатор быстродействия группы;
- количество VDisk'ов в группе;
- топология хранения данных.

Каждую группу хранения также можно раскрыть в список VDisk'ов, где для каждого VDisk'а приведены:

- VDiskID;
- уникальный в пределах узла PDiskID, на котором находится VDisk;
- ID узла, на котором находится VDiks;
- свободное/доступное место на блочном устройстве;
- путь, используемый для доступа к блочному устройству.

Данный список можно использовать для идентификации групп хранения на которые оказали влияние выход из строя дисков или узлов.

Список таблеток на узле

Многие компоненты YDB выполнены в виде таблеток. Система может перемещать таблетки между узлами. На любом из узлов может выполняться некоторое количество таблеток.

В верхней части списка таблеток расположен крупный индикатор работы таблеток на данном узле. Он показывает долю полностью запущенных и работающих таблеток.

Ниже индикатора располагается список таблеток, где каждая таблетка представлена маленьким значком цветового индикатора. При наведении на индикатор отображается краткая информация о таблетке:

- **Tablet** — ID таблетки;
- **NodeID** — ID узла, на котором работает таблетка;
- **State** — состояние таблетки;
- **Type** — тип таблетки;
- **Uptime** — время работы с момента текущего запуска таблетки;
- **Generation** — поколение (номер текущей попытки запуска) таблетки.

Страница тенанта

```
http://<ендпоинт>:8765/monitoring/tenant/healthcheck?name=<tenant-path>
```

Так же как и предыдущие страницы, данная страница содержит краткую информацию о тенанте, но в отличии от других страниц этот блок изначально свернут.

В блоке `Tenant Info` представлена следующая информация:

- **Pools** — суммарное потребление CPU узлами тенанта в разбивке по внутренним пулам потоков (подробнее о пулах см. страницу тенанта);
- **Metrics** — Информация о потреблении таблеток данного тенанта:
    - **Memory** — используемая таблетками память;
    - **CPU** — загрузка таблетками CPU;
    - **Storage** — объем хранимых таблетками данных;
    - **Network** — примерный объем передаваемых между узлами данных;
    - **Read throughput** — поток чтений создававемый таблетками тенанта;
    - **Write throughput** — поток записи создававемый таблетками тенанта.
- **Tablets/running** — количество работающих таблеток.

На странице тенанта также имеются следующие вкладки:

- **HealthCheck** — отчет о проблемах кластера при их наличии;
- **Storage** — список групп хранения, содержит информацию о том, какие VDisk'и на каких узлах и устройствах работают;

- **Compute** — список узлов, содержит узлы и таблетки запущенные на них;
- **Schema** — схема тенанта, позвляет просматривать таблицы, выполнять YQL запросы, проосматривать перечень самых медленные запросов и самых нагруженных шардов;
- **Network** — состояние сети кластера.

Список групп хранения в тенанте

Данная страница по аналогии со списком групп на ноде показывает список групп хранения заданного тенанта. Группы хранения сгруппированы в списке по пулам хранения. Каждый пул можно развенуть в список групп.

Список узлов принадлежащих тенанту

Здесь отображен список узлов принадлежадих текущему тенанту. В случае если тенант имеет тип domain, то в список входят storage узлы.

Каждый узел представлен следующими параметрами:

- # — ID узла;
- **Host** — хост на котором располагается узел;
- **Uptime** — время работы узла;
- **Endpoints** — прослушиваемые порты;
- **Version** — выполняемая версия YDB;
- **Pools usage** — потребление CPU в разбивке по внутренним пулам потоков;
- **CPU** — загрузка CPU таблетками;
- **Memory** — потребление ОЗУ таблетками;
- **Network** — загрузка сети таблетками;
- **Tablets** — список сгруппированных по типу таблеток запущенных на узле;

Схема тенанта

В левой части страницы показана схема объектов тенанта, позволяющая выбирать конкретные таблицы и просматривать информацию о них.

В правой части расположены вкладки:

- **Info** — информация о таблице и ее схема;
- **Preview** — превью первых элементов таблицы;
- **Graph** — различные встроенные графики с информацией о состоянии таблицы;
- **Describe** — результат выполнения соответствующей команды;
- **Query** — форма выполнения YQL запросов;
- **Tablets** — перечень таблеток обслуживающих таблицу;
- **ACL** — Access Control List, список конроля доступа;
- **Top queries** — топ запросов к таблице по времени выполнения;
- **Top shards** — топ загруженных шардов таблицы.

Состояние сети

В левой части страницы представлены узлы тенанта в виде квадратов индикаторов состояний.

При выборе любого узла в правой части экрана отобразится информация о состоянии сетевого соединения между этим и другими узлами.

При отметке чекбоксов ID и Racks будут также отображены ID узлов и их расположение по стойкам.

Мониторинг статической группы

Чтобы проверить статус статической группы, перейдите на панель **Storage**. По умолчанию отображается список групп, имеющих какие-либо проблемы.

В поисковой строке введите `static`. Если результат пуст, то статическая группа не имеет проблем. Если же в выдаче присутствует группа **static**, то необходимо проверить статус VDisk'ов в ней. Допустимы зеленый (проблем нет) и синий (VDisk реплицируется) состояния. Красный индикатор сигнализирует о проблеме. Наведите курсор на индикатор чтобы получить текстовое описание проблемы.

Индикаторы состояний

Слева от названия конкретного компонента может находится цветовой индикатор его состояния.

Цвета индикаторов имеют следующее значение:

- **Зеленый** — проблем нет, компонент работает в штатном режиме;
- **Синий** — происходит репликация данных, других проблем нет;
- **Желтый** — возможны проблемы, компонент продолжает работать;
- **Красный** — имеются критические проблемы, компонент не работает или работает с ограничениями.

Если компонент включает в себя другие компоненты, то в случае отсутствия собственных проблем, его состояние определяется путем агрегирования состояний его частей.

Hive web-viewer

Web-viewer Hive предоставляет визуальный интерфейс работы с Hive.

Попасть на страницу Web-viewer Hive можно по ссылке Developer UI из вкладки Tablets в YDB UI. Для доступа к странице необходимо быть аутентифицированным как пользователь, входящий в список `monitoring_allowed_sids` в конфигурации безопасности кластера.

Главная страница

Главная страница представляет информацию о распределении и потреблении ресурсов таблетками на каждом из узлов, представленную в виде таблицы.

Перед таблицей представлена краткая информация:

- **Info**:
  - **Tenant** — тенант, за который отвечает Hive. Для Hive конкрентой базы здесь указано имя этой базы, а для корневого Hive — имя корневого домена кластера (по умолчанию `/Root`).
  - **Nodes** — количество узлов, подключённых к Hive.
  - **Tablets** — процентное и количественное соотношение запущенных в данный момент таблеток к существующимм всего.
  - **Boot Queue** — количество таблеток, которые находятся в очереди на запуск.
  - **Wait Queue** — количество таблеток, которые не имеют возможности подняться, поскольку ни один из узлов не подходит для их запуска.
- **Totals** — суммарное потребление ресурсов таблетками (cnt, cpu, mem, net).
- **Variance** — среднеквадратическое отклонение потребления ресурсов (cnt, cpu, mem, net) между узлами.
- **Triggers** — значения показателей, которые используются для принятия решения об автобалансировке. Каждый сопровождается цветовым индикатором: зелёный означает, что значение находится на приемлемом уровне; жёлтый — что оно близко к порогу; красный — что значение превысило порог, и сейчас должна быть запущена автобалансировка.
  - **Counter**, **CPU**, **Memory**, **Network** — дисбаланс в потреблении соответствующего ресурса между узлами.
  - **MaxUsage** — максимальное потребление ресурса на одном узле.
  - **Imbalance** — дисбаланс в распределении по узлам таблеток одного объекта. Применяется только для таблеток без метрик.
- Статистика о числе запусков автобалансировки. Обнуляется при рестарте Hive. Время с последнего рестарта Hive можно увидеть на вкладке Tablets в YDB UI,

После идет таблица, где каждая строка представляет из себя один из узлов под управлением Hive и имеет следующие столбцы:

- **Node** — номер узла.
- **Name** — fqdn узла и ic—port.
- **DC** — датацентр, в котором расположен узел.
- **Domain** — тенант узла.
- **Uptime** — время, прошедшее с начала работы узла.
- **Unknown** — количество таблеток в неопределенном состоянии.
- **Starting** — количество таблеток в процессе подъема.
- **Running** — количество работающих таблеток.
- **Types** — распределение таблеток по их типу. С помощью нажатия на конкретный тип можно запретить запуск таблеток данного типа на конкретном узле. Нажатием на тип в заголовке таблицы можно установить такой запрет на все узлы по умолчанию.
- **Usage** — общее потребление ресурсов узла.
- **Resources** :
  - **cnt** — фиктивный ресурс Counter.
  - **cpu** — потребление ядра таблетками.
  - **mem** — потребление ОЗУ таблетками.
  - **net** — потребление полосы таблетками.
- **Active** — включение/отключение узла для перевоза таблеток на данный узел.
- **Freeze** — запрет для таблеток, запущенных на этом узле, подниматься на других узлах.
- **Kick** — перевоз всех таблеток разом с узла.
- **Drain** — плавный перевоз всех таблеток с узла — с ограниченным in-flight в отличие от Kick.

Ниже таблицы представлены дополнительные страницы:

- **Bad tablets** — список таблеток, имеющих проблемы или ошибки.
- **Heavy tablets** — список из 1000 таблеток, потребляющих наибольшее количество ресурсов.
- **Waiting tablets** — список таблеток без возможности подняться.
- **Resources** — потребление ресурсов каждой таблеткой.
- **Tenants** — список тенантов с указанием их локальных Hive таблеток.
- **Balancer** — меню для ручного запуска автобалансировки.
- **Operations log** — лог ручных изменений конфигураций Hive через viewer.
- **Nodes** — список узлов.
- **Storage** — список пулов групп хранения.
- **Groups** — список групп хранения для каждой таблетки.
- **Settings** — страница конфигураций Hive.
- **Reassign Groups** — страница перераспределения групп хранения между таблетками.
- **SubActors** — список запущенных в данный момент вспомогательных акторов.

Также можно посмотреть, какие таблетки используют конкретную группу, и наоборот, какие группы используются в конкретной таблетке.

Reassign Groups

При добавлении в базу данных новых групп хранения существующие таблетки продолжают писать в старые группы. Для таких ситуаций в интерфейсе предусмотрена возможность запуска перераспределения таблеток по группам хранения. При нажатии на кнопку **Reassign Groups** появится окно с параметрами для балансировки:

- **Storage pool** — пул групп хранения для балансировки.
- **Storage group** — если предыдущий пункт не указан, можно указать отдельно только одну группу.
- **Type** — тип таблеток, для которых будет производиться балансировка.
- **Channels** — диапазон каналов, для которых будет производиться балансировка.
- **Percent** — процент от общего количества каналов таблеток, которые переедут в результате балансировки.
- **Inflight** — количество таблеток, одновременно переезжающих на другие группы.

После указания всех параметров, следует нажать сначала "Query", который покажет количество каналов, попавших под переезд, и разблокирует кнопку "Reassign".
При нажатии которой начнется перераспределение.

Balancer

При нажатии на кнопку **Balancer** появится окно со следующими элементами:

- **Run Balancer** — ручной запуск балансировки таблеток по узлам с заданным ограничением на число перемещений таблеток.
- **Rebalance ALL tablets FROM SCRATCH** — перезапуск сразу всех таблеток базы. Позволяет быстро получить равномерное распределение таблеток ценой кратковременной недоступности базы. *Не рекомендуется к использованию в production-окружении.*
- **Latest tablet moves** — последние 5 перемещённых таблеток.

Settings

На странице **Settings** представлен список параметров конфигурации Hive, которыми можно управлять через интерфейс. Для каждого параметра указано три значения: текущее значение, значение из конфигурации YDB и значение по умолчанию. Параметры, значения которых выставлены через интерфейс, выделены полужирным шрифтом. Выставленные через интерфейс значения имеют приоритет над значениями из конфигурации YDB. Внесение изменений в настройки через интерфейс не изменяет значения в конфигурации. С описанием параметров можно ознакомиться в описании конфигурации.

Также на этой странице представлена таблица типов таблеток и видов метрик, с помощью которой можно управлять тем, какие метрики учитываются для каких типов таблеток.

Interconnect Overview

Обзор соединений узлов кластера друг с другом. Доступен по адресу:

```
http://<ендпоинт>:8765/actors/interconnect/overview
```

Показывает для каждого другого узла:

- пинг
- разницу системных часов
- наличие соединения
- последняя зарегистрированная ошибка

Логи

Уровни логирования

Уровень	Числовое значение	Значение
TRACE	8	Очень детальная отладочная информация.
DEBUG	7	Отладочная информация для разработчиков.
INFO	6	Отладочная информация для сбора статистики.
NOTICE	5	Произошло существенное для системы или пользователя событие.
WARN	4	Предупреждение, если носит не временный характер то стоит реагировать и исправлять.
ERROR	3	Некритическая ошибка.
CRIT	2	Критическое состояние.
ALERT	1	Возможна деградация системы, выход из строя компонентов системы.
EMERG	0	Возможен выход из строя системы, отказ кластера.

Уровень логирования различных компонентов YDB может настраиваться раздельно. Для каждого компонента может быть применено либо явно заданное значение, либо значение уровня логирования по умолчанию. Значение уровня логирования по умолчанию также может быть изменено.

Изменение уровня логирования

Чтобы изменить уровень логирования:

1. Перейти по ссылке вида

   ```
   http://<endpoint>:8765/cms
   ```

   Откроется страница `Cluster Management System`.
2. На вкладке **Configs** кликнуть на строку `LogConfigItems`. Появится кнопка `Create new item` и перечень уже созданных элементов конфигурации.
3. Нажать на кнопку `Create new item` для создания нового элемента конфигурации (либо нажать на кнопку с изображением карандаша чтобы отредактировать созданный ранее элемент).
4. Для изменения уровня логирования по умолчанию выбрать желаемый уровень логирования из выпадающего списка `Level` в разделе `Default log settings`. Глобальная настройка по умолчанию - `NOTICE`.
5. Для изменения уровня логирования отдельных компонентов предназначена таблица в разделе `Component log settings`. В строке с именем компонента уровень логирования которого требуется изменить в колонке `Component` выбрать из выпадающего списка в колонке `Log level` желаемый уровень логирования.
6. Сохранить изменения нажатием кнопки `Submit`

Графики

Просмотр графиков осуществляется через Grafana.

Основные метрики системы выведены на дашборд:

- **CPU Usage** — суммарная загрузка CPU на всех узлах (1 000 000 = 1 CPU);
- **Memory Usage** — потребление оперативной памяти по узлам;
- **Disk Space Usage** — потребление дискового пространства по узлам;
- **SelfPing** — наибольшее за интервал измерений фактическое время доставки отложенных сообщений в актор-системе. Измеряется для сообщений с отложенной на 10 мс доставкой. Рост данной величины может быть признаком микроберстов нагрузки, высокой утилизации CPU, вытеснения процесса YDB с ядер процессора другими процессами.

Интеграции YDB

В данном разделе приведена основная информация про интеграции YDB со сторонними системами.

> **Примечание**
>
> В дополнение к своему собственному нативному протоколу, YDB обладает слоем совместимости, что позволяет внешним системам подключаться к базам данных по сетевому протоколу Apache Kafka. Благодаря слою совместимости, множество инструментов, разработанных для работы с Kafka, могут также взаимодействовать с YDB. Уровень совместимости каждого конкретного приложения необходимо уточнять отдельно.

Графические пользовательские интерфейсы

Среда	Инструкция	Уровень поддержки
YDB UI	Справка	
DBeaver	Инструкция	С помощью JDBC-драйвера
JetBrains Database viewer	—	С помощью JDBC-драйвера
JetBrains DataGrip	Инструкция	С помощью JDBC-драйвера
Другие JDBC-совместимые IDE	—	С помощью JDBC-драйвера
Visual Studio Code	Инструкция	С помощью YDB for VS Code
Jupyter Notebook	Инструкция	С помощью YDB-SQLAlchemy

Визуализация данных (Business Intelligence, BI)

Среда	Уровень поддержки	Инструкция
Apache Superset	ydb-sqlalchemy	Инструкция
DataLens	Полный	Инструкция
Grafana	Полный	Инструкция

Оркестрация

Среда	Инструкция
Apache Airflow™	Инструкция

Поставка данных

Система поставки	Инструкция
FluentBit	Инструкция
LogStash	Инструкция
Kafka Connect Sink	Инструкция

Потоковая поставка данных

Система поставки	Инструкция
Apache Kafka API	Инструкция
Apache Kafka Connect	Инструкция

Миграции данных

Источник	Инструкция
Произвольные JDBC-источники данных	Инструкция
MySQL	Инструкция

Миграции схемы

Среда	Инструкция
goose	Инструкция
Liquibase	Инструкция
Flyway	Инструкция
dbt	Инструкция

Движки выполнения запросов

Движок	Инструкция

Apache Spark™	Инструкция

Объектно-реляционное отображение (ORM)

Система	Инструкция
Hibernate	Инструкция
Spring Data JDBC	Инструкция
Kotlin Exposed	Инструкция
JOOQ	Инструкция
Dapper	Инструкция
Entity Framework	Инструкция
Linq To DB	Инструкция
SQLAlchemy	Инструкция
Django	Инструкция
Drizzle ORM	Инструкция

Spring

Компонент	Инструкция
spring-ydb-retry	Автоматический повтор транзакций

Векторный поиск

Инструмент	Инструкция
LangChain	Инструкция
Mem0	Инструкция

Конвертация SQL-диалектов

Инструмент	Инструкция
Плагин ydb-sqlglot-plugin для SQLGlot	Инструкция
Конвертер SQL-диалектов в YQL	Инструкция

Смотрите также

- Справка по YDB SDK
- Kafka API

Графические пользовательские интерфейсы

Среда	Инструкция	Уровень поддержки
YDB UI	Справка	
DBeaver	Инструкция	С помощью JDBC-драйвера
JetBrains Database viewer	—	С помощью JDBC-драйвера
JetBrains DataGrip	Инструкция	С помощью JDBC-драйвера
Другие JDBC-совместимые IDE	—	С помощью JDBC-драйвера
Visual Studio Code	Инструкция	С помощью YDB for VS Code
Jupyter Notebook	Инструкция	С помощью YDB-SQLAlchemy

Подключение к YDB с помощью DBeaver

DBeaver — бесплатный кроссплатформенный инструмент управления базами данных с открытым исходным кодом, обеспечивающий визуальный интерфейс для подключения к различным базам данных и выполнения SQL-запросов. Он поддерживает множество систем управления базами данных, включая MySQL, PostgreSQL, Oracle и SQLite.

DBeaver позволяет работать с YDB по протоколу Java DataBase Connectivity (JDBC). Данная статья демонстрирует, как настроить такую интеграцию.

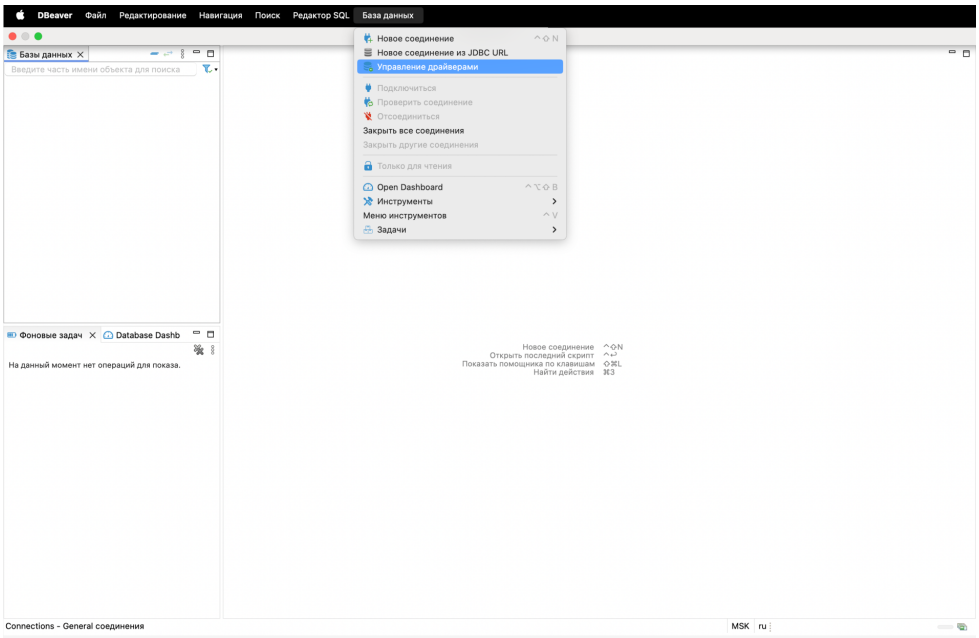
Подключение JDBC-драйвера YDB к DBeaver

Для подключения к YDB из DBeaver понадобится JDBC-драйвер. Для загрузки JDBC-драйвера выполните следующие шаги:

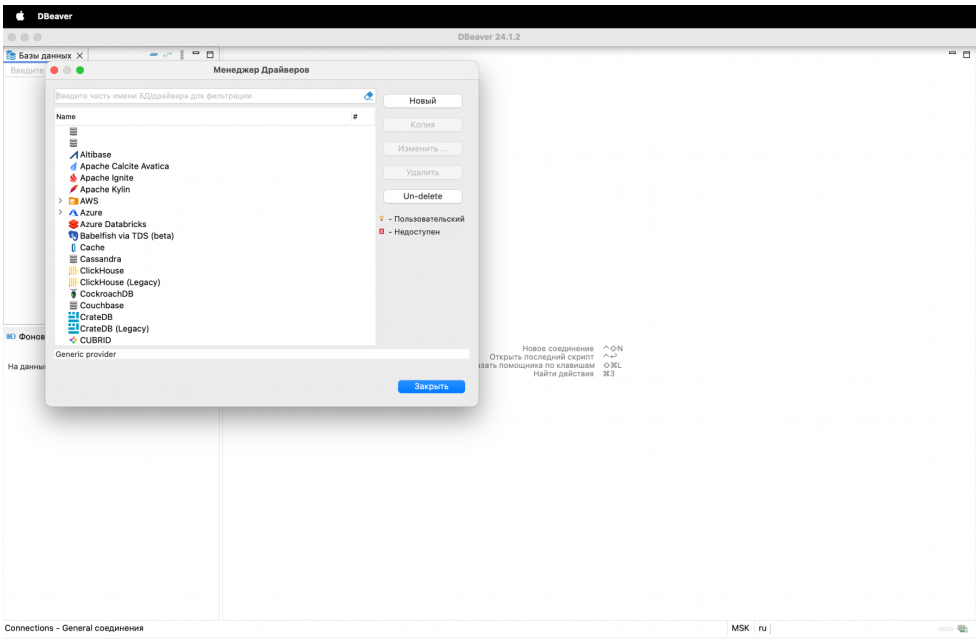
1. Перейдите в репозиторий ydb-jdbc-driver.
2. Выберите последний релиз (отмечен тегом `Latest`) и сохраните файл `ydb-jdbc-driver-shaded-<driver-version>.jar`.

Для подключения загруженного JDBC-драйвера выполните следующие шаги:

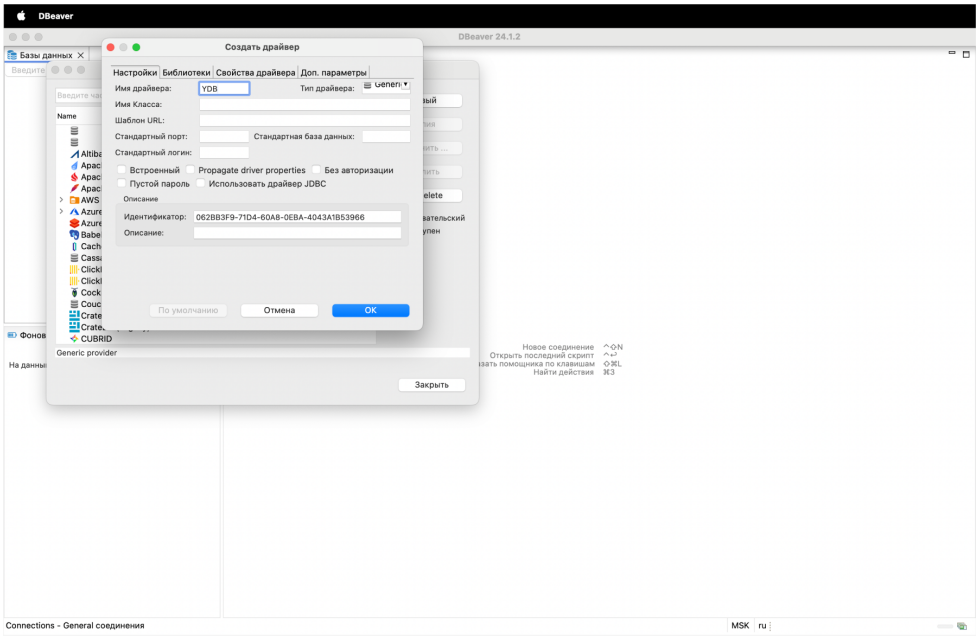
1. Выберите в верхнем меню DBeaver пункт **База данных**, а затем подпункт **Управление драйверами**:



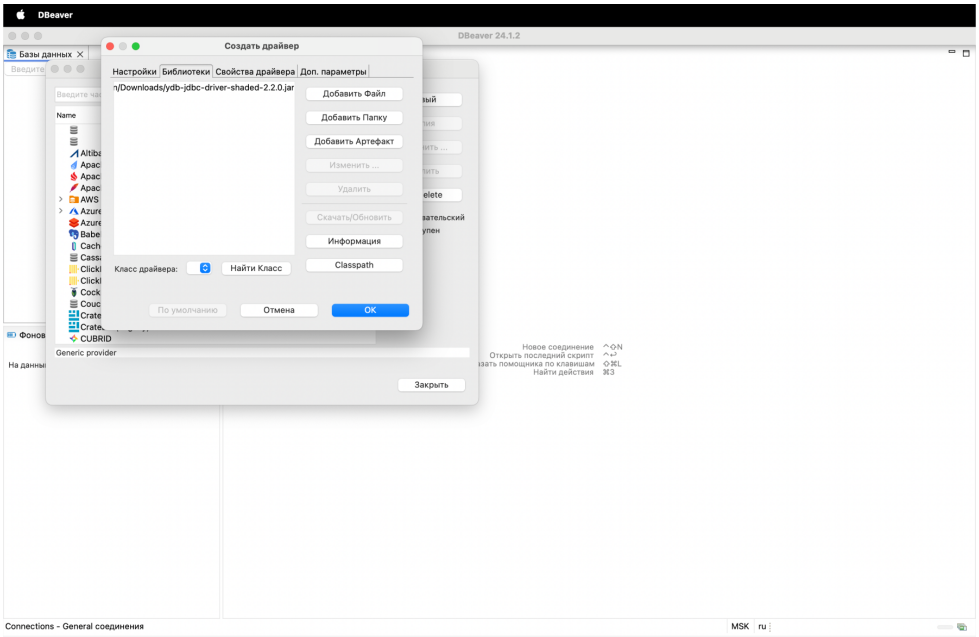
2. Чтобы создать новый драйвер, в открывшемся окне **Менеджер Драйверов** нажмите кнопку **Новый**:



3. В открывшемся окне **Создать драйвер**, в поле **Имя драйвера**, укажите `YDB`:

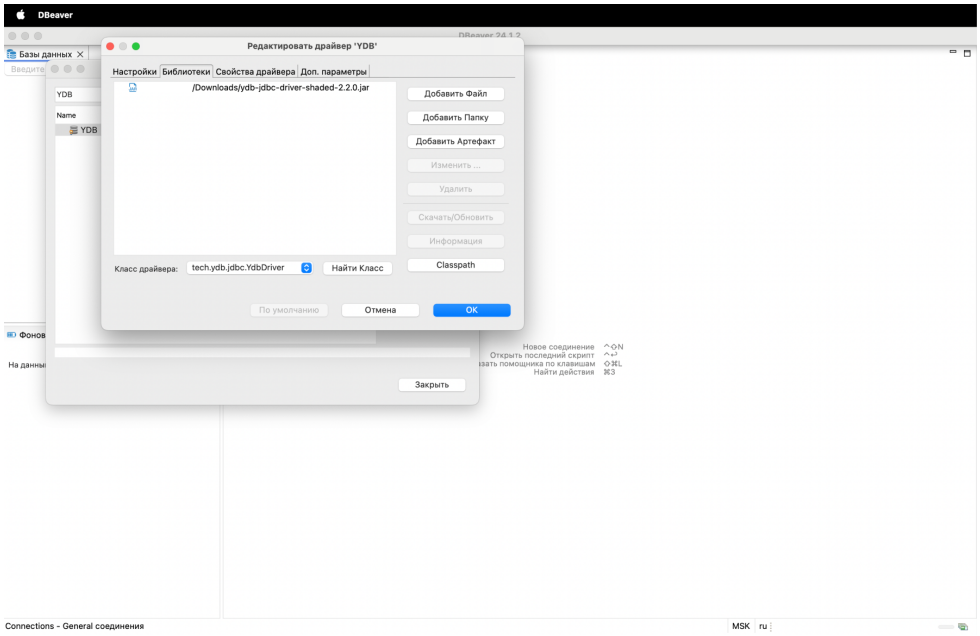


4. Перейдите в раздел **Библиотеки**, нажмите кнопку **Добавить файл**, укажите путь к скачанному ранее JDBC-драйверу YDB (файлу `ydb-jdbc-driver-shaded-<driver-version>.jar`) и нажмите кнопку **OK**:



5. В списке драйверов появится пункт **YDB**. Дважды кликните по новому драйверу и перейдите на вкладку **Библиотеки**, нажмите кнопку **Найти Класс** и в выпадающем списке выберите `tech.ydb.jdbc.YdbDriver`.

> **Важно**
>
> Обязательно явно выберите пункт выпадающего списка `tech.ydb.jdbc.YdbDriver`, нажав на него. В противном случае DBeaver будет считать, что драйвер не был выбран.



Создание подключения к YDB

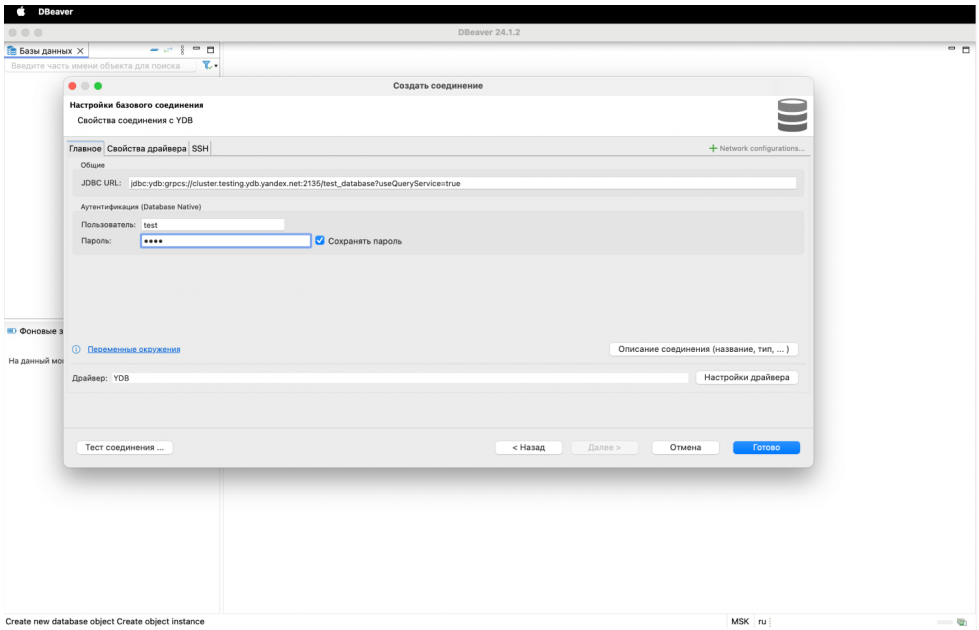
Для создания подключения необходимо выполнить следующие шаги:

1. В DBeaver создайте новое соединение, указав тип соединения `YDB`.
2. В открывшемся окне перейдите в раздел **Главное**.
3. В подразделе **Общие**, в поле ввода **JDBC URL**, укажите следующую строку соединения:

   ```
   jdbc:ydb:<ydb_endpoint>/<ydb_database>?useQueryService=true
   ```

   Где:

   - `ydb_endpoint` — эндпойнт кластера YDB, к которому будут выполняться подключение.
   - `ydb_database` — путь к базе данных в кластере YDB, к которой будут выполняться запросы.

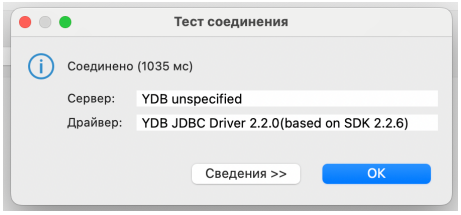


4. В поля **Пользователь** и **Пароль** введите логин и пароль для подключения к базе данных. Полный список способов аутентификации и строк подключения к YDB приведён в описании JDBC-драйвера.

> **Примечание**
>
> В Managed-инсталляциях YDB недоступна аутентификация по логину и паролю.

1. Нажмите кнопку **Тест соединения ...** для проверки настроек.

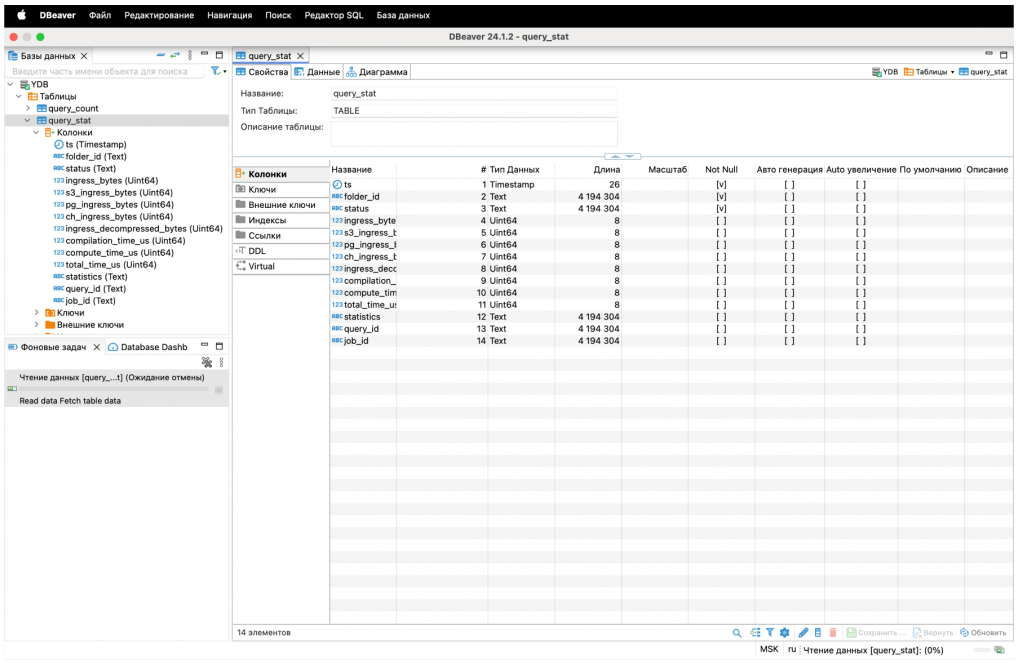
   Если все настройки указаны верно, то появится сообщение об успешном тестировании соединения:



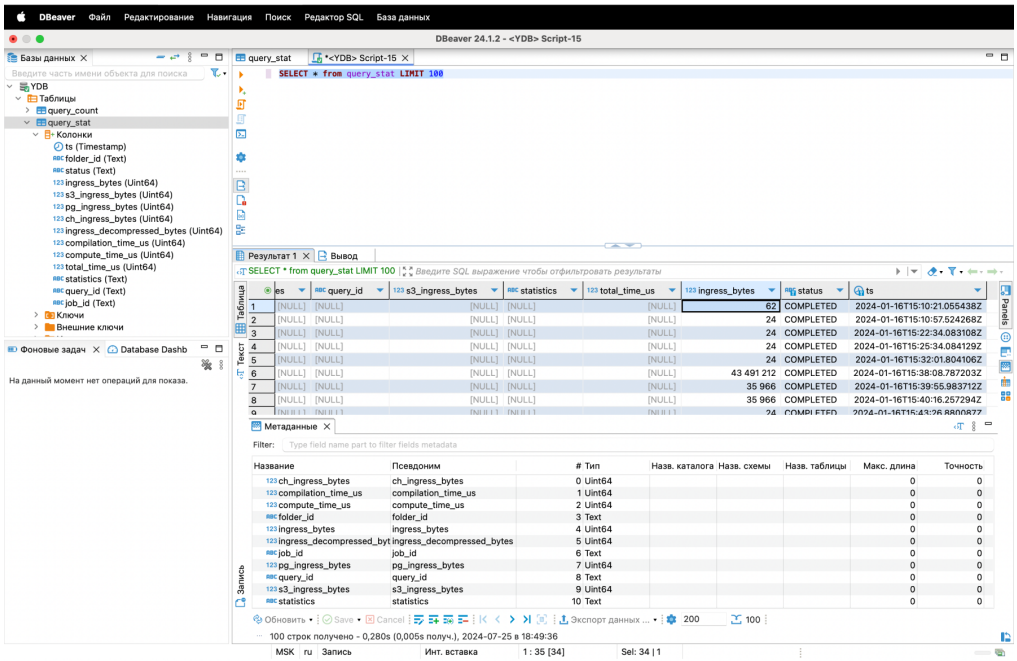
2. Нажмите кнопку **Готово** для сохранения соединения.

Работа с YDB

С помощью DBeaver можно просматривать список и структуру таблиц:



А также выполнять запросы к данным:



Подключение к YDB с помощью плагина DBeaver

DBeaver — бесплатный кроссплатформенный инструмент управления базами данных с открытым исходным кодом, обеспечивающий визуальный интерфейс для подключения к различным базам данных и выполнения SQL-запросов.

YDB DBeaver Plugin — расширение DBeaver с нативной поддержкой YDB. В отличие от подключения через JDBC-драйвер, плагин предоставляет специализированный интерфейс для работы с объектами YDB: иерархический навигатор таблиц, топиков, представлений, внешних источников данных, поддержку всех способов аутентификации, редактор YQL, визуализацию планов выполнения, мониторинг сессий и кластера, управление правами доступа (ACL) и другие возможности.

Ключевые возможности плагина

- Подключение к YDB со всеми способами аутентификации: анонимная, статическая, по токену, по сервисному аккаунту, по метаданным.
- Иерархический навигатор объектов: таблицы, топики, внешние источники данных, внешние таблицы, представления.
- Системные объекты: системные представления (`.sys`), пулы ресурсов и классификаторы пулов ресурсов.
- Редактор YQL с подсветкой 150+ ключевых слов и встроенных функций.
- Визуализация плана выполнения запроса (`EXPLAIN` / `EXPLAIN ANALYZE`).
- Мониторинг активных сессий через `.sys/query_sessions`.
- Дашборд кластера: загрузка CPU, использование дисков, памяти, сетевой трафик, статус узлов (обновление каждые 5 секунд).
- Управление правами доступа (ACL): выдача, отзыв, просмотр разрешений.
- Управление потоковыми запросами: просмотр, изменение, запуск, остановка.
- Федеративные запросы через внешние источники данных (S3, базы данных).
- Конвертер SQL-запросов из других диалектов (PostgreSQL, MySQL, ClickHouse и других) в YQL.
- Специализированные редакторы для типов данных `JSON`, `JSONDOCUMENT`, `YSON`.

Требования

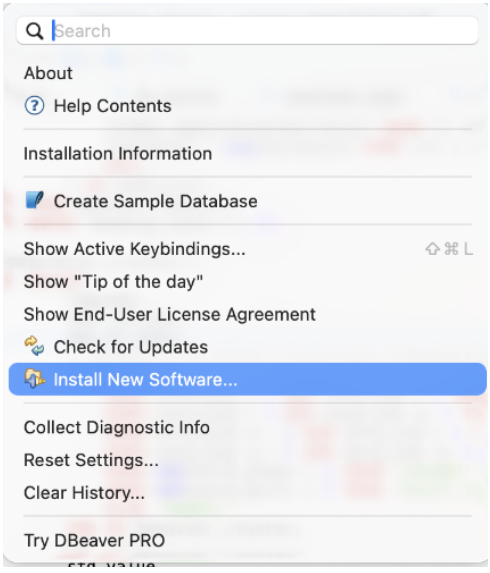
Для работы плагина требуется DBeaver Community Edition версии 24.x или новее.

Установка плагина

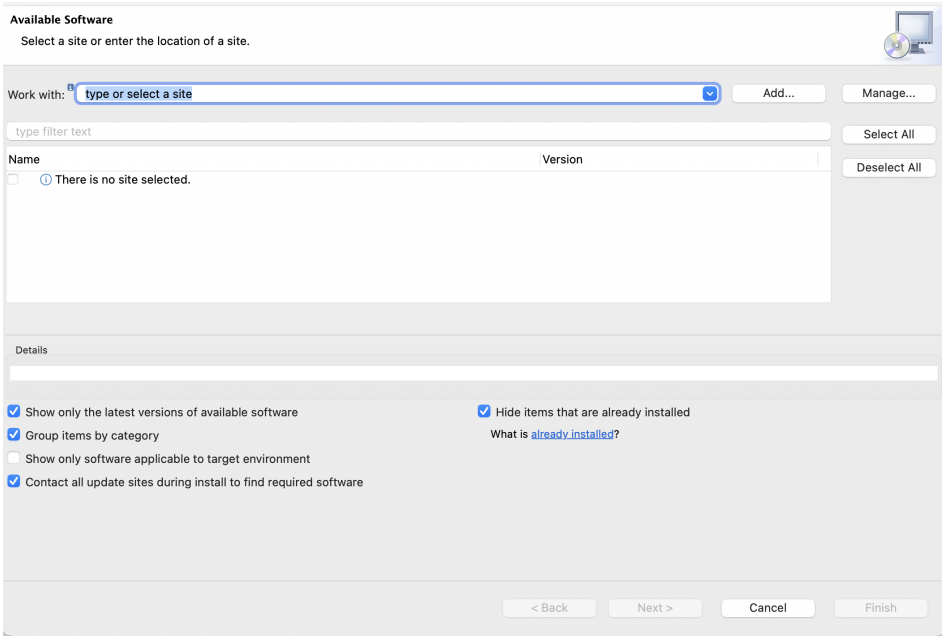
Плагин можно установить двумя способами:

- **Через URL P2-репозитория (рекомендуется)**: DBeaver загружает плагин из облачного хранилища и запоминает источник для последующих автоматических обновлений.
- **Из ZIP-архива (GitHub Releases)**: скачайте архив и установите через локальный файл. Подходит для сетей, где внешние Eclipse-репозитории недоступны, или для воспроизводимой установки конкретной версии. Автоматические обновления при этом способе не поддерживаются.

Установка через URL P2-репозитория

1. Откройте DBeaver. В верхнем меню выберите **Help** → **Install New Software...**.
   Скриншот
   

2. Нажмите кнопку **Add...** справа от поля **Work with:**.
   Скриншот

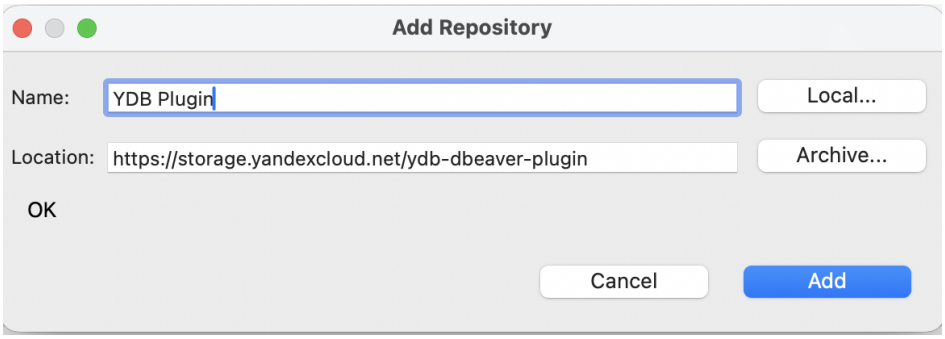


3. В открывшемся окне **Add Repository** укажите имя репозитория (например, `YDB Plugin`) и вставьте в поле **Location** следующий URL:

   ```
   https://storage.yandexcloud.net/ydb-dbeaver-plugin
   ```

   Нажмите **Add**. DBeaver загрузит метаданные репозитория.

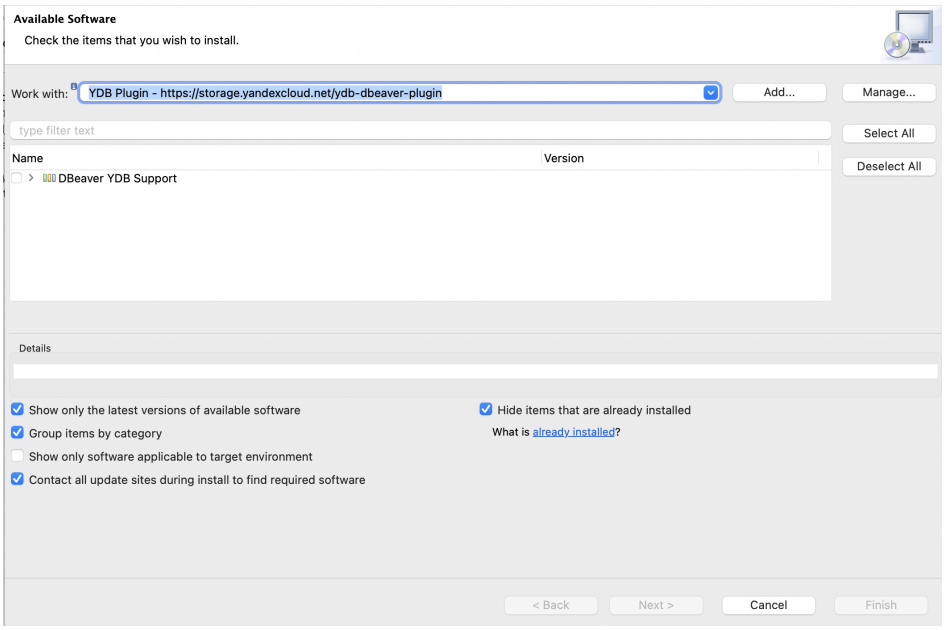
   Скриншот

   Важно

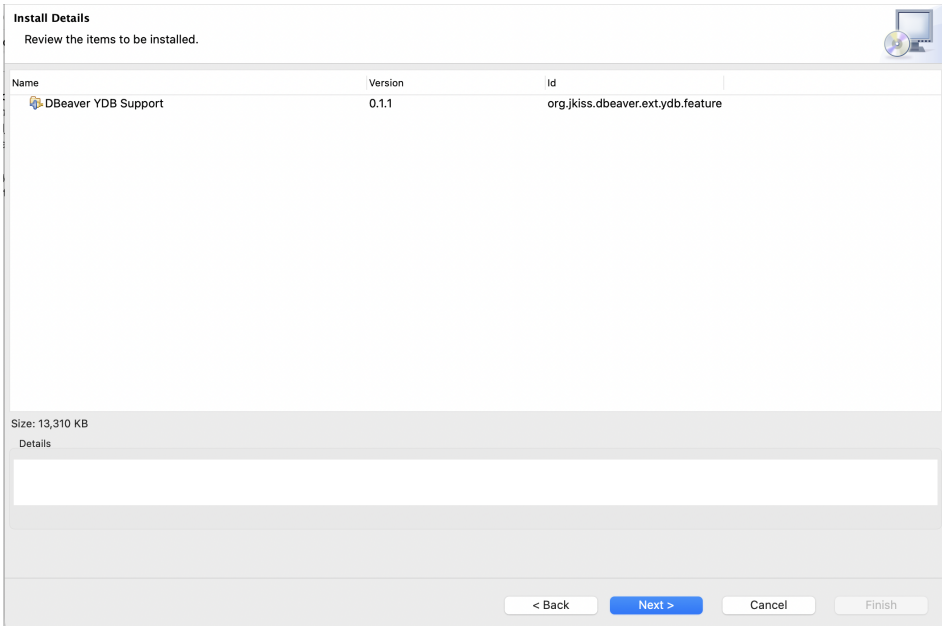
   В процессе установки DBeaver скачивает не только сам плагин, но и его зависимости (OSGi-компоненты) с серверов проекта Eclipse и других внешних репозиториев. Если эти ресурсы недоступны из вашей сети, обратитесь к сетевому администратору или воспользуйтесь установкой из ZIP-архива.

4. В списке компонентов появится категория **DBeaver YDB Support**. Отметьте её и нажмите **Next >**.
   Скриншот



5. На экране **Install Details** убедитесь, что в списке присутствуют оба компонента (`org.jkiss.dbeaver.ext.ydb` и `org.jkiss.dbeaver.ext.ydb.ui`), и нажмите **Next >**.

   Скриншот

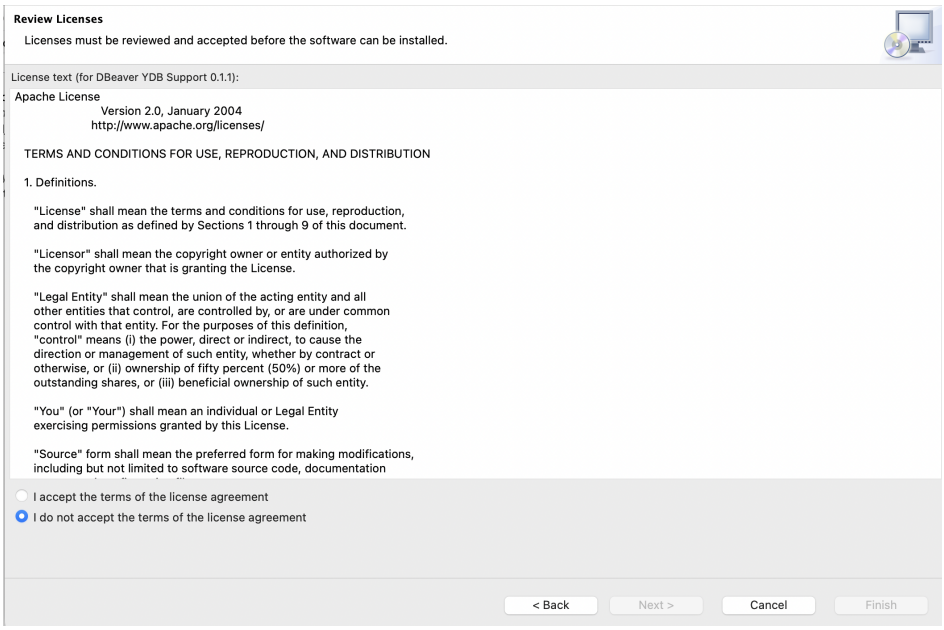


6. DBeaver может показать предупреждение о неподписанном содержимом. Это ожидаемое поведение — JAR-файлы плагина не подписаны коммерческим сертификатом. Нажмите **Install Anyway**.

> **Примечание**
>
> Eclipse, на котором основан DBeaver, проверяет подписи JAR-файлов для подтверждения подлинности. Этот плагин с открытым исходным кодом распространяется без подписи, исходный код доступен в репозитории.

7. Ознакомьтесь с лицензией (Apache License 2.0), выберите **I accept the terms of the license agreements** и нажмите **Finish**.

   Скриншот



8. DBeaver установит плагин и предложит перезапуск. Нажмите **Restart Now**. После перезапуска плагин станет активным.

Установка из ZIP-архива (GitHub Releases)

1. Перейдите на страницу GitHub Releases и скачайте файл `ydb-dbeaver-plugin-*.zip` нужной версии.
2. Откройте DBeaver. В верхнем меню выберите **Help** → **Install New Software...**.
3. Нажмите кнопку **Add...** справа от поля **Work with:**.
4. В открывшемся окне **Add Repository** нажмите **Archive...**, укажите путь к скачанному ZIP-файлу и нажмите **Add**.
5. В списке компонентов появится категория **DBeaver YDB Support**. Отметьте её.

> **Важно**
>
> Перед нажатием **Next >** снимите флаг **Contact all update sites during install to find required software** в нижней части окна **Install New Software**. Если этот флаг включён, DBeaver при расчёте зависимостей обращается ко всем зарегистрированным P2-репозиториям (`dbeaver.io`, `eclipse.org` и другим), и при их недоступности или медленных ответах диалог установки зависает на этапе **Calculating requirements and dependencies**. Плагин использует только компоненты DBeaver, уже установленные локально, поэтому обращение к другим репозиториям при установке не требуется.

6. Нажмите **Next >**.
7. На экране **Install Details** убедитесь, что в списке присутствуют оба компонента (`org.jkiss.dbeaver.ext.ydb` и `org.jkiss.dbeaver.ext.ydb.ui`), и нажмите **Next >**.
8. DBeaver может показать предупреждение о неподписанном содержимом. Нажмите **Install Anyway**.
9. Ознакомьтесь с лицензией (Apache License 2.0), выберите **I accept the terms of the license agreements** и нажмите **Finish**.
10. DBeaver установит плагин и предложит перезапуск. Нажмите **Restart Now**. После перезапуска плагин станет активным.

> **Важно**
>
> При установке из ZIP-архива автоматические обновления не поддерживаются: DBeaver не знает, где искать новые версии. Для обновления:
>
> 1. Скачайте новый архив со страницы GitHub Releases.
> 2. Откройте **Help** → **Installation Information**, выберите плагин и нажмите **Uninstall**.
> 3. Установите новую версию, следуя инструкции по установке из ZIP-архива.

Создание подключения к YDB

Для создания подключения к YDB выполните следующие шаги:

1. В верхнем меню выберите **Database** → **New Database Connection** (или нажмите `Ctrl+Shift+N`).
2. В поле поиска введите `YDB`. Выберите **YDB** из списка и нажмите **Next**.
3. Откроется страница настройки подключения к YDB. Заполните поля:

Поле	Описание	Пример
Host	Хост эндпойнта кластера YDB	`ydb.example.com`
Port	Порт (по умолчанию `2135`)	`2135`
Database	Путь к базе данных	`/Root/database`

Monitoring URL	URL YDB UI с путём к базе данных, используется для дашборда (необязательно)	`http://ydb.example.com:8765/monitoring/tenant?name=%2FRoot%2Fdatabase`
Use secure connection	Использовать защищённое соединение (`grpcs://`)	☑
Enable autocomplete API	Автодополнение через API YDB	☑

4. Выберите способ аутентификации в выпадающем списке **Auth type** (см. Способы аутентификации).
5. Нажмите кнопку **Test Connection** для проверки настроек. При успешном подключении появится диалог с временем соединения в миллисекундах.
6. Нажмите кнопку **Finish**. Подключение появится в панели **Database Navigator**.

Способы аутентификации

Плагин поддерживает все способы аутентификации, доступные в YDB. Способ выбирается в выпадающем списке **Auth type** на странице настройки подключения.

Anonymous

Подключение без учётных данных. Используется для локальных или тестовых установок YDB. Дополнительных полей заполнять не требуется.

Static (логин и пароль)

Аутентификация по логину и паролю. Укажите имя пользователя в поле **User** и пароль в поле **Password**. Используется, если на сервере YDB включена аутентификация по логину и паролю.

> **Примечание**
>
> В managed-инсталляциях YDB аутентификация по логину и паролю отключена: управляемые сервисы используют централизованную систему управления доступом облачной платформы (IAM).

Token

Аутентификация по IAM- или OAuth-токену. Введите токен в поле **Token**. Токен передаётся в заголовке каждого запроса.

Service Account

Аутентификация по ключу сервисного аккаунта Yandex Cloud. Укажите путь к JSON-файлу с ключом в поле **SA Key File** (для выбора файла используйте кнопку **...**). Подробнее о том, как создать авторизованный ключ, см. в документации Yandex Cloud.

Формат файла ключа:

```
{
  "id": "aje...",
  "service_account_id": "aje...",
  "private_key": "-----BEGIN RSA PRIVATE KEY-----\n..."
}
```

Metadata

Аутентификация через сервис метаданных Yandex Cloud. Плагин получает IAM-токен от сервиса метаданных виртуальной машины. Используется, только когда DBeaver запущен на виртуальной машине Yandex Cloud.

Навигатор объектов

После подключения в панели **Database Navigator** отображается иерархия объектов YDB. Корневой узел — подключение, внутри — путь к базе данных, который содержит следующие папки:

- **Tables** — таблицы, организованные по поддиректориям согласно пути в YDB (например, таблица по пути `folder1/subfolder/mytable` будет вложена в `folder1 → subfolder`).
- **Topics** — топики.
- **Views** — представления.
- **External Data Sources** — внешние источники данных.
- **External Tables** — внешние таблицы.
- **System Views (.sys)** — системные представления, такие как `partition_stats`, `query_sessions`.
- **Resource Pools** — пулы ресурсов.

Работа с плагином

Редактор YQL

Откройте **SQL Editor** (`F3` или двойной клик по подключению). Редактор поддерживает:

- Подсветку синтаксиса YQL: ключевые слова (`UPSERT`, `REPLACE`, `EVALUATE`, `PRAGMA`, `WINDOW` и 145+ других), типы данных, встроенные функции.
- Автодополнение имён таблиц, колонок и функций.
- Выполнение запросов: `Ctrl+Enter` — текущий запрос, `Ctrl+Shift+Enter` — весь скрипт.

Пример YQL-запроса:

```
UPSERT INTO `users` (id, name, created_at)
VALUES (1, "Alice", CurrentUtcDatetime());
```

EXPLAIN и план выполнения

Нажмите **Explain** (или `Ctrl+Shift+E`), чтобы получить план выполнения запроса. Плагин отображает:

- **Text plan** — дерево операций в текстовом виде.
- **Diagram** — графическое представление в виде DAG.
- **SVG plan** — интерактивная визуализация.

`EXPLAIN ANALYZE` дополнительно показывает статистику выполнения (количество строк, время выполнения).

Менеджер сессий

Выберите пункт меню **Tools** → **Sessions Manager**, либо кликните правой кнопкой по подключению и выберите соответствующий пункт. В открывшемся представлении отображаются все активные сессии с текущим запросом, состоянием и длительностью (данные из системного представления `.sys/query_sessions`). Галочка **Hide Idle** скрывает сессии без активного запроса.

Дашборд кластера

Откройте вкладку **Dashboard** в редакторе подключения (требует заполненного поля **Monitoring URL** при настройке подключения).

> **Важно**
>
> Дашборд доступен только при работе с self-hosted инсталляциями YDB, где есть доступ к YDB UI. В Yandex Cloud Managed Service for YDB UI не публикуется, поэтому данные дашборда недоступны — для мониторинга используйте средства облачной платформы.

Дашборд отображает в реальном времени (обновление каждые 5 секунд):

- Загрузку CPU по узлам.
- Использование дискового пространства.
- Использование памяти.
- Сетевой трафик.
- Количество выполняющихся запросов.
- Статус узлов кластера.

Стриминговые запросы

В навигаторе раскройте папку **Streaming Queries**. Для каждого запроса доступны:

- Просмотр исходного YQL.
- Просмотр ошибок (issues).
- Просмотр плана выполнения.
- Действия: **Start**, **Stop**, **Alter**.

Конвертер SQL-диалектов

Плагин позволяет преобразовать SQL-запрос, написанный на другом диалекте (PostgreSQL, MySQL, ClickHouse и других), в YQL. Конвертер доступен в контекстном меню **Tools** → **Convert Dialect** в редакторе подключения.

Чтобы преобразовать запрос:

1. В выпадающем списке **Source Dialect** выберите исходный диалект SQL. Список диалектов запрашивается с внешнего сервиса плагина при первом открытии вкладки.
2. Вставьте исходный SQL-код в поле **Input SQL**.
3. Нажмите **Convert**. Результат появится в нижнем поле.
4. Нажмите **Copy**, чтобы скопировать результат в буфер обмена.

Подробнее о принципах работы конвертера, поддерживаемых диалектах и ограничениях см. в статье Конвертер SQL-диалектов в YQL.

> **Важно**
>
> Для преобразования плагин отправляет исходный запрос на внешний HTTPS-сервис. Конвертер не работает без доступа в интернет. Не используйте конвертер для запросов, содержащих конфиденциальные данные.

Создание объектов

Кликните правой кнопкой по папке или объекту и выберите **Create New**:

- **Create Table** — создать новую таблицу.
- **Create Topic** — создать новый топик.

Обновление плагина

Автоматические обновления поддерживаются только при установке через URL P2-репозитория. DBeaver запоминает источник (URL репозитория) и при публикации новой версии предлагает обновление одним из двух способов:

1. Автоматически при следующем запуске DBeaver (если включена проверка обновлений в **Window** → **Preferences** → **Install/Update** → **Automatic Updates**).
2. Вручную через **Help** → **Check for Updates**: выберите доступное обновление и пройдите те же шаги, что и при первой установке (лицензия → предупреждение о неподписанном содержимом → перезапуск).

При установке из ZIP-архива обновление выполняется вручную: скачайте новый архив со страницы GitHub Releases, откройте **Help → Installation Information**, выберите плагин, нажмите **Uninstall**, затем установите новую версию заново.

Подключение к YDB с помощью плагина VS Code

Visual Studio Code — бесплатный кроссплатформенный редактор кода с открытым исходным кодом, поддерживающий широкую экосистему расширений для работы с базами данных, языковых серверов и инструментов разработки.

YDB for VS Code — расширение VS Code с нативной поддержкой YDB. Плагин предоставляет специализированный интерфейс для работы с объектами YDB: иерархический навигатор таблиц, топиков, представлений, внешних источников данных, поддержку всех способов аутентификации, редактор YQL с подсветкой синтаксиса, визуализацию планов выполнения, мониторинг сессий и кластера, управление правами доступа (ACL), встроенный MCP-сервер для AI-ассистентов и другие возможности.

Ключевые возможности плагина

- Подключение к YDB со всеми способами аутентификации: анонимная, статическая, по токену, по сервисному аккаунту, по метаданным.
- Иерархический навигатор объектов: таблицы (строковые и колоночные), топики, представления, внешние источники данных, внешние таблицы, трансферы, потоковые запросы.
- Системные объекты: системные представления (`.sys`), пулы ресурсов.
- Редактор YQL с подсветкой синтаксиса, автодополнением таблиц и колонок.
- Выполнение запросов и визуализация результатов: таблица, JSON, диаграмма.
- Визуализация плана выполнения запроса (`EXPLAIN`).
- Мониторинг активных сессий через `.sys/query_sessions`.
- Дашборд кластера на базе YDB UI: загрузка CPU, использование памяти, сетевой трафик (обновление каждые 10 секунд).
- Управление правами доступа (ACL): просмотр разрешений на объекты базы данных.
- Генерация DDL-скриптов (`CREATE`) для любого объекта базы данных.
- Управление потоковыми запросами: просмотр, запуск, остановка.
- Конвертер SQL-запросов из других диалектов (PostgreSQL, MySQL, ClickHouse и других) в YQL.
- Встроенный MCP-сервер — прямой доступ к базам данных из AI-ассистентов (Claude Code и других).
- Семантический поиск по документации YQL (RAG) для AI-assisted написания запросов.

Требования

Для работы плагина требуется Visual Studio Code версии 1.75.0 или новее.

Установка плагина

Плагин можно установить из VS Code Marketplace или из `.vsix`-файла на странице GitHub Releases.

Установка из VS Code Marketplace

1. Откройте панель расширений в VS Code (`Ctrl+Shift+X` / `Cmd+Shift+X`).
2. В строке поиска введите `YDB for VS Code` и выберите расширение от издателя `ydb-tech` (прямая ссылка).
3. Нажмите **Install**.

Альтернативно, установить плагин из Marketplace можно одной командой в терминале:

```
code --install-extension ydb-tech.ydb-vscode-plugin
```

Установка из VSIX-файла

Способ подходит, если нужна конкретная версия или нет доступа к Marketplace.

1. Перейдите на страницу GitHub Releases и скачайте файл `ydb-vscode-plugin-*.vsix` нужной версии.
2. Установите расширение одним из способов:
   - **Через терминал:**

     ```
     code --install-extension ydb-vscode-plugin-X.X.X.vsix
     ```

     Где `X.X.X` — номер скачанной версии.
   - **Через интерфейс VS Code:**
     1. Откройте панель расширений (`Ctrl+Shift+X` / `Cmd+Shift+X`).
     2. Нажмите `...` (три точки) в правом верхнем углу панели.
     3. Выберите **Install from VSIX...**.
     4. Укажите путь к скачанному `.vsix`-файлу.

После установки любым из способов перезагрузите VS Code. В панели Activity Bar появится значок **YDB**.

Создание подключения к YDB

1. Нажмите на значок **YDB** в Activity Bar слева.
2. В панели **Connections** нажмите кнопку **Add Connection** (значок `+`).
3. Откроется форма создания подключения. Заполните поля:

Поле	Описание	Пример
Connection Name	Произвольное имя подключения	`my-ydb`
Host	Хост эндпойнта кластера YDB	`ydb.example.com`
Port	Порт (по умолчанию `2135`)	`2135`
Database	Путь к базе данных	`/Root/database`

Monitoring URL	URL YDB UI, используется для дашборда (заполняется автоматически по хосту, можно переопределить)	`http://ydb.example.com:8765`
Secure connection (grpcs)	Использовать защищённое соединение (`grpcs://`)	☑
Use RAG	Включить поиск по документации YQL для данного подключения	☑

4. При необходимости укажите путь к пользовательскому CA-сертификату (PEM) в поле **CA Certificate File** — для подключений с нестандартным TLS. Если поле не заполнено, используется встроенный сертификат Yandex Cloud.
5. Выберите способ аутентификации в выпадающем списке **Auth type** (см. Способы аутентификации).
6. Нажмите **Test Connection** для проверки настроек. При успешном подключении появится сообщение об успехе.
7. Нажмите **Save**. Подключение появится в панели **Connections**.

Способы аутентификации

Плагин поддерживает все способы аутентификации, доступные в YDB. Способ выбирается в выпадающем списке **Auth type** на форме создания подключения.

Anonymous

Подключение без учётных данных. Используется для локальных или тестовых установок YDB. Дополнительных полей заполнять не требуется.

Static Credentials (логин и пароль)

Аутентификация по логину и паролю. Укажите имя пользователя в поле **Username** и пароль в поле **Password**. Используется, если на сервере YDB включена аутентификация по логину и паролю.

> **Примечание**
>
> В managed-инсталляциях YDB аутентификация по логину и паролю отключена: управляемые сервисы используют централизованную систему управления доступом облачной платформы (IAM).

Access Token

Аутентификация по IAM- или OAuth-токену. Введите токен в поле **Token**. Токен передаётся в заголовке каждого запроса.

> **Важно**
>
> IAM-токен имеет ограниченный срок жизни — не более 12 часов, после чего его необходимо получить заново и обновить в настройках подключения. Для долгоживущих подключений используйте аутентификацию по сервисному аккаунту или сервису метаданных.

Service Account Key File

Аутентификация по ключу сервисного аккаунта Yandex Cloud. Укажите путь к JSON-файлу с ключом в поле **Service Account Key File** (для выбора файла используйте кнопку **Browse**). Подробнее о том, как создать авторизованный ключ, см. в документации Yandex Cloud.

Формат файла ключа:

```
{
  "id": "aje...",
  "service_account_id": "aje...",
  "private_key": "-----BEGIN RSA PRIVATE KEY-----\n..."
}
```

Metadata Service

Аутентификация через сервис метаданных Yandex Cloud. Плагин получает IAM-токен от сервиса метаданных виртуальной машины. Используется, только когда VS Code запущен на виртуальной машине Yandex Cloud.

Навигатор объектов

После подключения кликните на подключение в панели **Connections** — откроется панель **Navigator** с иерархией объектов YDB. Навигатор содержит следующие разделы:

- **Tables** — таблицы (строковые и колоночные), организованные по поддиректориям согласно пути в YDB.
- **System Views** — системные представления (`.sys`), такие как `query_sessions` , `partition_stats` .
- **Views** — представления.
- **Topics** — топики.
- **External Data Sources** — внешние источники данных.
- **External Tables** — внешние таблицы.
- **Resource Pools** — пулы ресурсов.
- **Transfers** — трансферы данных.
- **Streaming Queries** — потоковые запросы.

Правый клик на любом объекте в навигаторе открывает контекстное меню с доступными действиями.

Работа с плагином

Рабочее пространство запросов

Откройте рабочее пространство запросов через `Ctrl+Shift+Q` (`Cmd+Shift+Q` на macOS) или нажмите **Open Query Workspace** в контекстном меню подключения. В рабочем пространстве можно писать и выполнять YQL-запросы, просматривать историю и результаты.

Для быстрого открытия редактора с предзаполненным запросом кликните правой кнопкой по таблице или представлению в навигаторе и выберите:

- **Show Preview** — `SELECT` первых 100 строк.
- **Make Query** — `SELECT` с именем объекта.

Редактор YQL

Редактор поддерживает:

- Подсветку синтаксиса YQL: ключевые слова, типы данных, встроенные функции.
- Автодополнение имён таблиц и колонок (по схеме подключённой базы данных).
- Выполнение запроса: `Ctrl+Enter` (`Cmd+Enter` на macOS).

Пример YQL-запроса:

```
UPSERT INTO `users` (id, name, created_at)
VALUES (1, "Alice", CurrentUtcDatetime());
```

Результаты выполнения отображаются в панели **Results** в виде таблицы, JSON или диаграммы (переключение вкладками).

EXPLAIN и план выполнения

Выберите **Explain YQL Query** в контекстном меню редактора или в командной палитре (`Ctrl+Shift+P`), чтобы получить план выполнения запроса. Плагин отображает дерево операций плана в текстовом виде.

Менеджер сессий

В панели **Sessions** (Activity Bar → YDB) отображаются все активные сессии с текущим запросом, состоянием и длительностью (данные из системного представления `.sys/query_sessions`). Кнопка **Toggle Hide Idle** скрывает сессии без активного запроса.

Дашборд кластера

В панели **Database Load** (Activity Bar → YDB) отображается нагрузка на кластер в реальном времени (обновление каждые 10 секунд):

- Загрузка CPU (% и количество ядер).
- Использование памяти (% и объём).
- Сетевой трафик.

> **Важно**
>
> Дашборд доступен только при работе с self-hosted инсталляциями YDB, где есть доступ к YDB UI. В Yandex Cloud Managed Service for YDB UI не публикуется, поэтому данные дашборда недоступны — для мониторинга используйте средства облачной платформы.

Потоковые запросы

В навигаторе раскройте раздел **Streaming Queries**. Для каждого запроса доступны:

- Просмотр исходного YQL (**View Source**).
- Запуск и остановка (**Start** / **Stop**).

Конвертер SQL-диалектов

Плагин позволяет преобразовать SQL-запрос, написанный на другом диалекте (PostgreSQL, MySQL, ClickHouse и других), в YQL. Конвертер доступен в панели **Convert Dialect** на Activity Bar.

Чтобы преобразовать запрос:

1. В выпадающем списке **Source Dialect** выберите исходный диалект SQL.
2. Вставьте исходный SQL-код в поле **Input SQL**.
3. Нажмите **Convert**. Результат появится в нижнем поле.
4. Скопируйте результат для использования в редакторе YQL.

Подробнее о принципах работы конвертера, поддерживаемых диалектах и ограничениях см. в статье Конвертер SQL-диалектов в YQL.

> **Важно**
>
> Для преобразования плагин отправляет исходный запрос на внешний HTTPS-сервис. Конвертер не работает без доступа в интернет. Не используйте конвертер для запросов, содержащих конфиденциальные данные.

Просмотр разрешений

Кликните правой кнопкой по объекту в навигаторе и выберите **View Permissions** для просмотра прав доступа (ACL), назначенных на этот объект.

Генерация DDL

Кликните правой кнопкой по таблице, топику или другому объекту в навигаторе и выберите **Create DDL** для получения `CREATE`-скрипта объекта.

Создание объектов

Кликните правой кнопкой по соответствующей папке в навигаторе, чтобы создать новый объект:

- **New Row Table** / **New Column Table** — создать строковую или колоночную таблицу.

- **New Topic** — создать топик.
- **New View** — создать представление.
- **New Object Storage Data Source** / **New YDB Data Source** — создать внешний источник данных.
- **New CSV/JSON/Parquet External Table** — создать внешнюю таблицу.
- **New Transfer** — создать трансфер.
- **New Streaming Query** — создать потоковый запрос.

Интеграция с AI-ассистентами (MCP)

Плагин запускает встроенный MCP-сервер (Model Context Protocol), который позволяет AI-ассистентам (Claude Code и другим) напрямую выполнять запросы к базам данных YDB, настроенным в плагине.

Настройка порта

Сервер запускается на порту **3333** (только localhost) по умолчанию. Изменить порт можно в настройках VS Code:

```
{
  "ydb.mcpPort": 3333
}
```

Если порт уже занят, расширение покажет предупреждение и продолжит работу без MCP.

Подключение Claude Code

1. Убедитесь, что расширение YDB запущено в VS Code и в панели **Connections** добавлено хотя бы одно подключение (см. Создание подключения).
2. Зарегистрируйте MCP-сервер в Claude Code глобально для текущего пользователя:

   ```
   claude mcp add --scope user --transport sse ydb http://localhost:3333/sse
   ```

   Флаг `--scope user` сохраняет конфигурацию глобально — сервер будет доступен из любого каталога, в котором запускается Claude Code. Без этого флага используется scope `local` (по умолчанию), и регистрацию придётся повторять для каждого проекта.
3. Проверьте подключение:

   ```
   claude mcp list
   ```

Доступные MCP-инструменты

Инструмент	Параметры	Описание
`ydb_list_connections`	—	Список всех подключений, настроенных в плагине
`ydb_query`	`connection`, `sql`	Выполнить YQL-запрос
`ydb_describe_table`	`connection`, `path`	Получить схему таблицы (колонки, первичный ключ)
`ydb_list_directory`	`connection`, `path?`	Список объектов в директории базы данных
`ydb_list_all`	`connection`, `path?`, `limit?`, `offset?`	Рекурсивный список всех объектов
`ydb_yql_help`	`query`, `connection?`	Поиск по документации YQL (требует включённого RAG)

Параметр `connection` — имя подключения, как оно отображается в панели **Connections**.

Поиск по документации YQL (RAG)

Плагин поддерживает семантический и ключевой поиск по документации YQL с помощью встроенного RAG-индекса (Retrieval-Augmented Generation). Это позволяет AI-ассистентам (через инструмент `ydb_yql_help`) находить релевантные фрагменты синтаксического справочника при написании запросов.

Включение RAG

1. При создании или редактировании подключения установите флаг **Use RAG**.
2. Плагин автоматически определит версию YDB и загрузит соответствующий индекс из облака при первом подключении.

Режимы поиска

- **Ключевой поиск (keyword)** — работает без дополнительных зависимостей, используется по умолчанию.
- **Семантический поиск (vector)** — требует локально запущенного Ollama с моделью эмбеддингов `nomic-embed-text`. См. Установка Ollama и модели эмбеддингов.

Статус RAG (Running / Not running) и статус Ollama отображаются прямо в форме подключения.

Установка Ollama и модели эмбеддингов

Ollama — локальный сервер для запуска языковых моделей и моделей эмбеддингов. Плагин обращается к нему по HTTP, чтобы преобразовывать тексты документации и поисковые запросы в векторы для семантического поиска.

1. Установите Ollama по инструкции на официальной странице загрузки (поддерживаются macOS, Windows и Linux).
2. Убедитесь, что сервис Ollama запущен и доступен на `http://localhost:11434`:

   ```
   curl http://localhost:11434/api/tags
   ```

   Ответ в формате JSON со списком моделей означает, что сервис работает.
3. Скачайте модель эмбеддингов `nomic-embed-text` (~270 МБ):

```
ollama pull nomic-embed-text
```

4. Убедитесь, что модель доступна:

```
ollama list
```

   В выводе должна появиться строка `nomic-embed-text`.

5. Откройте форму подключения в плагине и убедитесь, что статус Ollama отображается как **Running**. Если статус **Not running** — проверьте, что сервис Ollama запущен и доступен на `http://localhost:11434`.

> **Примечание**
>
> URL Ollama (`http://localhost:11434`) и имя модели (`nomic-embed-text`) уже заданы в плагине по умолчанию — переопределять настройки `ydb.ragOllamaUrl` и `ydb.ragOllamaModel` нужно только при нестандартной установке.
>
> Использовать другую модель эмбеддингов нельзя: индексы, публикуемые плагином в облаке, построены именно на `nomic-embed-text`, и при смене модели векторы запроса и документации окажутся несовместимы.

Обновление плагина

Способ обновления зависит от того, как был установлен плагин.

Обновление из VS Code Marketplace

Если плагин установлен из Marketplace, VS Code обновляет его автоматически.

Чтобы обновить плагин вручную:

1. Откройте панель расширений (`Ctrl+Shift+X` / `Cmd+Shift+X`).
2. В строке поиска введите `YDB for VS Code` и откройте установленное расширение.
3. Если доступна новая версия, появится кнопка **Update** — нажмите её.

Обновление из VSIX-файла

Если плагин установлен из VSIX-файла, автоматическое обновление недоступно — VS Code не отслеживает обновления для расширений, установленных вручную.

1. Скачайте новый `.vsix`-файл со страницы GitHub Releases.
2. Установите его поверх старой версии тем же способом, что и при первой установке — VS Code автоматически заменит предыдущую версию.

Подключение к YDB с помощью DataGrip

DataGrip — это эффективный кросс-платформенный инструмент для работы с реляционными базами данных и NoSQL.

DataGrip позволяет работать с YDB по протоколу Java Database Connectivity (JDBC). Данная статья демонстрирует, как настроить такую интеграцию.

Подключение JDBC-драйвера YDB к DataGrip

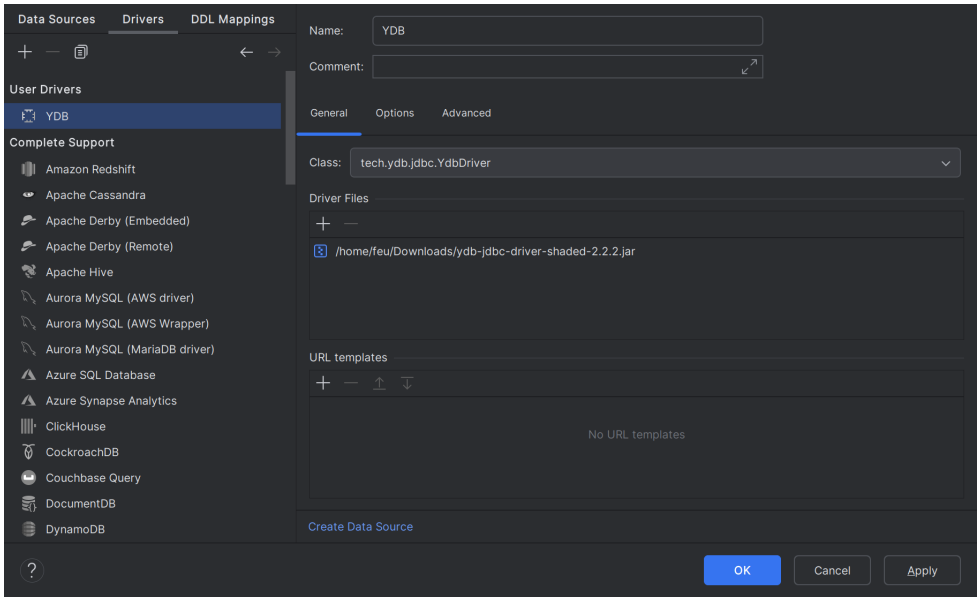
Для подключения к YDB из DataGrip понадобится JDBC-драйвер. Для загрузки JDBC-драйвера выполните следующие шаги:

1. Перейдите в репозиторий ydb-jdbc-driver.
2. Выберите последний релиз (отмечен тегом `Latest`) и сохраните файл `ydb-jdbc-driver-shaded-<driver-version>.jar`.

Для подключения загруженного JDBC-драйвера выполните следующие шаги:

1. Выберите в верхнем меню DataGrip пункт **File**, а затем подпункт **Data Sources...**.

   Откроется диалоговое окно **Data Sources and Drivers**.
2. Чтобы создать новый драйвер, в открывшемся окне **Data Sources and Drivers** перейдите на вкладку **Drivers** и нажмите кнопку **+**.
3. В поле **Name**, укажите `YDB`.
4. В разделе **Driver Files**, нажмите кнопку **+**, в выпадающем списке выберите **Custom JARs...**, укажите путь к скачанному ранее JDBC-драйверу YDB (файлу `ydb-jdbc-driver-shaded-<driver-version>.jar`) и нажмите кнопку **OK**.
5. В выпадающем списке **Class** выберите `tech.ydb.jdbc.YdbDriver`.



6. Нажмите кнопку **OK**.

Создание подключения к YDB

Для создания подключения необходимо выполнить следующие шаги:

1. Выберите в верхнем меню DataGrip пункт **File**, а затем подпункт **Data Sources...**.

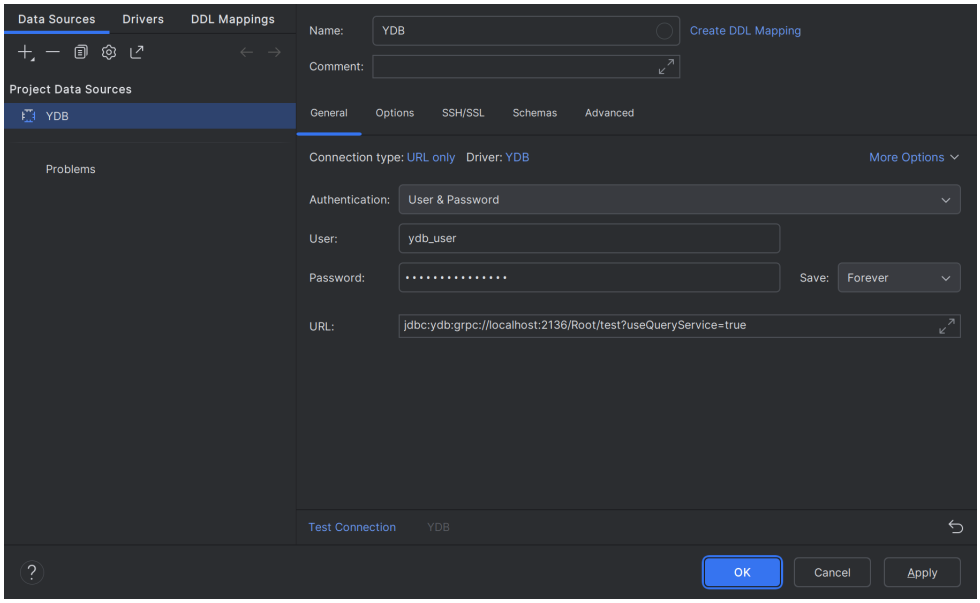
   Откроется диалоговое окно **Data Sources and Drivers**.
2. Чтобы создать новое соединение, в открывшемся окне **Data Sources and Drivers** на вкладке **Data Sources** нажмите кнопку **+** и укажите тип соединения `YDB`.
3. В выпадающем списке **Authentication** укажите тип аутентификации.
4. Если вы выбрали `User & Password` в качестве метода аутентификации, в поля **User** и **Password** введите логин и пароль для подключения к базе данных YDB.
5. В поле **URL**, укажите следующую строку соединения:

   ```
   jdbc:ydb:<ydb_endpoint>/<ydb_database>?useQueryService=true
   ```

   Где:

   - `ydb_endpoint` — эндпойнт кластера YDB, к которому будут выполняться подключение.
   - `ydb_database` — путь к базе данных в кластере YDB, к которой будут выполняться запросы.
6. Нажмите кнопку **Test Connection** для проверки настроек.

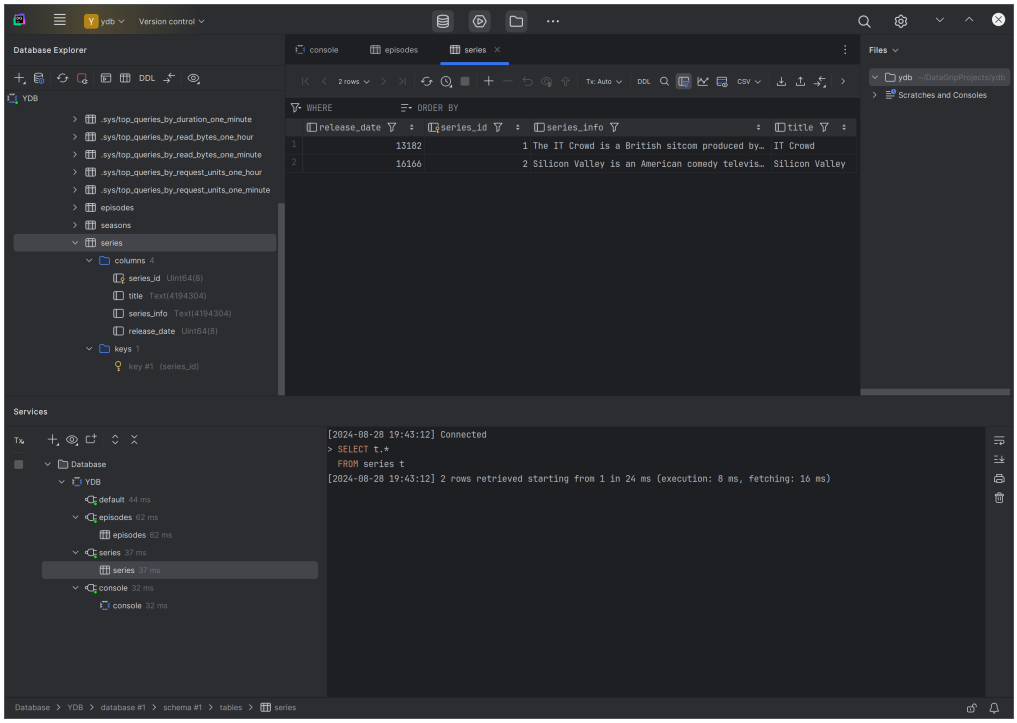
   Если все настройки указаны верно, то появится сообщение об успешном тестировании соединения.



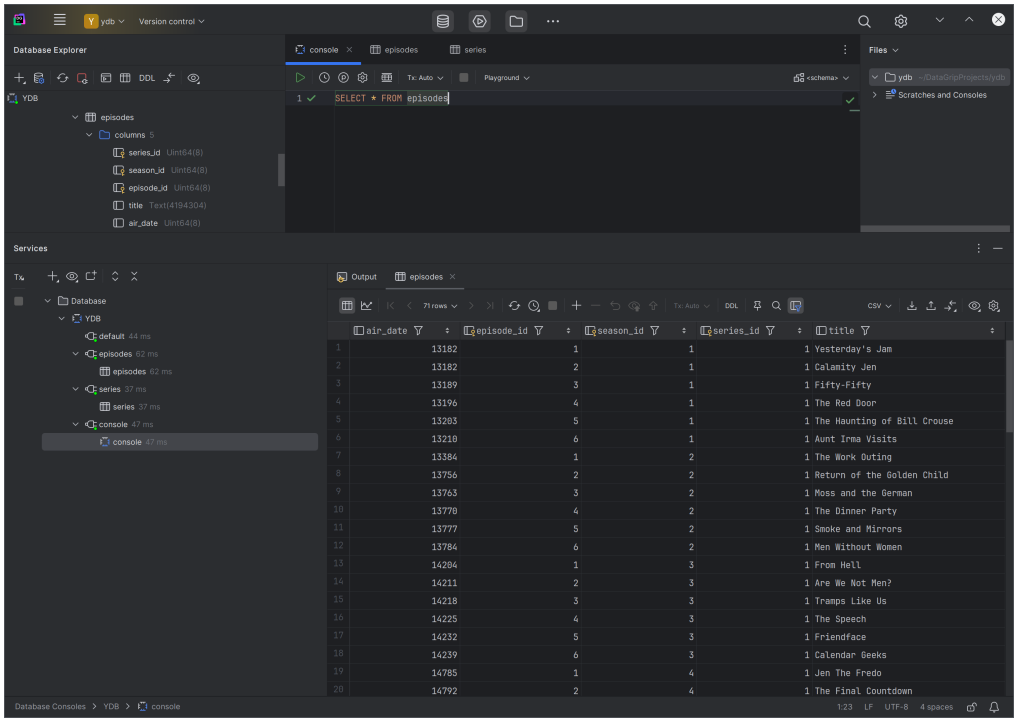
7. Нажмите кнопку **OK** для сохранения соединения.

Работа с YDB

С помощью DataGrip можно просматривать список и структуру таблиц:



А также выполнять запросы к данным:


Database Explorer
console
episodes
series
SELECT * FROM episodes
Services
Output
episodes
air_date
episode_id
season_id
series_id
title
1 13182 1 1 1 Yesterday's Jam
2 13182 2 1 1 Calamity Jen
3 13189 3 1 1 Fifty-Fifty
4 13196 4 1 1 The Red Door
5 13203 5 1 1 The Haunting of Bill Crouse
6 13210 6 1 1 Aunt Irma Visits
7 13384 1 2 1 The Work Outing
8 13756 2 2 1 Return of the Golden Child
9 13763 3 2 1 Moss and the German
10 13770 4 2 1 The Dinner Party
11 13777 5 2 1 Smoke and Mirrors
12 13784 6 2 1 Men Without Women
13 14204 1 3 1 From Hell
14 14211 2 3 1 Are We Not Men?
15 14218 3 3 1 Tramps Like Us
16 14225 4 3 1 The Speech
17 14232 5 3 1 Friendface
18 14239 6 3 1 Calendar Geeks
19 14785 1 4 1 Jen The Fredo
20 14792 2 4 1 The Final Countdown

Работа с YDB с помощью Jupyter Notebook

Jupyter Notebook - это инструмент с открытым исходным кодом для создания общедоступных документов, сочетающий в себе код, описания на простом языке, данные, богатую визуализацию и интерактивные элементы управления.

Диалект ydb-sqlalchemy позволяет работать с YDB напрямую из таких инструментов как:

- Pandas
- JupySQL

Пример

Подробный пример работы доступен в специальном ноутбуке.

Требования для запуска:

1. Python 3.8+
2. Jupyter Notebook
3. Существующий кластер YDB, однонодовая инсталляция из быстрого старта будет достаточной

Для запуска примера скачайте файл ноутбука YDB SQLAlchemy - Jupyter Notebook Example.ipynb, откройте его в Jupyter и последовательно пройдитесь по каждой ячейке, выполняя код по мере необходимости.

Визуализация данных (Business Intelligence, BI)

Среда	Уровень поддержки	Инструкция
Apache Superset	ydb-sqlalchemy	Инструкция
DataLens	Полный	Инструкция
Grafana	Полный	Инструкция

Apache Superset

Apache Superset — современная платформа для анализа и визуализации данных. В этой статье описано, как создавать визуализации на основе данных, хранящихся в YDB.

Установка необходимых зависимостей

Для работы с YDB необходимо установить драйвер ydb-sqlalchemy.

Способ установки зависит от способа развёртывания Superset. Дополнительную информацию см. в официальной документации Superset.

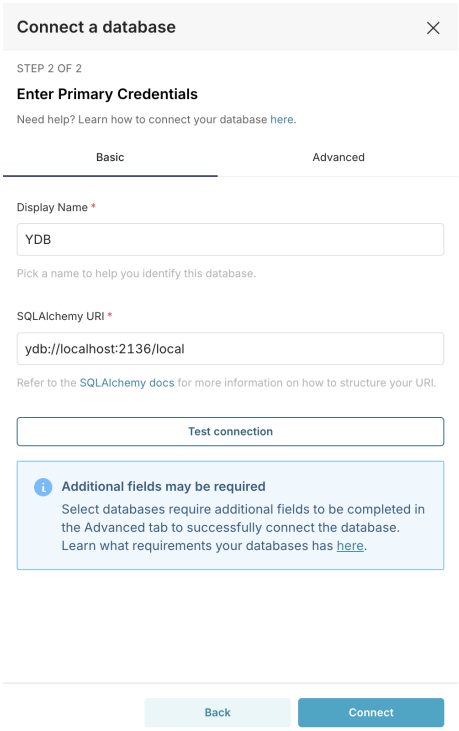
Создание подключения к YDB

Нативное подключение с помощью SQLAlchemy-драйвера

Чтобы создать подключение к YDB из Apache Superset **версии 5.0.0 и выше**, выполните следующие шаги:

1. В верхнем меню Apache Superset наведите курсор на **Settings** и выберите в выпадающем списке пункт **Database Connections**.
2. Нажмите кнопку **+ DATABASE**.

   Откроется окно мастера **Connect a database**.
3. На первом шаге мастера выберите **YDB** из списка **Supported databases**. Если опция **YDB** недоступна, убедитесь, что установлены все необходимые компоненты.
4. На втором шаге мастера введите данные для подключения к YDB в следующие поля:
   - **Display Name** — наименование соединения с YDB в Apache Superset;
   - **SQLAlchemy URI** — строка вида `ydb://{host}:{port}/{database_name}`, где **host** и **port** — соответствующие значения из эндпоинта кластера YDB, **database_name** — путь к базе данных.



5. Опционально, с помощью поля `Secure Extra` на вкладке `Advanced / Security` можно указать параметры аутентификации.

   Определите параметры следующим образом:

   Пароль

   ```
   {
       "credentials": {
           "username": "...",
           "password": "..."
       }
   }
   ```

   Токен

   ```
   {
       "credentials": {
           "token": "...",
       }
   }
   ```

Сервисный аккаунт

```
{
    "credentials": {
        "service_account_json": {
            "id": "...",
            "service_account_id": "...",
            "created_at": "...",
            "key_algorithm": "...",
            "public_key": "...",
            "private_key": "..."
        }
    }
}
```

6. Нажмите кнопку **CONNECT**.
7. Нажмите кнопку **FINISH**, чтобы сохранить подключение.

Для дополнительной информации о возможностях конфигурации соединения обратитесь к разделу YDB официальной документации.

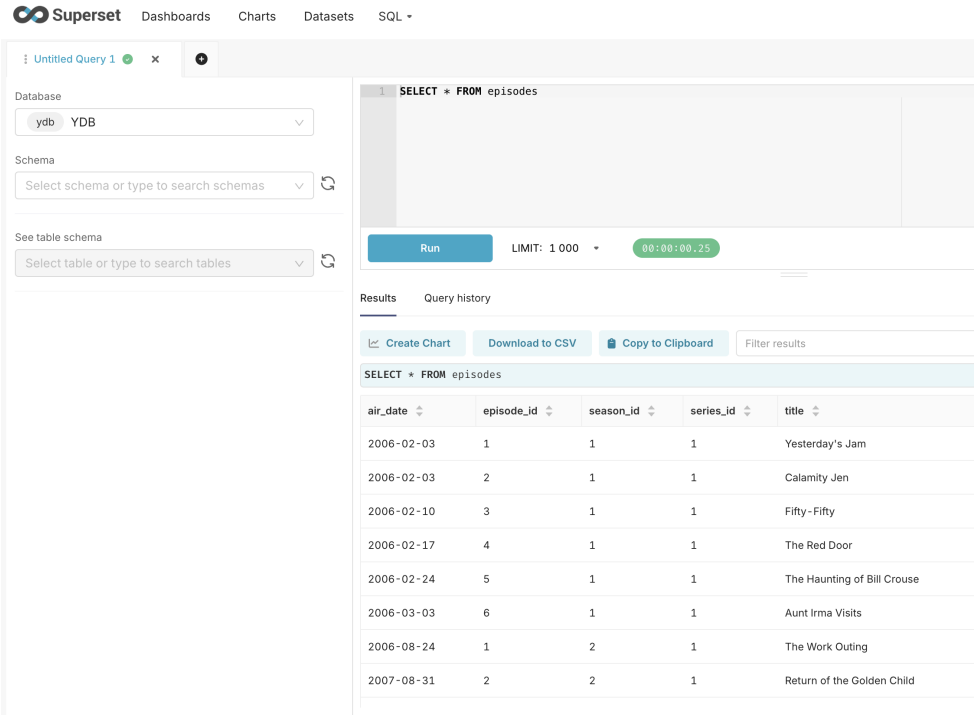
Создание набора данных (dataset)

Чтобы создать набор данных из таблицы YDB, выполните следующие шаги:

1. В верхнем меню Apache Superset наведите курсор на кнопку **+** и выберите в выпадающем списке пункт **SQL query**.
2. В выпадающем списке **DATABASE** выберите подключение к YDB.
3. Введите текст SQL-запроса в правой части страницы. Например, `SELECT * FROM <наименование_таблицы>`.

> **Совет**
>
> Если вы хотите создать набор данных из таблицы, которая расположена в поддиректории YDB, необходимо указать путь к таблице в самом наименовании таблицы. Например:
>
> ```
> SELECT * FROM "<путь/к/таблице/наименование_таблицы>";
> ```

4. Нажмите кнопку **RUN**, чтобы проверить SQL-запрос.



5. Нажмите на стрелку рядом с кнопкой **SAVE** и выберите **Save dataset** в выпадающем списке.

   Откроется диалоговое окно **Save or Overwrite Dataset**.
6. В открывшемся окне **Save or Overwrite Dataset** выберите **Save as new**, введите наименование набора данных и нажмите **SAVE & EXPLORE**.

После создания наборов данных вы можете использовать данные из YDB для создания диаграмм в Apache Superset. См. документацию Apache Superset.

Создание диаграммы

Теперь давайте создадим пример диаграммы с использованием набора данных из таблицы `episodes`, которая описана в Туториале по YQL.

Таблица `episodes` содержит следующие колонки:

- series_id;

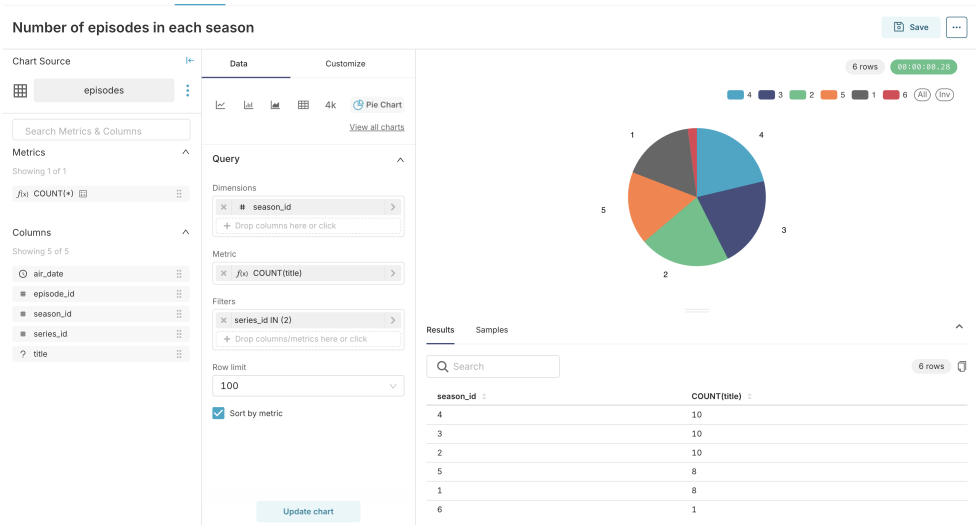
- season_id;
- episode_id;
- title;
- air_date.

Предположим, что мы хотим построить круговую диаграмму, в которой бы было видно, сколько серий содержит каждый сезон сериала.

Чтобы создать диаграмму, выполните следующие шаги:

1. В верхнем меню Apache Superset наведите курсор на **+** и выберите в выпадающем списке пункт **Chart**.
2. В выпадающем списке **Choose a dataset**, выберите набор данных из таблицы `episodes`.
3. На панели **Choose chart type**, выберите тип диаграммы `Pie chart`.
4. Нажмите кнопку **CREATE NEW CHART**.
5. На панели **Query** настройте диаграмму:
   - В выпадающем списке **DIMENSIONS** выберите колонку `season_id`.
   - В поле **METRIC** введите функцию `COUNT(title)`.
   - В поле **FILTERS** введите фильтр `series_id in (2)`.
6. Нажмите кнопку **CREATE CHART**.

   Круговая диаграмма появится на панели справа.



7. Нажмите кнопку **SAVE**.

   Откроется диалоговое окно **Save chart**.
8. В открывшемся окне **Save chart** в поле **CHART NAME** введите наименование диаграммы.
9. Нажмите кнопку **SAVE**.

DataLens

DataLens — инструмент бизнес-аналитики (BI) и визуализации данных с открытым исходным кодом, который позволяет анализировать и отображать данные из различных источников, включая YDB. DataLens позволяет описывать модели данных, строить графики и визуализации, собирать дашборды и обеспечивать коллективный доступ к аналитике.

Предварительные требования

DataLens должен быть развёрнут и настроен.

> **Примечание**
>
> В этой статье рассматривается интеграция YDB и DataLens, развёрнутых самостоятельно. Документацию по интеграции соответствующих управляемых сервисов см. в документации Yandex Cloud.

Добавление подключения к базе данных YDB

Чтобы создать подключение к YDB:

1. Перейдите на страницу воркбука или создайте новый.
2. В правом верхнем углу нажмите **Создать** → **Подключение**.
3. Выберите тип подключения **YDB**.
4. Укажите тип аутентификации:

   Анонимный

   - **Имя хоста** — хостнейм для подключения к YDB.
   - **Порт** — порт подключения к YDB. По умолчанию — 2135.
   - **Путь к базе данных** — имя базы данных для подключения.

   Пароль

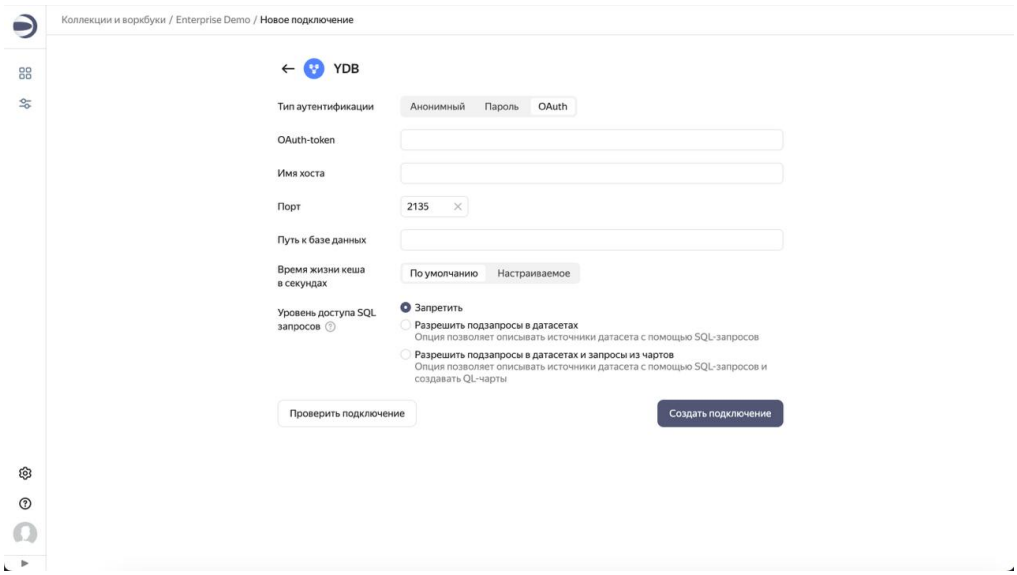
   - **Имя хоста** — хостнейм для подключения к YDB.
   - **Порт** — порт подключения к YDB. По умолчанию — 2135.
   - **Путь к базе данных** — имя базы данных для подключения.
   - **Имя пользователя** — имя пользователя для подключения к YDB.
   - **Пароль** — пароль пользователя.

   OAuth

   - **OAuth-токен** — OAuth-токен для доступа к YDB.
   - **Имя хоста** — хостнейм для подключения к YDB.
   - **Порт** — порт подключения к YDB. По умолчанию — 2135.
   - **Путь к базе данных** — имя базы данных для подключения.

   - **Время жизни кэша в секундах** — установите время жизни кэша или оставьте значение по умолчанию. Рекомендуемое значение — 300 секунд (5 минут).
   - **Уровень доступа к SQL-запросам** — позволяет использовать собственные SQL-запросы для создания датасета.
5. Нажмите **Создать подключение**.
6. Укажите имя подключения и нажмите **Создать**.
7. Перейдите к созданию датасета.

Пример



Плагин-источник данных Grafana для интеграции с YDB

YDB data source plugin позволяет использовать Grafana для запросов и визуализации данных из YDB.

Установка

Для работы плагина понадобится Grafana версии не ниже `9.2`.

Следуйте документации Grafana, чтобы установить плагин под названием `ydb-grafana-datasource-plugin`.

Конфигурация

YDB пользователь для источника данных

Настройте аккаунт пользователя YDB с правами **только на чтение** (подробнее об правах доступа) и доступом до баз данных и таблиц, к которым потребуется делать запросы.

> **Важно**
>
> Обратите внимание, что Grafana не валидирует запросы с точки зрения их безопасности. Запросы могут содержать в себе любые SQL выражения, включая модифицирующие данные инструкции.

Протокол передачи данных

Плагин поддерживает транспортные протоколы gRPC и gRPCS.

Настройка через UI

После установки плагина следуйте этим инструкциям, чтобы добавить новый источник данных YDB, а затем укажите настройки для соединения с кластером.

Настройка через файл конфигурации

Источник данных можно настроить с помощью файла конфигурации. Как это сделать описано в документации Grafana.

Аутентификация

Плагин поддерживает следующие типы аутентификации в YDB: Anonymous, Access Token, Metadata, Service Account Key и Static Credentials.

Пример настройки источника данных YDB с использованием аутентификации в через логин/пароль:

```
apiVersion: 1
datasources:
  - name: YDB
    type: ydbtech-ydb-datasource
    jsonData:
      authKind: '<password>'
      endpoint: 'grpcs://<hostname>:2135'
      dbLocation: '<location_to_db>'
      user: '<username>'
    secureJsonData:
      password: '<userpassword>'
      certificate: |
        <full content of *.pem file>
```

Поддерживаемые поля для создания соединения:

Имя	Описание	Тип
authKind	Тип аутентификации	`"Anonymous"`, `"ServiceAccountKey"`, `"AccessToken"`, `"UserPassword"`, `"MetaData"`
endpoint	Эндпоинт	`string`
dbLocation	Путь к базе	`string`
user	Имя пользователя	`string`
serviceAccAuthAccessKey	Ключ доступа для сервисного аккаунта	`string` (защищенное поле)
accessToken	Access токен	`string` (защищенное поле)
password	Пароль	`string` (защищенное поле)
certificate	Если на вашем кластере YDB используются самоподписанные сертификаты TLS, то для соединения с YDB необходимо указать сертификат сертификационного центра, через который они были выпущены	`string` (защищенное поле)

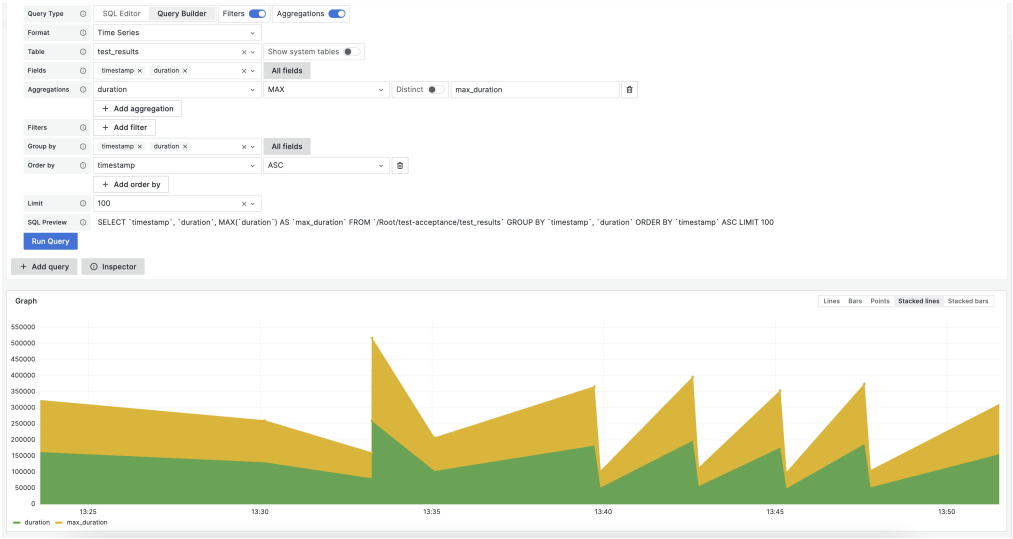
Написание запросов

Для запроса данных из баз YDB используется YQL.

Редактор запросов позволяет получать данные в следующих представлениях: временные ряды, таблицы и логи.

Временные ряды

Визуализировать данные как временные ряды возможно при условии наличия в результатах запроса одного поля с типами `Date`, `Datetime` или `Timestamp` (на текущий момент поддержана работа со временем только во временной зоне UTC) и как минимум одного поля с типом `Int64`, `Int32`, `Int16`, `Int8`, `Uint64`, `Uint32`, `Uint16`, `Uint8`, `Double` или `Float`. Визуализацию в виде временных рядов можно выбрать с помощью настроек. Все остальные колонки интерпретируются как значения.



Многострочные временные ряды

Чтобы создать многострочный временной ряд, результаты запроса должны содержать в себе как минимум три поля:

- поле с типом данных `Date`, `Datetime` или `Timestamp` (на текущий момент поддержана работа со временем только во временной зоне UTC)
- метрика - поле с типом `Int64`, `Int32`, `Int16`, `Int8`, `Uint64`, `Uint32`, `Uint16`, `Uint8`, `Double` или `Float`
- либо метрика, либо поле с типом `String` или `Utf8` - значение для разбиения метрик по отдельным рядам.

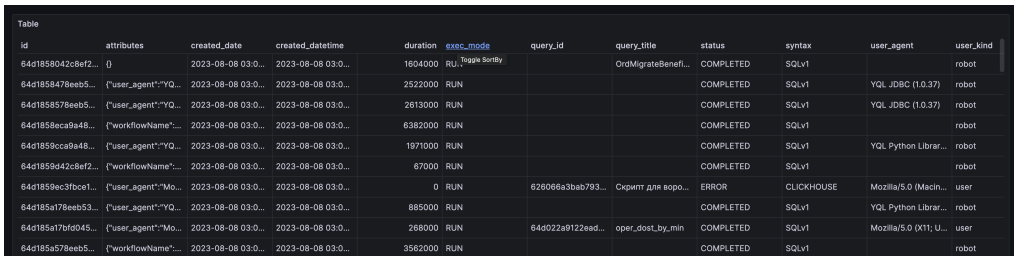
Например:

```
SELECT
    `timestamp`,
    `responseStatus`
    AVG(`requestTime`) AS `avgReqTime`
FROM `/database/endpoint/my-logs`
GROUP BY `responseStatus`, `timestamp`
ORDER BY `timestamp`
```

> **Совет**
>
> Для таких запросов использование колоночных таблиц может быть полезным с точки зрения производительности.

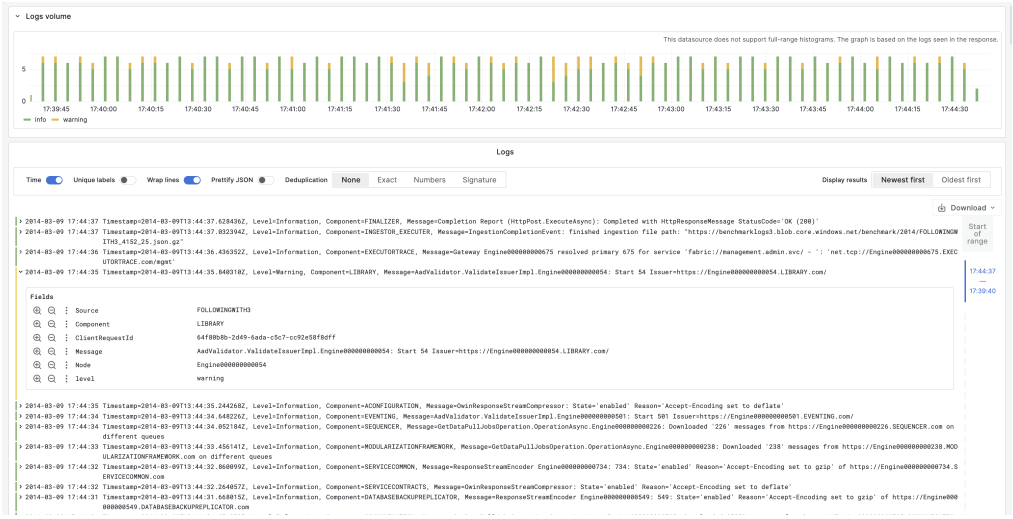
Таблицы

Табличное представление доступно для любого валидного YDB запроса, возвращающего ровно один набор результатов.



Визуализация логов

Для визуализации данных в виде логов запрос должен возвращать поле с типом данных `Date`, `Datetime` или `Timestamp` и прое с типом `String`. Выбрать тип визуализации можно с помощью настроек. По умолчанию только первое встреченное текстовое поле трактуется как строка лога, но это поведение может быть изменено с помощью конструктора запросов.



Макросы

Чтобы упростить синтаксис и получить возможность динамически изменять параметры (например, значение временного диапазона), запрос может содержать в себе макросы.

В запросе могут содержаться два вида макросов - уровня Grafana и уровня YQL. Перед отправкой запроса в YDB, плагин проанализирует текст запроса и заменит макросы уровня Grafana на конкретные значения. После этого макросы уровня YQL будут обработаны на сервере YDB.

Пример запроса с макросом, который позволяет использовать временной фильтр Grafana.

```
SELECT `timeCol`
FROM `/database/endpoint/my-logs`
WHERE $__timeFilter(`timeCol`)
```

```
SELECT `timeCol`
FROM `/database/endpoint/my-logs`
WHERE $__timeFilter(`timeCol` + Interval("PT24H"))
```

Макрос	Описание	Пример вывода
`$__timeFilter(columnName)`	Заменяется условием, которое фильтрует данные в указанной колонке или результате выражения на основании временного диапазона, заданного на панели в микросекундах	`foo >= CAST(1636717526371000 AS Timestamp) AND foo <= CAST(1668253526371000 AS Timestamp)' )`
`$__fromTimestamp`	Заменяется временем начала диапазона, заданного на панели в формате Timestamp	`CAST(1636717526371000 AS Timestamp)`
`$__toTimestamp`	Заменяется временем окончания диапазона, заданного на панели в формате Timestamp	`CAST(1636717526371000 AS Timestamp)`
`$__varFallback(condition, $templateVar)`	Заменяется первым параметром в том случае, если второй параметр не определен.	`$__varFallback('foo', $bar)` `foo` если переменная `bar` не определена, или значение переменной `$bar`

Шаблоны и переменные

Чтобы добавить новую переменную для YDB запроса, следуйте
инструкции.
После создания переменная может быть использована в запросе к YDB с помощью специального синтаксиса.
Для более подробной информации о переменных, см. документацию Grafana.

Дополнительная информация

- Добавить Аннотации.
- Настроить и использовать Шаблоны и переменные.
- Добавить Трансформации.
- Настроить Уведомления.

Оркестрация

Среда	Инструкция
Apache Airflow™	Инструкция

Apache Airflow™

Интеграция YDB с Apache Airflow™ позволяет автоматизировать сложные рабочие процессы и управлять ими. Apache Airflow™ предоставляет возможности для планирования задач, мониторинга их выполнения и управления зависимостями между ними — оркестрацией. Использование Airflow для оркестрации задач, таких как загрузка данных в YDB, выполнение запросов и управление транзакциями, позволяет автоматизировать и оптимизировать операционные процессы. Это особенно важно для задач ETL, где данные больших объемов требуют регулярного извлечения, преобразования и загрузки.

Для работы под управлением Apache Airflow™ YDB предоставляет пакет провайдера apache-airflow-providers-ydb. Задания Apache Airflow™ представляют собой приложения на языке Python, состоящие из набора операторов Apache Airflow™ и их зависимостей, определяющих порядок выполнения.

Установка

Для корректной работы пакета `apache-airflow-providers-ydb` необходимо на всех хостах Apache Airflow™ выполнить следующие команды:

```
pip install ydb apache-airflow-providers-ydb
```

Для работы требуется Python версии не ниже, чем 3.8.

Объектная модель

Пакет `airflow.providers.ydb` содержит набор компонентов для взаимодействия с YDB:

- Оператор YDBExecuteQueryOperator для интеграции задач в планировщик Apache Airflow™.
- Хук YDBHook для прямого взаимодействия с YDB.

YDBExecuteQueryOperator

Для выполнения запросов к YDB используется Apache Airflow™ оператор `YDBExecuteQueryOperator`.

Обязательные аргументы:

- `task_id` — название задания Apache Airflow™.
- `sql` — текст SQL-запроса, который необходимо выполнить в YDB.

Опциональные аргументы:

- `ydb_conn_id` — идентификатор подключения с типом `YDB`, содержащий параметры соединения с YDB. Если не указан, то используется соединение с именем `ydb_default`. Соединение `ydb_default` предустанавливается в составе Apache Airflow™, отдельно его заводить не нужно.
- `is_ddl` — признак, что выполняется SQL DDL запрос. Если аргумент не указан, или установлен в `False`, то будет выполняться SQL DML запрос.
- `params` — словарь параметров запроса.

Пример:

```
ydb_operator = YDBExecuteQueryOperator(task_id="ydb_operator", sql="SELECT 'Hello, world!'")
```

В данном примере создается задание Apache Airflow™ с идентификатором `ydb_operator`, которое выполняет запрос `SELECT 'Hello, world!'`.

YDBHook

Для выполнения низкоуровневых команд в YDB используется Apache Airflow™ класс `YDBHook`.

Опциональные аргументы:

- `ydb_conn_id` — идентификатор подключения с типом `YDB`, содержащий параметры соединения с YDB. Если не указан, то используется соединение с именем `ydb_default`. Соединение `ydb_default` предустанавливается в составе Apache Airflow™, отдельно его заводить не нужно.
- `is_ddl` — признак, что выполняется SQL DDL запрос. Если аргумент не указан, или установлен в `False`, то будет выполняться SQL DML запрос.

`YDBHook` поддерживает следующие методы:

- bulk_upsert;
- get_conn.

bulk_upsert

Выполняет пакетную вставку данных в таблицы YDB.

Обязательные аргументы:

- `table_name` — название таблицы YDB, куда будет выполняться вставка данных.
- `rows` — массив строк для вставки.
- `column_types` — описание типов колонок.

Пример:

```
hook = YDBHook(ydb_conn_id=...)
column_types = (
        ydb.BulkUpsertColumns()
        .add_column("pet_id", ydb.OptionalType(ydb.PrimitiveType.Int32))
        .add_column("name", ydb.PrimitiveType.Utf8)
        .add_column("pet_type", ydb.PrimitiveType.Utf8)
        .add_column("birth_date", ydb.PrimitiveType.Utf8)
        .add_column("owner", ydb.PrimitiveType.Utf8)
```

```
    )

rows = [
    {"pet_id": 3, "name": "Lester", "pet_type": "Hamster", "birth_date": "2020-06-23", "owner": "Lily"},
    {"pet_id": 4, "name": "Quincy", "pet_type": "Parrot", "birth_date": "2013-08-11", "owner": "Anne"},
]
hook.bulk_upsert("pet", rows=rows, column_types=column_types)
```

В данном примере создается объект `YDBHook`, через который выполняется операция пакетной вставки данных `bulk_upsert`.

get_conn

Возвращает объект `YDBConnection`, реализующий интерфейс `DbApiConnection` для работы с данными. Класс `DbApiConnection` обеспечивает стандартизированный интерфейс для взаимодействия с базой данных, позволяющий выполнять такие операции, как подключение, выполнение SQL-запросов и управление транзакциями, независимо от конкретной системы управления базами данных.

Пример:

```
hook = YDBHook(ydb_conn_id=...)

# Выполняем SQL-запрос и получаем курсор
connection = hook.get_conn()
cursor = connection.cursor()
cursor.execute("SELECT * from pet;")

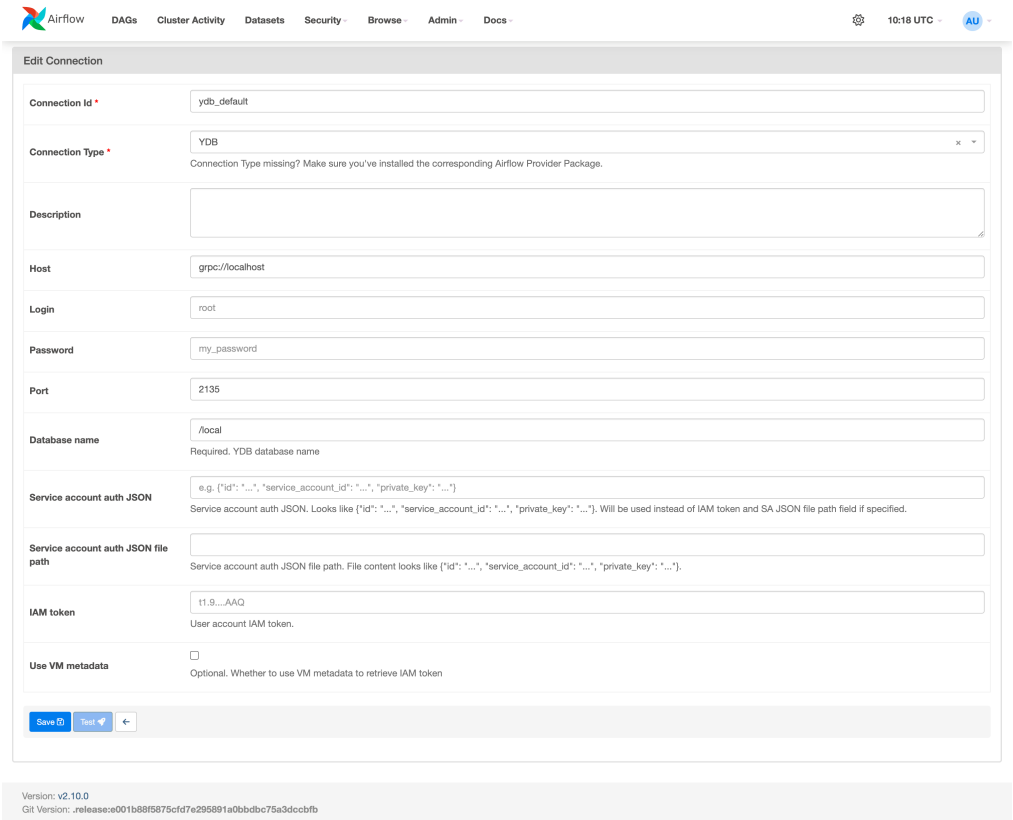
# Извлекаем результат и имена колонок
result = cursor.fetchall()
columns = [desc[0] for desc in cursor.description]

# Закрываем курсор и соединение
cursor.close()
connection.close()
```

В данном примере создается объект `YDBHook`, у созданного объекта запрашивается объект `YDBConnection`, через который выполняется чтение данных и получение списка колонок.

Подключение к YDB

Для подключения к YDB необходимо создать новое или отредактировать существующее подключение Apache Airflow™ с типом `YDB`.



Где:

- `Connection Id` - название подключения Apache Airflow™.
- `Host` - протокол и адрес кластера YDB.
- `Port` - порт для подключения к кластеру YDB.
- `Database name` - название базы данных YDB.

Укажите реквизиты для одного из следующих способов аутентификации на кластере YDB:

- `Login` и `Password` - укажите реквизиты пользователя для аутентификации по логину и паролю.
- `Service account auth JSON` - укажите значение `Service Account Key`.

- `Service account auth JSON file path` - укажите путь к файлу, содержащему `Service Account Key`.
- `IAM token` - укажите IAM токен.
- `Use VM metadata` - указание использовать метаданные виртуальной машины.

Соответствие YQL и Python-типов

Ниже приведены правила преобразования YQL-типов в Python-результаты. Типы, не указанные в списке ниже, не поддерживаются.

Скалярные типы

YQL-тип	Python-тип	Пример в Python
`Int8`, `Int16`, `Int32`, `Uint8`, `Uint16`, `Uint32`, `Int64`, `Uint64`	`int`	`647713`
`Bool`	`bool`	True
`Float`, `float`	`float` NaN и Inf представляются в виде `None`	`7.88731023` `None`
`Decimal`	`Decimal`	`45.23410083`
`Utf8`	`str`	`Текст строки`
`String`	`str`	`Текст строки`

Сложные типы

YQL-тип	Python-тип	Пример в Python
`Json`, `JsonDocument`	`str` (весь узел вставляется как строка)	`{"a":[1,2,3]}`
`Date`	`datetime.date`	`2022-02-09`
`Datetime`, `Timestamp`	`datetime.datetime`	`2022-02-09 10:13:11`

Опциональные типы

YQL-тип	Python-тип	Пример в Python
`Optional`	Оригинальный тип или None	`1`

Контейнеры

YQL-тип	Python-тип	Пример в Python
`List<Type>`	`list`	`[1,2,3,4]`
`Dict<KeyType, ValueType>`	`dict`	`{key1: "value1", key2: "value2"}`
`Set<KeyType>`	`set`	`set(key_value1, key_value2)`
`Tuple<Type1, Type2>`	`tuple`	`(element1, element2)`
`Struct<Name:Utf8,Age:Int32>`	`dict`	`{ "Name": "value1", "Age": value2 }`

Специальные типы

YQL-тип	Python-тип
`Void`, `Null`	`None`
`EmptyList`	`[]`
`EmptyDict`	`{}`

Пример

Для выполнения запросов к YDB в составе пакета содержится оператор Apache Airflow™ `YDBExecuteQueryOperator` и хук `YDBHook`.

В примере ниже создается задание `create_pet_table`, создающее таблицу в YDB. После успешного создания таблицы вызывается задание `populate_pet_table`, заполняющее таблицу данными с помощью команд `UPSERT`, и задание `populate_pet_table_via_bulk_upsert`, заполняющее таблицу с помощью команд пакетной вставки данных `bulk_upsert`. После выполнения вставки данных выполняется операция чтения с помощью задания `get_all_pets` и задание для параметризованного чтения данных `get_birth_date`.



Для выполнения запросов к базе данных YDB используется предварительно созданное соединение с YDB типа YDB Connection с именем `test_ydb_connection`.

```python
from __future__ import annotations

import datetime

import ydb
from airflow import DAG
from airflow.decorators import task
from airflow.providers.ydb.hooks.ydb import YDBHook
from airflow.providers.ydb.operators.ydb import YDBExecuteQueryOperator

@task
def populate_pet_table_via_bulk_upsert():
    hook = YDBHook(ydb_conn_id="test_ydb_connection")
    column_types = (
        ydb.BulkUpsertColumns()
        .add_column("pet_id", ydb.OptionalType(ydb.PrimitiveType.Int32))
        .add_column("name", ydb.PrimitiveType.Utf8)
        .add_column("pet_type", ydb.PrimitiveType.Utf8)
        .add_column("birth_date", ydb.PrimitiveType.Utf8)
        .add_column("owner", ydb.PrimitiveType.Utf8)
    )

    rows = [
        {"pet_id": 3, "name": "Lester", "pet_type": "Hamster", "birth_date": "2020-06-23", "owner": "Lily"},
        {"pet_id": 4, "name": "Quincy", "pet_type": "Parrot", "birth_date": "2013-08-11", "owner": "Anne"},
    ]
    hook.bulk_upsert("pet", rows=rows, column_types=column_types)


with DAG(
    dag_id="ydb_demo_dag",
    start_date=datetime.datetime(2020, 2, 2),
    schedule="@once",
    catchup=False,
) as dag:
    create_pet_table = YDBExecuteQueryOperator(
        task_id="create_pet_table",
        sql="""
            CREATE TABLE pet (
            pet_id INT,
            name TEXT NOT NULL,
            pet_type TEXT NOT NULL,
            birth_date TEXT NOT NULL,
            owner TEXT NOT NULL,
            PRIMARY KEY (pet_id)
            );
          """,
        is_ddl=True,  # must be specified for DDL queries
        ydb_conn_id="test_ydb_connection"
    )

    populate_pet_table = YDBExecuteQueryOperator(
        task_id="populate_pet_table",
        sql="""
              UPSERT INTO pet (pet_id, name, pet_type, birth_date, owner)
              VALUES (1, 'Max', 'Dog', '2018-07-05', 'Jane');

              UPSERT INTO pet (pet_id, name, pet_type, birth_date, owner)
              VALUES (2, 'Susie', 'Cat', '2019-05-01', 'Phil');
            """,
        ydb_conn_id="test_ydb_connection"
    )

    get_all_pets = YDBExecuteQueryOperator(task_id="get_all_pets", sql="SELECT * FROM pet;", ydb_conn_id="tes
t_ydb_connection")

    get_birth_date = YDBExecuteQueryOperator(
        task_id="get_birth_date",
        sql="SELECT * FROM pet WHERE birth_date BETWEEN '{{params.begin_date}}' AND '{{params.end_date}}'",
        params={"begin_date": "2020-01-01", "end_date": "2020-12-31"},
        ydb_conn_id="test_ydb_connection"
    )

    (
        create_pet_table
        >> populate_pet_table
        >> populate_pet_table_via_bulk_upsert()
        >> get_all_pets
        >> get_birth_date
    )
```

Поставка данных

Система поставки	Инструкция
FluentBit	Инструкция
LogStash	Инструкция
Kafka Connect Sink	Инструкция

Потоковая поставка данных

Система поставки	Инструкция
Apache Kafka API	Инструкция
Apache Kafka Connect	Инструкция

Сбор логов с помощью FluentBit

В разделе представлена реализация интеграции между YDB и инструментом захвата логов FluentBit с целью сохранения данных логов в YDB для последующего просмотра или анализа.

Введение

FluentBit – инструмент, который может собирать текстовые данные, манипулировать ими (изменять, преобразовывать, объединять) и отправлять в различные хранилища для дальнейшей обработки. Для поставки логов в таблицы YDB с помощью FluentBit разработана библиотека-плагин. Исходный код библиотеки доступен здесь.

Чтобы развернуть схему доставки логов с помощью FluentBit с последующим сохранением их в YDB, необходимо:

1. Создать таблицы YDB для хранения логов
2. Развернуть FluentBit и библиотеку-плагин для поддержки YDB
3. Сконфигурировать FluentBit для сбора и обработки логов в соответствии с документацией
4. Настроить сохранение логов в таблицы YDB

Создание таблиц для хранения логов

Для сохранения логов необходимо создать таблицы в выбранной базе данных YDB. Структура создаваемых таблиц определяется набором полей конкретного лога, поставляемого с помощью FluentBit. Типичный набор полей включает в себя:

- метку времени;
- уровень критичности сообщения;
- имя хоста, с которого получено сообщение;
- наименование сервиса;
- текст сообщения либо его структурированные данные в формате JSON.

В таблицах YDB обязательно наличие первичного ключа, уникально идентифицирующего конкретную запись. Поскольку метка времени не обеспечивает уникальной идентификации записи лога, в состав хранимых полей обычно добавляют значение хеш-кода над данными сообщения, которое включают в состав первичного ключа.

Поддерживается хранение логов как в строковых, так и колоночных таблицах. Рекомендуется использование колоночных таблиц, как обеспечивающих более эффективное сканирование данных при выполнении запросов.

Пример команды создания строковой таблицы для хранения логов:

```
CREATE TABLE `fluent-bit/log` (
    `timestamp`         Timestamp NOT NULL,
    `hostname`          Text NOT NULL,
    `input`             Text NOT NULL,
    `datahash`          Uint64 NOT NULL,
    `level`             Text NULL,
    `message`           Text NULL,
    `other`             JsonDocument NULL,
    PRIMARY KEY (
         `datahash`, `timestamp`, `hostname`, `input`
    )
);
```

Пример команды создания колоночной таблицы для хранения логов:

```
CREATE TABLE `fluent-bit/log` (
    `timestamp`         Timestamp NOT NULL,
    `hostname`          Text NOT NULL,
    `input`             Text NOT NULL,
    `datahash`          Uint64 NOT NULL,
    `level`             Text NULL,
    `message`           Text NULL,
    `other`             JsonDocument NULL,
    PRIMARY KEY (
         `timestamp`, `hostname`, `input`, `datahash`
    )
) PARTITION BY HASH(`timestamp`, `hostname`, `input`)
  WITH (STORE = COLUMN);
```

Команда создание колоночной таблицы отличается от команды создания строковой таблицы следующими деталями:

- указанием в двух последних строках колоночного режима хранения и отдельного ключа партиционирования;
- колонка `timestamp` указана первой в составе первичного ключа таблицы, что является оптимальным и рекомендованным для колоночных таблиц, и не рекомендовано для строчных таблиц. См. также рекомендации по выбору первичного ключа для колоночных таблиц и для строчных таблиц.

Опционально можно настроить TTL для строк таблицы, ограничив срок хранения и обеспечив автоматическое удаление устаревших данных. Для этого при создании таблицы в секции `WITH` указываются настройки TTL, например `TTL = Interval("P14D") ON timestamp` устанавливает срок хранения записей в 14 дней на основе значения поля `timestamp`.

Развертывание FluentBit и настройка сбора логов

Развертывание FluentBit осуществляется в соответствии с собственной документацией.

Плагин для поддержки YDB в FluentBit доступен в репозитории вместе с инструкциями по сборке. Для установки в формате контейнера предусмотрен Docker-образ `ghcr.io/ydb-platform/fluent-bit-ydb`.

Общая логика и порядок настройки процессов получения, обработки и поставки логов в среде FluentBit изложены в документации FluentBit.

Настройка записи логов в YDB

Перед использованием выходного плагина для YDB, его необходимо включить в настройках FluentBit. Перечень используемых плагинов FluentBit определяется в отдельном файле (например, `plugins.conf`), путь к которому устанавливается параметром `plugins_file` секции `SERVICE` основного файла настроек FluentBit. Пример файла плагинов со ссылкой на библиотеку плагина для записи в YDB (путь к библиотеке в вашей системе может отличаться):

```
# plugins.conf
[PLUGINS]
    Path /usr/lib/fluent-bit/out_ydb.so
```

Состав конфигурационных параметров, поддерживаемых выходным плагином YDB для FluentBit, приведен в таблице ниже.

Ключ	Описание
`Name`	Тип плагина, константа `ydb`
`Match`	(опционально) Выражение для отбора тегов записей, направляемых в YDB
`ConnectionURL`	URL для подключения к БД, включая протокол, хост, номер порта и путь к базе данных (см. документацию)
`TablePath`	Путь к таблице, начиная от корня базы данных (пример: `fluent-bit/log`)
`Columns`	JSON-структура, состоящая из пар имен колонок, и устанавливающая соответствие между исходными колонками записи и целевыми колонками таблицы. Может включать перечисленные далее служебные псевдо-колонки
`CredentialsAnonymous`	Устанавливается в значение `1` для использования анонимной аутентификации
`CredentialsToken`	Значение токена аутентификации, для использования аутентификации непосредственно по значению токена
`CredentialsYcMetadata`	Устанавливается в значение `1` для использования аутентификации через метаданные виртуальной машины
`CredentialsStatic`	Логин и пароль для использования статического режима аутентификации, указанные в формате `Логин:Пароль@`
`CredentialsYcServiceAccountKey`	Путь к файлу ключа сервисного аккаунта, для аутентификации по ключу сервисного аккаунта
`CredentialsYcServiceAccountKeyJson`	JSON-данные ключа сервисного аккаунта, для аутентификации по ключу сервисного аккаунта без указания имени файла (удобно в среде K8s)
`Certificates`	Путь к файлу с сертификатом CA, либо непосредственно содержимое сертификата CA
`LogLevel`	Уровень логирования для плагина, одно из значений `disabled` (по умолчанию), `trace`, `debug`, `info`, `warn`, `error`, `fatal` либо `panic`

Для использования в карте сопоставления колонок (параметр `Columns`), в дополнение к полям записи FluentBit, доступны следующие служебные псевдо-колонки:

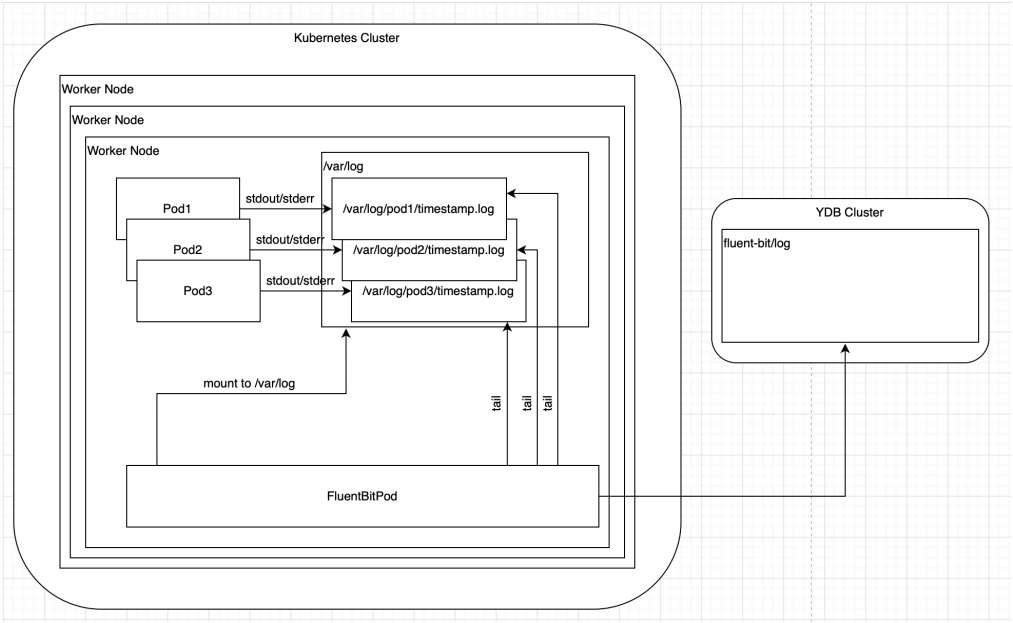
- `.timestamp` - метка времени сообщения (обязательная)
- `.input` - имя входного потока сообщений (обязательная)
- `.hash` - uint64 хеш-код, вычисленный над всеми полями сообщения, кроме псевдо-колонок (опциональная)
- `.other` - документ JSON, содержащий все поля сообщения, которые не были явным образом сопоставлены с конкретной выходной колонкой (опциональная)

Пример значения параметра `Columns`:

```
{".timestamp": "timestamp", ".input": "input", ".hash": "datahash", "log": "message", "level": "level", "hos
t": "hostname", ".other": "other"}
```

Сбор логов в кластере Kubernetes

FluentBit очень часто используется для сбора логов в кластерах Kubernetes. Принципиальная схема доставки логов запущенных приложений в Kubernetes с помощью FluentBit с последующим сохранением их в YDB выглядит следующим образом:



На этой схеме:

- Поды приложений пишут логи в stdout/stderr
- Текст из stdout/stderr сохраняется в виде файлов на рабочих узлах Kubernetes
- Под с FluentBit:
  - Монтирует каталог с лог-файлами рабочего узла Kubernetes
  - Вычитывает из лог-файлов содержимое
  - Обогащает записи дополнительными метаданными
  - Сохраняет записи в базу данных YDB

Таблица для хранения логов Kubernetes

Структура таблицы YDB для хранения логов Kubernetes:

```
CREATE TABLE `fluent-bit/log` (
    `timestamp`         Timestamp NOT NULL,
    `file`              Text NOT NULL,
    `pipe`              Text NOT NULL,
    `message`           Text NULL,
    `datahash`          Uint64 NOT NULL,
    `message_parsed`    JSON NULL,
    `kubernetes`        JSON NULL,

    PRIMARY KEY (
        `timestamp`, `file`, `datahash`
    )
) PARTITION BY HASH(`timestamp`, `file`)
  WITH (STORE = COLUMN, TTL = Interval("P14D") ON `timestamp`);
```

Предназначение колонок:

- `timestamp` – временная метка лога;
- `file` – название источника, из которого прочитан лог. В случае Kubernetes это будет имя файла, на worker ноде, в который записываются логи определенного pod;
- `pipe` – stdout или stderr поток, куда была осуществлена запись на уровне приложения;
- `datahash` – хеш-код, вычисленный над записью лога;
- `message` – текстовая часть записи лога;
- `message_parsed` – структурированное представление записи лога, если его удалось разобрать из текстовой части с помощью настроенных парсеров FluentBit;
- `kubernetes` – информация о pod, включая название, неймспейс и аннотации.

Опционально, можно установить TTL для строк таблицы, как показано в примере выше.

Конфигурация FluentBit

Перед развертыванием FluentBit в среде Kubernetes необходимо подготовить файл настроек (обычно `values.yaml`), в котором указываются параметры сбора и обработки логов. В этом разделе представлены необходимые пояснения по заполнению этого файла с примерами.

Необходимо указать репозиторий и версию образа контейнера FluentBit:

```
image:
  repository: ghcr.io/ydb-platform/fluent-bit-ydb
  tag: latest
```

В данном образе, по сравнению со стандартным, добавлена библиотека-плагин для поддержки YDB.

В следующих строках определены правила монтирования папок с логами в поды FluentBit:

```
volumeMounts:
  - name: config
    mountPath: /fluent-bit/etc/conf

daemonSetVolumes:
  - name: varlog
    hostPath:
      path: /var/log
  - name: varlibcontainers
    hostPath:
      path: /var/lib/containerd/containers
  - name: etcmachineid
    hostPath:
      path: /etc/machine-id
      type: File

daemonSetVolumeMounts:
  - name: varlog
    mountPath: /var/log
  - name: varlibcontainers
    mountPath: /var/lib/containerd/containers
    readOnly: true
  - name: etcmachineid
    mountPath: /etc/machine-id
    readOnly: true
```

Также необходимо переопределить команду и аргументы запуска FluentBit:

```
command:
  - /fluent-bit/bin/fluent-bit

args:
  - --workdir=/fluent-bit/etc
  - --plugin=/fluent-bit/lib/out_ydb.so
  - --config=/fluent-bit/etc/conf/fluent-bit.conf
```

Требуется настроить пайплайн сбора, преобразования и доставки логов:

```
config:
  inputs: |
    [INPUT]
        Name tail
        Path /var/log/containers/*.log
        multiline.parser cri
        Tag kube.*
        Mem_Buf_Limit 5MB
        Skip_Long_Lines On

  filters: |
    [FILTER]
        Name kubernetes
        Match kube.*
        Keep_Log On
        Merge_Log On
        Merge_Log_Key log_parsed
        K8S-Logging.Parser On
        K8S-Logging.Exclude On

    [FILTER]
        Name modify
        Match kube.*
        Remove time
        Remove _p

  outputs: |
    [OUTPUT]
        Name ydb
        Match kube.*
        TablePath fluent-bit/log
        Columns {".timestamp":"timestamp",".input":"file",".hash":"datahash","log":"message","log_parsed":"me
ssage_structured","stream":"pipe","kubernetes":"metadata"}
        ConnectionURL ${OUTPUT_YDB_CONNECTION_URL}
        CredentialsToken ${OUTPUT_YDB_CREDENTIALS_TOKEN}
```

Описание конфигурационных блоков:

- `inputs` - в этом блоке указываются откуда считывать и как разбирать логи. В данном случае, будет осуществляться чтение файликов `*.log` из папки `/var/log/containers/`, которая была смонтирована с хоста
- `filters` - в этом блоке указывается как будет осуществляться обработка логов. В данном случае: для каждого лога будут найдены соответствующие метаданные (в помощью kubernetes фильтра), а также, вырезаны неиспользуемые поля (_p, time)
- `outputs` - в этом блоке указывается, куда будут отгружены логи. В данном случае в таблицу `fluent-bit/log` в базе данных YDB. Параметры подключения к базе данных (в данном случае `ConnectionURL` и `CredentialsToken`) задаются с помощью переменных окружения – `OUTPUT_YDB_CONNECTION_URL`, `OUTPUT_YDB_CREDENTIALS_TOKEN`. При необходимости настройки аутентификации и состав используемых переменных окружения корректируются в зависимости от настроек используемого кластера YDB.

Переменные окружения определяются следующим образом:

```
env:
  - name: OUTPUT_YDB_CONNECTION_URL
    value: grpc://ydb-endpoint:2135/path/to/database
  - name: OUTPUT_YDB_CREDENTIALS_TOKEN
    valueFrom:
      secretKeyRef:
        key: token
        name: fluent-bit-ydb-plugin-token
```

Данные аутентификации необходимо сохранить в конфигурации кластера Kubernetes в виде секрета. Пример команды для создания секрета:

```
kubectl create secret -n ydb-fluent-bit-integration generic fluent-bit-ydb-plugin-token --from-literal=token=
<YDB TOKEN>
```

Развертывание FluentBit в кластере Kubernetes

HELM – это инструмент пакетирования и установки приложений в кластере Kubernetes. Для развертывания FluentBit необходимо добавить репозиторий с соответствующим чартом (сценарием установки) с помощью команды:

```
helm repo add fluent https://fluent.github.io/helm-charts
```

После этого установка FluentBit в кластер Kubernetes выполняется с помощью следующей команды:

```
helm upgrade --install fluent-bit fluent/fluent-bit \
  --version 0.37.1 \
  --namespace ydb-fluent-bit-integration \
  --create-namespace \
  --values values.yaml
```

В команде выше в аргументе `--values` указывается ранее подготовленный файл с настройками FluentBit.

Проверка установки

Проверяем что FluentBit запустился, читая его логи (должны отсутствовать записи уровня `[error]`):

```
kubectl logs -n ydb-fluent-bit-integration -l app.kubernetes.io/instance=fluent-bit
```

Проверяем, что записи в таблице YDB есть (появятся спустя примерно несколько минут после запуска FluentBit):

```
SELECT * FROM `fluent-bit/log` LIMIT 10 ORDER BY `timestamp` DESC
```

Очистка ресурсов

Для удаления FluentBit достаточно удалить Kubernetes namespace, в который была выполнена установка:

```
kubectl delete namespace ydb-fluent-bit-integration
```

Далее можно удалить таблицу с логами в базе данных YDB:

```
DROP TABLE `fluent-bit/log`
```

Настройка интеграции между Logstash и YDB

В разделе представлены варианты интеграции между системой сбора и анализа данных Logstash и YDB.

Введение

Logstash – инструмент сбора, фильтрации и нормализации логов. Данный продукт позволяет динамически получать, фильтровать и доставлять данные независимо от их формата и сложности. Конфигурация системы производиться с помощью плагинов различных типов - input, output и filter. Набор Logstash плагинов для работы с YDB располагается в репозитории ydb-logstash-plugins:

- Storage Plugin для сохранения данных в строчной или колоночной таблице YDB;
- Input Topic Plugin для чтение данных из YDB топика;
- Output Topic Plugin для отправки данных в YDB топик.

Плагины можно собрать самостоятельно из исходного кода, либо воспользоваться готовыми сборками под две последнии версии Logstash.

> **Примечание**
>
> Дальнейшие инструкции используют placeholder `<path-to-logstash>`, который нужно заменить на путь до установленного в системе Logstash.

Установить любой собранный плагин в Logstash можно с помощью команды

```
<path-to-logstash>/bin/logstash-plugin install </path/to/logstash-plugin.gem>
```

Проверить что плагин установлен, можно командой

```
<path-to-logstash>/bin/logstash-plugin list
```

Которая выведет список всех установленных плагинов, в котором можно найти интересующие.

Конфигурация подключения плагинов к YDB

Все плагины используют одни и те же параметры для настройки подключения к базе данных YDB. Среди этих параметров обязателен только `connection_string`, а все остальные параметры опциональны и позволяют задать режим аутентификации. Если ни один из них не указан, то будет использоваться анонимная аутентификация.

```
# Пример указан для плагина ydb_storage, настройка подключения
# в плагинах ydb_topics_output и ydb_topics_input аналогична
ydb_storage {
    # Строка подключения к БД, содержит схему, адрес, порт и путь БД
    connection_string => "grpc://localhost:2136/local"
    # Значение токена аутентификации (режим Access Token)
    token_auth => "<token_value>"
    # Путь до файла со значением токена аутентификации (режим Access Token)
    token_file => "</path/to/token/file>"
    # Путь до ключа сервисного аккаунта (режим Service Account Key)
    sa_key_file => "</path/to/key.json>"
    # Флаг для использования сервиса метаданных (режим Metadata)
    use_metadata => true
}
```

Плагин YDB Storage

Данный плагин позволяет перенаправить поток данных из Logstash в таблицу YDB для дальнейшего анализа. Для этого удобно использовать колоночные таблицы, оптимизированные для аналитических запросов. Каждое отдельное поле события Logstash будет записано в отдельный столбец таблицы. Те поля, которые не соответствуют ни одной колонке, будут проигнорированы.

Конфигурация плагина

Для настройки плагина необходимо добавить секцию `ydb_storage` в раздел `output` файла конфигурации Logstash. Плагин поддерживает стандартный набор опций для подключения плагинов YDB, плюс несколько специфичных для него опций:

- `table_name` — обязательный параметр с именем таблицы, в которую будут записываться события Logstash
- `uuid_column` — необязательный параметр с именем колонки, в которую плагин будет записывать случайно сгенерированный идентификатор
- `timestamp_column` — необязательный параметр с именем колонку для записи времени создания события Logstash

> **Важно**
>
> Плагин Storage не проверяет события Logstash с точки зрения корректности и уникальности первичных ключей таблицы YDB. Для записи в YDB используется пакетная загрузка данных и при получении различных событий с одинаковыми идентификаторами они будут перезаписывать друг друга. Рекомендуется выбирать в качестве первичные ключей те поля события, которые будут присутствовать в каждом сообщении и однозначно его идентифицировать. Если же такого набора полей нет, то можно добавить в первичный ключ специальную колонку и заполнять ее случайным значением с помощью параметра `uuid_column`

Пример использования

Создание колоночной таблицы для хранения записей

В выбранной базе данных YDB создадим новую колоночную таблицу с нужными нами колонками. В качестве первичного ключа будем использовать случайное значение, генерируемое на стороне плагина:

```
CREATE TABLE `logstash_demo` (
    `uuid`     Text NOT NULL,      -- Уникальный идентификатор
    `ts`       Timestamp NOT NULL, -- Время создания сообщения
    `message`  Text,
    `user`     Text,
    `level`    Int32,

    PRIMARY KEY (
        `uuid`
    )
)
WITH (STORE = COLUMN);
```

Настройка плагина в Logstash

Добавим секцию плагина `ydb_storage` в раздел `output` файла конфигурации Logstash. Дополнительно для проверки работоспособности также добавим плагин `http` в секцию `input`, что позволит создавать события в `Logstash` через http запросы:

```
output {
  ydb_storage {
    connection_string => "..."    # Адрес подключения к  YDB
    table_name => "logstash_demo" # Имя таблицы
    uuid_column => "uuid"         # Колонка в которую плагин будет записывать сгенерированный идентификатор
    timestamp_column => "ts"      # Колонку в которую плагин будет записывать timestamp события
  }
}

input {
  http {
    port => 9876 # Можно указать любой порт
  }
}
```

После изменения данного файла `Logstash` нужно перезапустить.

Отправка тестовых сообщений

Затем можно отправить несколько тестовых сообщений:

```
curl -H "content-type: application/json" -XPUT 'http://127.0.0.1:9876/http/ping' -d '{ "user" : "demo_user",
"message" : "demo message", "level": 4}'
curl -H "content-type: application/json" -XPUT 'http://127.0.0.1:9876/http/ping' -d '{ "user" : "test1", "lev
el": 1}'
curl -H "content-type: application/json" -XPUT 'http://127.0.0.1:9876/http/ping' -d '{ "message" : "error",
"level": -3}'
```

Все команды должны вернуть `ok` в случае успешного выполнения.

Проверка наличия записанных сообщений в YDB

Теперь можно убедиться, что все отправленные сообщения записаны в таблице. Выполним запрос через ydb sql (команда `ydb table query execute -t scan` устарела):

```
ydb sql -s 'SELECT * FROM `logstash_demo`;'
```

и получим список записанных событий:

```
┌─────────┬─────────────────┬──────────────────────────────┬─────────────┬──────────────────────────────────────
──┐
│ level │ message │ ts │ user │ uuid │
├─────────┼─────────────────┼──────────────────────────────┼─────────────┼──────────────────────────────────────
──┤
│ -3 │ "error" │ "2024-05-22T13:16:06.491000Z" │ null │ "74cd4048-0b61-4fb9-9385-308714e21881" │
│ 1 │ null │ "2024-05-22T13:15:56.591000Z" │ "test1" │ "1df27d0a-9aa0-42c7-9ea2-ab69bc1f5d87" │
│ 4 │ "demo message" │ "2024-05-22T13:15:38.760000Z" │ "demo_user" │ "b7468cb1-e1e3-46fa-965d-83e604e80a31" │
└─────────┴─────────────────┴──────────────────────────────┴─────────────┴──────────────────────────────────────
──┘
```

Плагин YDB Topic Input

Данный плагин позволяет читать из YDB топика и преобразовывать их в события `Logstash` для дальнейшей обработки.

Конфигурация плагина

Для настройки плагина мы должны добавить секцию `ydb_topic` в раздел `input` файла конфигурации Logstash. Плагин поддерживает стандартный набор опций для подключения плагинов YDB, плюс несколько специфичных для него опций:

- `topic_path` — обязательный параметр с полным путем топика для чтения;
- `consumer_name` — обязательный параметр с именем читателя топика;

- `schema` — необязательный параметр с вариантам обработки сообщений YDB. По умолчанию плагин читает и отправляет сообщения топика в бинарном виде, но если указать режим `JSON`, то каждое сообщение из топика будет трактоваться как JSON объект.

Пример использования

Создание топика

В выбранной базе данных YDB заранее создадим топик и читателя, из которого будет производиться чтение:

```
ydb -e grpc://localhost:2136 -d /local topic create /local/logstash_demo_topic
ydb -e grpc://localhost:2136 -d /local topic consumer add --consumer logstash-consumer /local/logstash_demo_t
opic
```

Настройка плагина в Logstash

Добавим секцию плагина `ydb_topic` в раздел `input` файла конфигурации Logstash. Дополнительно для проверки работоспособности также добавим плагин `stdout` в секцию `output`, что позволит отслеживать все события в логе выполнения `Logstash`:

```
input {
  ydb_topic {
    connection_string => "grpc://localhost:2136/local" # Адрес подключения к YDB
    topic_path => "/local/logstash_demo_topic"         # Имя топика
    consumer_name => "logstash-consumer"               # Имя читателя
    schema => "JSON"                                   # Используем JSON в качестве формата сообщений топика
  }
}

output {
  stdout {
    codec => rubydebug
  }
}
```

После изменения данного файла `Logstash` нужно перезапустить.

Тестовая запись сообщение в топик

Затем отправим несколько тестовых сообщений:

```
echo '{"message":"test"}' | ydb -e grpc://localhost:2136 -d /local topic write /local/logstash_demo_topic
echo '{"user":123}' | ydb -e grpc://localhost:2136 -d /local topic write /local/logstash_demo_topic
```

Проверка обработки сообщений в Logstash

Благодаря включенному плагину `stdout` в логе Logstash появятся прочитанные из топика сообщения:

```
{
      "message" => "test",
      "@timestamp" => 2024-05-23T10:31:47.712896899Z,
      "@version" => "1"
}
{
      "user" => 123.0,
      "@timestamp" => 2024-05-23T10:34:08.574599108Z,
      "@version" => "1"
}
```

Плагин YDB Topic Output

Данный плагин позволяет записывать события `Logstash` в YDB топик.

Конфигурация плагина

Для настройки плагина мы должны добавить секцию `ydb_topic` в раздел `output` файла конфигурации Logstash. Плагин поддерживает стандартный набор опций для подключения плагинов YDB, плюс дополнительные опции:

- `topic_path` — обязательный параметр с полным путем топика для записи.

Пример использования

Создание топика

В выбранной базе данных YDB заранее создадим топик, в который будет производиться запись:

```
ydb -e grpc://localhost:2136 -d /local topic create /local/logstash_demo_topic
```

Настройка плагина в Logstash

Добавим секцию плагина `ydb_topic` в раздел `output` файла конфигурации Logstash.. Дополнительно для проверки работоспособности также добавим плагин `http` в секцию `input`, что позволит создавать события в `Logstash` через http запросы:

```
output {
  ydb_topic {
    connection_string => "grpc://localhost:2136/local" # Адрес подключения к  YDB
```

```
    topic_path => "/local/logstash_demo_topic"         # Имя топика для записи
  }
}

input {
  http {
    port => 9876 # Можно указать любой порт
  }
}
```

После изменения данного файла `Logstash` нужно перезапустить.

Отправка тестового сообщения

С помощью плагина `http` отправим тестовое сообщение:

```
curl -H "content-type: application/json" -XPUT 'http://127.0.0.1:9876/http/ping' -d '{ "user" : "demo_user",
"message" : "demo message", "level": 4}'
```

Команда вернет `ok` в случае успешного выполнения.

Контрольное чтение сообщения из топика

Для проверки успешности записи в топик создадим нового читателя и прочитаем одно сообщение из топика:

```
ydb -e grpc://localhost:2136 -d /local topic consumer add --consumer logstash-consumer /local/logstash_demo_t
opic
ydb -e grpc://localhost:2136 -d /local topic read /local/logstash_demo_topic --consumer logstash-consumer --c
ommit true
```

Чтение вернет содержимое отправленного сообщения:

```
{"level":4,"message":"demo message","timestamp":1716470292640,"user":"demo_user"}
```

Миграции данных

Источник	Инструкция
Произвольные JDBC-источники данных	Инструкция
MySQL	Инструкция

Импорт структуры таблиц и данных из источников JDBC

Введение

Задача импорта в YDB структуры таблиц и содержащейся в таблицах информации для источников данных, поддерживающих доступ через JDBC-интерфейс, может быть решена с помощью специально разработанной утилиты. Это может быть полезно для перехода с других систем управления базами данных на YDB или, к примеру, для импорта ваших реальных данных для запуска тестов производительности на YDB.

Импорт данных осуществляется в следующем порядке:

1. Утилита импорта подключается к БД-источнику, и определяет состав таблиц и SQL-запросов для импорта.
2. Производится формирование структуры целевых таблиц YDB, которые опционально могут быть сохранены в виде YQL-скрипта.
3. Над БД-получателем производится проверка наличия там целевых таблиц, с созданием недостающих таблиц и с опциональным пересозданием уже существующих таблиц.
4. Осуществляется импорт данных путём чтения из БД-источника с помощью SQL-запроса и вставки в БД-получатель YDB с помощью механизма Bulk Upsert.

Извлечение метаданных из БД-источника, создание таблиц в БД-получателе и импорт данных производится в параллельном режиме, с использованием нескольких конкурентных потоков и одновременно открытых соединений к БД-источнику и БД-получателю. Максимальная степень параллелизма регулируется специальной настройкой, при этом фактическое количество конкурентных операций не может превышать количества импортируемых таблиц.

Установка утилиты импорта

Для работы утилиты импорта требуется JDK 8 или более поздней версии, исполняемый файл `java` должен быть доступен в путях поиска программ. Сборка и тестирование утилиты выполнялись с использованием OpenJDK 8.

Для установки утилиты импорта рекомендуется использовать последнюю версию, доступную на странице релизов. Дистрибутив представляет собой архив в формате ZIP, который необходимо распаковать в локальный каталог сервера, на котором предполагается запускать утилиту.

В состав дистрибутива включены примеры файлов настроек, пример скрипта запуска и набор архивов `lib/*.jar`, содержащих собственно исполняемый код утилиты и её зависимости, включая YDB Java SDK.

Перед запуском утилиты импорта драйверы JDBC для исходных баз данных следует разместить в виде файлов `*.jar` в подкаталоге `lib` каталога инсталляции утилиты.

Порядок использования утилиты импорта

При запуске утилита считывает настройки из файла в формате XML, указанного в качестве аргумента командной строки. Имя файла настроек передается в командной строке при запуске утилиты импорта. В настройках указывается:

- тип БД-источника и параметры подключения к источнику;
- параметры подключения к БД-получателю YDB;
- (опционально) имя файла для сохранения YQL скрипта со структурой таблиц для YDB;
- правила отбора таблиц для импорта из БД-источника;
- перечень явно указанных SQL запросов на выборку данных из БД-источника (может использоваться для импорта результата запроса вместо конкретных таблиц);
- правила формирования имён импортируемых таблиц в БД-получателе;
- степень параллелизма операций (размер пула рабочих потоков, количество соединений с источником и получателем).

Файл настроек необходимо подготовить перед запуском утилиты на основе представленных примеров.

Пример команды запуска утилиты импорта приведён в файле ydb-importer.sh, который можно использовать для запуска, как показано ниже:

```
./ydb-importer.sh my-import-config.xml
```

Особенности и ограничения импорта

На текущий момент протестирована работа утилиты импорта со следующими видами источников данных:

- PostgreSQL,
- MySQL,
- Oracle Database,
- Microsoft SQL Server,
- IBM Db2,
- IBM Informix.

Утилита импорта может оказаться работоспособной для многих не протестированных источников данных JDBC, поскольку в ней используются стандартные вызовы JDBC для доступа к данным и метаданным.

Импорт вторичных индексов не осуществляется.

Некоторые типы данных и виды табличных структур не поддерживаются, включая:

- вложенные таблицы Oracle Database,
- пространственные типы данных Microsoft SQL Server,
- все объектные типы данных Informix.

Особенности импорта таблиц без первичного ключа

В каждой таблице YDB должен быть определён первичный ключ. При наличии у исходной таблицы первичного ключа (или, как минимум, уникального индекса) утилита импорта создаёт первичный ключ для соответствующей таблицы YDB с теми же колонками и в том же порядке, что и у исходной таблицы. При отсутствии первичного ключа и наличии нескольких уникальных индексов утилита выбирает индекс с минимальным количеством используемых колонок.

В случае отсутствия в исходной таблице явно определённого первичного ключа состав его колонок можно определить явным образом в настройках импорта соответствующей таблицы в виде элементов `key-column` раздела `table-ref` (см. далее пример в разделе с описанием формата файла настроек).

Если первичный ключ явно не указан в настройках, и отсутствует в определении исходной таблицы в БД-источнике, то утилита импорта автоматически добавляет в целевую таблицу колонку `ydb_synth_key` и создаёт первичный ключ на её основе. Значения колонки синтетического ключа вычисляются как хеш-код по алгоритму "SHA-256" над всеми значениями всех колонок строки таблицы, за исключением колонок типа BLOB.

При наличии в исходной таблице нескольких строк с полностью одинаковыми значениями, в целевую таблицу для каждого набора дубликатов будет в итоге записана одна строка. Это также означает, что утилита импорта не позволяет корректно загрузить данные из исходной таблицы, все колонки которой имеют тип BLOB.

Особенности импорта больших объектов (BLOB, XML)

Для каждого поля BLOB на источнике создаётся дополнительная таблица YDB следующей структуры:

```
CREATE TABLE `blob_table`(
    `id` Int64,
    `pos` Int32,
    `val` String,
    PRIMARY KEY(`id`, `pos`)
)
```

Формат имени дополнительной таблицы определяется настройкой `table-options` / `blob-name-format`.

Данные из исходного BLOB-поля сохраняются в виде последовательности записей созданной дополнительной таблицы.
Для каждого исходного BLOB-значения генерируется уникальное число, хранимое в поле `id`.
Ссылка на значение сохраняется в основной таблице в поле, имя которого соответствует имени исходного BLOB-поля.
Каждая запись хранит не более 64 Кбайт данных в поле `val`, последовательность блоков данных определяется значением поля `pos`.

Для PostgreSQL работа с данными BLOB в варианте `EXTENSION lo` может требовать явного предоставления специальных полномочий для учётной записи, используемой для подключения к БД-источнику.
Альтернативный вариант - установка режима совместимости на уровне базы данных PostgreSQL:

```
ALTER DATABASE dbname SET lo_compat_privileges TO on;
```

Формат файла настроек

Примеры файлов настроек с комментариями:

- для PostgreSQL;
- для MySQL;
- для Oracle Database;
- для Microsoft SQL Server;
- для IBM Db2;
- для IBM Informix.

Пример и описание формата файла настроек приведены ниже.

```
<?xml version="1.0" encoding="UTF-8"?>
<ydb-importer>
    <workers>
        <!-- Количество рабочих потоков (целое число от 1 и выше).
             Также соответствует максимальному устанавливаемому количеству соединений с источником
             и максимальному количеству сессий с получателем.
         -->
        <pool size="4"/>
    </workers>
    <!-- Параметры подключения к БД-источнику.
         type - обязательный атрибут, влияющий на логику взаимодействия с источником
      -->
    <source type="generic|postgresql|mysql|oracle|mssql|db2|informix">
        <!-- Имя основного класса драйвера JDBC. Типичные значения:
              org.postgresql.Driver
              com.mysql.cj.jdbc.Driver
              org.mariadb.jdbc.Driver
              oracle.jdbc.driver.OracleDriver
              com.microsoft.sqlserver.jdbc.SQLServerDriver
              com.ibm.db2.jcc.DB2Driver
              com.informix.jdbc.IfxDriver
        -->
        <jdbc-class>driver-class-name</jdbc-class>
        <!-- URL JDBC для подключения к источнику. Примеры значений:
              jdbc:postgresql://hostname:5432/dbname
              jdbc:mysql://hostname:3306/dbname
              jdbc:mariadb://hostname:3306/dbname
              jdbc:oracle:thin:@//hostname:1521/serviceName
              jdbc:sqlserver://localhost;encrypt=true;trustServerCertificate=true;database=AdventureWorks202
2;
              jdbc:db2://localhost:50000/SAMPLE
              jdbc:informix-sqli://localhost:9088/stores_demo:INFORMIXSERVER=informix
        -->
        <jdbc-url>jdbc-url</jdbc-url>
        <username>username</username>
        <password>password</password>
    </source>
    <!-- Параметры подключения к БД-получателю. -->
```

```
    <target type="ydb">
        <!-- Выгрузить скрипт создания таблиц для YDB в указанный файл.
             Может использоваться в том числе при отсутствии указания
             connection-string для генерации схемы без фактического создания
             таблиц.  -->
        <script-file>sample-database.yql.tmp</script-file>
        <!-- Строка подключения: protocol + endpoint + database. Примеры значений:
            grpcs://ydb.serverless.yandexcloud.net:2135?database=/ru-central1/b1gfvslmokutuvt2g019/etn63999hr
inbapmef6g
            grpcs://localhost:2135?database=/local
            grpc://localhost:2136?database=/Root/testdb
         -->
        <connection-string>ydb-connection-string</connection-string>
        <!-- Режим аутентификации:
            ENV      использование переменных окружения для настройки аутентификации
            NONE     анонимное подключение к БД - не для продуктивных систем
            STATIC   аутентификация по логину и паролю
            SAKEY    Аутентификация по ключу сервисного аккаунта (управляемый сервис YDB)
            METADATA Аутентификация по метаданным виртуальной машины облака (управляемый сервис YDB)
        -->
        <auth-mode>ENV</auth-mode>
        <!--
            В режиме ENV данные аутентификации необходимо установить в переменных окружения,
            как описано в документации: https://ydb.tech/ru/docs/reference/ydb-sdk/auth#env
            Если аутентификация по ключу сервисного аккаунта (явно либо через указание переменной
            окружения YDB_SERVICE_ACCOUNT_KEY_FILE_CREDENTIALS), то файл ключа надо
            генерировать как написано здесь:
            https://cloud.yandex.ru/docs/iam/operations/authorized-key/create
        -->
        <!-- Логин и пароль для auth-mode: STATIC -->
        <static-login>username</static-login>
        <static-password>password</static-password>
        <!-- Удалять ли уже существующие таблицы с теми же именами, что предполагается заливать -->
        <replace-existing>false</replace-existing>
        <!-- Заливать ли данные в таблицы после их создания или пересоздания -->
        <load-data>true</load-data>
        <!-- Максимальная порция заливки обычных данных, в строках
             (используется при записи данных в основную таблицу) -->
        <max-batch-rows>1000</max-batch-rows>
        <!-- Максимальная порция заливки BLOB-данных, в строках
             (используется при записи данных в дополнительные таблицы для BLOB-полей) -->
        <max-blob-rows>200</max-blob-rows>
    </target>
    <!-- Настройки преобразования структуры исходных таблиц.
         Устанавливаются централизованно с присвоением имени,
         затем используются для конкретной группы отбираемых таблиц.   -->
    <table-options name="default">
        <!--  Возможные значения case-mode: ASIS (по умолчанию), LOWER, UPPER.
              Влияет на регистр подставляемых в шаблоны имён.  -->
        <case-mode>ASIS</case-mode>
        <!-- Шаблон полного имени таблицы, включая каталог размещения.
             Используются подстановочные имена ${schema} и ${table},
             соответствующие схеме и имени копируемой исходной таблицы. -->
        <table-name-format>oraimp1/${schema}/${table}</table-name-format>
        <!-- Шаблон полного имени таблицы, включая каталог размещения.
             Используются подстановочные имена ${schema}, ${table} и ${field},
             соответствующие схеме, имени копируемой исходной таблицы и имени
             поля типа BLOB. -->
        <blob-name-format>oraimp1/${schema}/${table}_${field}</blob-name-format>
        <!-- Возможные значения: DATE (по умолчанию), INT, STR.
             Вариант DATE не позволяет сохранить даты ранее 1 января 1970 года,
                 при попытке импорта неподдерживаемых значений будут возникать ошибки
             Вариант INT хранит дату как 32-битное целое в формате ГГГГММДД,
                 а для меток времени сохраняет количество миллисекунд как 64-битное целое
             Вариант STR хранит дату как строку в формате ГГГГ-ММ-ДД,
                 а для меток времени в формате "ГГГГ-ММ-ДД чч:мм:сс.xxx"
         -->
        <conv-date>INT</conv-date>
        <conv-timestamp>STR</conv-timestamp>
        <!-- Если указано значение true, колонки таблиц неподдерживаемых типов пропускаются
             с выводов в лог соответствующего предупреждения. В противном случае генерируется
             ошибка импорта, и соответствующая таблица пропускается целиком. -->
        <skip-unknown-types>true</skip-unknown-types>
    </table-options>
    <!-- Фильтр для отбора копируемых таблиц с источника -->
    <table-map options="default">
        <!-- Включаемые схемы -->
        <include-schemas regexp="true">.*</include-schemas>
        <!-- Исключаемые схемы -->
        <exclude-schemas>SOMESCHEMA</exclude-schemas>
        <!-- Также могут присутствовать include-tables и exclude-tables,
             для явного указания фильтра по именам таблиц
             и/или регулярным выражениям над именами таблиц. -->
    </table-map>
    <!-- Конкретный запрос или указание ссылки на конкретную таблицу -->
    <table-ref options="default">
        <schema-name>ora$sys</schema-name>
        <table-name>all_tables</table-name>
        <!-- Если указан запрос, он выполняется как написано -->
        <query-text>SELECT * FROM all_tables</query-text>
        <!-- Для запроса желательно явно определить ключевые колонки.  -->
```

```
        <key-column>OWNER</key-column>
        <key-column>TABLE_NAME</key-column>
    </table-ref>
</ydb-importer>
```

Импорт структуры таблиц и данных из MySQL

Введение

Перенос схемы и данных из **MySQL** в YDB — в том числе **больших таблиц**, объём которых превышает доступную оперативную память, — выполняется утилитой mysql2ydb.

Цель утилиты — получить копию исходной базы данных один к одному. Имена таблиц и колонок сохраняются как в MySQL; типы данных сопоставляются с ближайшими эквивалентами YDB (`INT` → `Int32`, `VARCHAR` → `Text`, `AUTO_INCREMENT` → `BigSerial` и т. д.). Поскольку имена переносятся без изменений, они должны быть допустимыми идентификаторами в YDB (символы, длина, зарезервированные слова). В результате должна получиться база YDB, повторяющая исходную структуру и позволяющая выполнять запросы с минимальным «переводом» с MySQL.

Каждый фрагмент данных читается из MySQL одним ограниченным запросом `SELECT`, записывается в YDB порцией (`BulkUpsert` или транзакционный `UPSERT`), после чего освобождается из памяти. Процесс повторяется, пока не будет обработана вся таблица — в памяти одновременно хранится не больше одной порции (при конвейерном чтении — не больше двух).

Импорт выполняется в следующем порядке:

1. Утилита подключается к MySQL и по метаданным `information_schema` определяет состав таблиц и их структуру.
2. В YDB создаются целевые таблицы с сопоставленными типами данных.
3. Данные переносятся порциями: чтение из MySQL и запись в YDB.
4. После успешной записи каждой порции в служебную таблицу состояния сохраняется позиция курсора — при сбое миграцию можно продолжить с последнего успешного фрагмента.

Соответствие схемы MySQL

При переносе схемы mysql2ydb стремится к **структурному соответствию**: те же имена таблиц и колонок, индексы там, где YDB их поддерживает, типы, при которых значения сохраняются без потерь (беззнаковые целые остаются беззнаковыми, логические типы — логическими и т. д.).

Утилита читает метаданные MySQL из `information_schema` и формирует DDL YDB напрямую. Основные сопоставления типов:

Тип MySQL	Тип в YDB
`INT`, `MEDIUMINT`, `SMALLINT`, `TINYINT`	`Int32`
`BIGINT`	`Int64`
`INT UNSIGNED`, `MEDIUMINT UNSIGNED`, `SMALLINT UNSIGNED`, `TINYINT UNSIGNED`	`Uint32`
`BIGINT UNSIGNED`	`Uint64`
`TINYINT(1)`	`Bool`
`FLOAT`	`Float`
`DOUBLE`, `REAL`	`Double`
`BIT`	`Uint64` (в том числе `BIT(1)` — оборачивается в `Uint64`, не в `Bool`)
`CHAR`, `VARCHAR`, `TEXT`, `MEDIUMTEXT`, `LONGTEXT`	`Text`
`ENUM`, `SET`, `JSON`	`Text`
`BINARY`, `VARBINARY`, `BLOB`, `MEDIUMBLOB`, `LONGBLOB`	`String` (байты inline в той же колонке)
`DATE`	`Date`
`DATETIME`, `TIMESTAMP`	`Timestamp` — значение сохраняется как UTC. У `DATETIME` (тип без часового пояса) трактовка зависит от `time_zone` MySQL во время чтения; учитывайте это при миграции исторических данных.
`DECIMAL(p, s)`, `NUMERIC(p, s)`	`Decimal(22, 9)` для всех — исходные `p` / `s` не сохраняются; значения, не помещающиеся в 22 цифры, будут потеряны
`YEAR`	`Uint16`
Прочие типы	`Text` (запасной вариант)

Структурные решения

Элемент схемы MySQL	Решение mysql2ydb
Имена таблиц и колонок	Как в MySQL (`users`, `orders`, …); должны быть допустимыми идентификаторами YDB
`AUTO_INCREMENT`	`BigSerial` + `ALTER SEQUENCE … START WITH` из `TABLES.AUTO_INCREMENT`
Вторичные `KEY` / `UNIQUE KEY`	`INDEX … GLOBAL ASYNC` / `GLOBAL UNIQUE SYNC` в `CREATE TABLE`
Таблица без `PRIMARY KEY`	Не поддерживается при автоматическом создании схемы — см. Таблицы без первичного ключа
Партиционирование	Только `AUTO_PARTITIONING_BY_LOAD`

Вторичные индексы

Неуникальные ключи MySQL становятся `INDEX … GLOBAL ASYNC`, уникальные (кроме первичного) — `INDEX … GLOBAL UNIQUE SYNC`. Таблицы с синхронными уникальными индексами при загрузке данных автоматически переключаются на транзакционный `UPSERT`, поскольку `BulkUpsert` поддерживает только асинхронные индексы.

Примеры

`AUTO_INCREMENT` — колонка MySQL `id BIGINT AUTO_INCREMENT` превращается в:

```
`id` BigSerial NOT NULL,
PRIMARY KEY (`id`)
```

После `CREATE TABLE`, если известно значение `information_schema.TABLES.AUTO_INCREMENT`:

```
ALTER SEQUENCE `<db>/<table>/_serial_column_id` START WITH <next_value> RESTART
```

Таблицы без первичного ключа

В YDB у каждой таблицы должен быть `PRIMARY KEY`. mysql2ydb берёт ключ **только** из колонок MySQL с `COLUMN_KEY = 'PRI'`; индексы `UNIQUE KEY` становятся вторичными `GLOBAL UNIQUE SYNC`, но **не** подставляются вместо первичного ключа. Если у таблицы нет `PRIMARY KEY`, создание схемы завершается ошибкой (`PRIMARY KEY ()` — недопустимый DDL), и перенос данных не начинается.

Перед миграцией для таких таблиц нужно задать ключ в MySQL, например:

```
ALTER TABLE my_table ADD COLUMN id BIGINT AUTO_INCREMENT PRIMARY KEY FIRST;
```

или назначить `PRIMARY KEY` на существующую колонку (или набор колонок), если он действительно уникален.

Если добавить `PRIMARY KEY` в MySQL нельзя, но схему в YDB можно подготовить вручную, создайте таблицу с нужным ключом сами и запустите утилиту с флагом `-data-only`. Чтение из MySQL при отсутствии `PRIMARY KEY` в метаданных пойдёт через `LIMIT/OFFSET`: на каждой странице MySQL заново сканирует все предыдущие строки, поэтому время миграции растёт квадратично от размера таблицы — для больших таблиц лучше предварительно добавить ключ в MySQL. Ключевые колонки в YDB должны присутствовать среди колонок MySQL.

Если нужен автоматический импорт без ключа в MySQL, рассмотрите YDB Importer — он добавляет синтетическую колонку `ydb_synth_key` (SHA-256 по строке). mysql2ydb синтетические ключи не создаёт.

Эффективный доступ к MySQL

Утилита минимизирует обращения к MySQL и ограничивает объём каждого запроса:

1. **Постраничное чтение по первичному ключу** — `WHERE (pk…) > (?) ORDER BY pk LIMIT n`. Без полного сканирования таблицы в память.
2. **Конвейер чтение → запись** — одна горутина читает следующую порцию из MySQL, пока другая записывает предыдущую в YDB. На таблицу одновременно в полёте не более двух порций.
3. **Адаптивный размер порции** — число строк ограничивается доступной RAM и средним размером строки из `information_schema`.
4. **Контрольная точка прогресса** — после каждой успешно записанной порции позиция курсора и счётчик строк сохраняются в служебной таблице YDB; перезапуск продолжает с последней порции.

```
MySQL: SELECT порция N   →  BulkUpsert(порция N)   →  YDB
MySQL: SELECT порция N+1 →  BulkUpsert(порция N+1) →  YDB
...
```

Возможности

- **Копия схемы один к одному** — исходные имена таблиц и колонок; сопоставление типов MySQL с ближайшими типами YDB (`AUTO_INCREMENT` → `BigSerial`, `UNSIGNED` → `Uint32` / `Uint64`, вторичные индексы, `TINYINT(1)` → `Bool`; см. раздел Соответствие схемы MySQL).
- **Создание схемы** — таблицы в YDB создаются по метаданным MySQL (`information_schema`).
- **Пофрагментная миграция данных** — таблица обрабатывается порциями фиксированного размера; объём данных может превышать доступную RAM.
- **Перезапись по ключу** — данные записываются через `BulkUpsert` или транзакционный `UPSERT`; повторный запуск не создаёт дубликаты строк.
- **Возобновление** — прогресс по каждой таблице хранится в YDB; прерванную миграцию можно продолжить с последней порции.

Установка

Исходный код и инструкции по сборке — в репозитории mysql-ydb-importer на GitHub.

Для сборки требуется Go версии не ниже указанной в `go.mod` репозитория:

```
git clone https://github.com/ydb-platform/mysql-ydb-importer.git
cd mysql-ydb-importer
go build -o mysql2ydb ./cmd/mysql2ydb
```

Порядок использования

Параметры подключения к MySQL по умолчанию читаются из файла **~/.my.cnf** (как у клиента `mysql`), секция `[client]`. Флаг `-mysql` переопределяет конфигурационный файл.

> **Таблицы без первичного ключа**
>
> Таблицы без `PRIMARY KEY` в MySQL **не поддерживаются** при автоматическом создании схемы. Перед запуском добавьте первичный ключ в MySQL или подготовьте схему вручную и используйте `-data-only` (см. раздел Таблицы без первичного ключа).

Пример `~/.my.cnf`:

```
[client]
user = myuser
password = mypass
host = localhost
port = 3306
database = mydb
```

Если сервер требует защищённое соединение (`require_secure_transport=ON`), добавьте в `[client]`:

```
ssl-mode = REQUIRED
```

Для самоподписанного сертификата (без проверки):

```
ssl-mode = REQUIRED
ssl-verify = 0
```

Альтернативный вариант — `ssl=1` и `ssl-verify=0`.

Минимальный запуск при наличии `~/.my.cnf` (обязателен только `-ydb`):

```
./mysql2ydb -ydb "grpc://localhost:2136"
```

С явным DSN MySQL (игнорирует `~/.my.cnf`):

```
./mysql2ydb -mysql "user:$MYSQL_PASSWORD@tcp(localhost:3306)/mydb" -ydb "grpc://localhost:2136" -batch-size 1
0000
```

Флаги командной строки

Флаг	Описание
`-mysql`	DSN MySQL (если задан — переопределяет `~/.my.cnf`)
`-ydb`	Конечная точка YDB (обязательный)
`-ydb-database`	Путь к базе данных YDB (по умолчанию `local`)
`-schema-only`	Только создать схему, без переноса данных
`-data-only`	Только перенести данные (схема в YDB должна уже существовать — в том числе созданная вручную)
`-batch-size`	Целевой размер порции в строках (по умолчанию 10 000)
`-max-chunk-rows`	Жёсткий верхний предел строк в порции (по умолчанию 25 000)
`-parallel-tables`	Число таблиц для параллельного переноса (по умолчанию 1)
`-tables`	Список таблиц через запятую (по умолчанию — все)
`-force`	Перенести все таблицы заново, игнорируя сохранённое состояние
`-force-recreate`	Удалить все таблицы в YDB и пересоздать схему с нуля

Примеры

Только схема:

```
./mysql2ydb -mysql "..." -ydb "grpc://localhost:2136" -schema-only
```

Только данные (после создания схемы):

```
./mysql2ydb -mysql "..." -ydb "grpc://localhost:2136" -data-only -batch-size 5000
```

Выбранные таблицы:

```
./mysql2ydb -mysql "..." -ydb "grpc://localhost:2136" -tables "users,orders"
```

Пофрагментное чтение

- Для таблиц с **первичным ключом** в MySQL используется постраничное чтение по курсору: `WHERE (pk > ?) ORDER BY pk LIMIT batch_size`. В памяти одновременно находится не больше одной порции.

- Если в MySQL нет `PRIMARY KEY`, но схема в YDB уже создана вручную (режим `-data-only`), применяется `LIMIT batch_size OFFSET offset` — см. раздел Таблицы без первичного ключа.

Запись в YDB

По умолчанию данные записываются через `BulkUpsert`. Для таблиц с синхронными уникальными индексами утилита автоматически переключается на транзакционный `UPSERT`, поскольку `BulkUpsert` поддерживает только асинхронные индексы.

Запросы к YDB помечены как идемпотентные, поэтому SDK автоматически повторяет их при сетевых сбоях. Отсутствие дубликатов при перезапуске миграции обеспечивается перезаписью строк по ключу и сохранением прогресса в служебной таблице (см. раздел Возможности, пункт «Возобновление»).

Сравнение с YDB Importer

Для загрузки данных в YDB есть и другие инструменты. YDB Importer — универсальный JDBC-импортёр для PostgreSQL, Oracle, SQL Server и других источников: XML-конфигурация, параллельный импорт, настройка партиционирования.

mysql2ydb решает более узкую задачу — перенос базы MySQL «один к одному» одним бинарным файлом, без JVM. Для миграции только из MySQL с большими таблицами достаточно `~/.my.cnf` и флага `-ydb`; прогресс сохраняется в служебной таблице YDB.

	YDB Importer	**mysql2ydb**
Среда выполнения	Java + JDBC-драйверы + XML-конфигурация	Один бинарный файл на Go, нативный протокол MySQL
Область применения	Множество JDBC-источников	Только MySQL
Размер порции по умолчанию	1 000 строк (`max-batch-rows`)	10 000 строк (`-batch-size`), с автонастройкой
Большие таблицы	Параллельные диапазоны, буферы партиций	Постраничное чтение по курсору, размер порции с учётом RAM
Возобновление после сбоя	Повторный запуск импорта	Контрольная точка на каждую порцию в служебной таблице YDB
Настройка	JAR-файлы JDBC, XML-конфигурация	`~/.my.cnf` и два флага

YDB Importer рассчитан на импорт из многих JDBC-источников в **настраиваемую** структуру YDB (`table-name-format`, `blob-name-format`, режимы преобразования дат, опциональные синтетические ключи). Ниже — практические отличия при импорте одной и той же базы MySQL двумя инструментами:

Возможность MySQL	mysql2ydb (копия один к одному)	ydb-importer
Имена таблиц	Как в MySQL (`users`, `orders`, …)	Переименование по шаблону, например `mysql1/${schema}/${table}` → `mysql1/mydb/users` (пример конфигурации)
Имена колонок	Как в MySQL	Санитизация: пробелы, `.`, `/`, `` ` `` → `_`
`AUTO_INCREMENT`	`BigSerial` + `ALTER SEQUENCE … START WITH` из `TABLES.AUTO_INCREMENT`	Обычный `Int32` / `Int64`; без `BigSerial` и сброса последовательности
`INT UNSIGNED`, `BIGINT UNSIGNED`, …	`Uint32` / `Uint64`	JDBC сопоставляет целые со знаковыми `Int32` / `Int64`
`TINYINT(1)`	`Bool`	`TINYINT` → `Int32` (`BOOLEAN` → `Bool`)
`BIT`	`Uint64`	`Bool` (JDBC `Types.BIT`)
Вторичные `KEY` / `UNIQUE KEY`	`INDEX … GLOBAL ASYNC` / `GLOBAL UNIQUE SYNC` в `CREATE TABLE`	В DDL только `PRIMARY KEY`; вторичные индексы не создаются
`ENUM`, `SET`, `JSON`	`Text`	`Text` (аналогично)
`BLOB` / `BINARY`	Та же колонка, значение inline в `String` (байты)	Колонка становится `Int64` (идентификатор blob); данные выносятся в **отдельную таблицу** `${schema}/${table}_${field}` со строками `(id, pos, val)` блоками по 64 КБ
`TEXT` / `CLOB` (большой текст)	Inline `Text` в основной таблице	Опционально отдельная CLOB-таблица (блоки по 32 К символов) или inline `Text` в зависимости от настроек
Таблица **без первичного ключа**	**Не поддерживается** — `CREATE TABLE` завершается ошибкой; добавьте `PRIMARY KEY` в MySQL перед миграцией	Добавляется колонка `ydb_synth_key Text` как PK (SHA-256 по строке); **дубликаты схлопываются** в одну строку
`DATE`	`Date`	`Date32` по умолчанию (`conv-date=DATE_NEW`)
`DATETIME` / `TIMESTAMP`	`Timestamp`	`Datetime64` / `Timestamp64` по умолчанию
`TIME`	`Text` (запасной вариант)	`Int32` (секунды с полуночи)
`DECIMAL(p,s)`	`Decimal(22, 9)` для всех	`Decimal(p,s)` при `allow-custom-decimal=true`
`YEAR`	`Uint16`	Не покрыто в тестах типов MySQL

Партиционирование	Только `AUTO_PARTITIONING_BY_LOAD`	`PARTITION_AT_KEYS`, разбиение по партициям источника, `HASH` для column store и др.

Работа с BLOB

Таблица MySQL `attachments (id INT, data MEDIUMBLOB)`:

	Колонка `data` в основной таблице	Где лежат байты
mysql2ydb	`data String`	В той же строке
ydb-importer	`data Int64` (ссылка)	Отдельная таблица `…/attachments_data` с порционными строками

С mysql2ydb можно выполнить `SELECT data FROM attachments WHERE id = 1` так же, как в MySQL. С ydb-importer потребуется соединение основной таблицы с дополнительной blob-таблицей.

Миграции схемы

Среда	Инструкция
goose	Инструкция
Liquibase	Инструкция
Flyway	Инструкция
dbt	Инструкция

Интеграция dbt с YDB

Введение

dbt (data build tool) - инструмент для трансформации данных в аналитических хранилищах, предназначенный для организации процессов разработки моделей данных по аналогии с разработкой программного обеспечения. Он охватывает этап трансформации (T) в ETL-пайплайнах и позволяет структурировать логику преобразований - от сырых данных до витрин - с поддержкой тестирования, документирования и data lineage - сквозной прослеживаемости данных от источника до конечной модели.

Модели в dbt определяются единообразно - как `SELECT`-выражения, что упрощает их сопровождение и композицию. Инструмент обеспечивает их материализацию в хранилище данных, а также отслеживает зависимости между моделями, в том числе в процессе эволюции одних и тех же данных через стадии преобразований.

Для интеграции dbt с YDB используется коннектор dbt-ydb. Он обеспечивает подключение к YDB в качестве целевой платформы и поддерживает материализацию моделей в виде таблиц и представлений, работу с инкрементальными данными, загрузку тестовых датасетов (seeds), а также выполнение тестов и генерацию документации.

В этом разделе описаны возможности коннектора, а также шаги по его первоначальной настройке и запуску.

> **Важно**
>
> Коннектор dbt-ydb находится в стадии Preview и, в настоящий момент, не обеспечивает поддержку всех возможностей dbt. Ниже приводится список возможностей и ограничений.

Возможности

Модели и их материализация

Ключевое понятие dbt — модель данных. По своей сути, это SQL-выражение, в котором могут быть использованы все источники данных внутри вашего хранилища, включая другие модели. Существуют разные подходы к физическому созданию модели (ее материализации) внутри YDB, которые поддерживает коннектор dbt-ydb:

1. Представление (view) — сохраняется как представление YDB.
2. Таблица (table) — сохраняется как таблица внутри YDB и пересоздаётся при каждом обновлении модели силами dbt.

   dbt-ydb коннектор поддерживает возможность определять следующие параметры таблицы через конфигурацию модели:

Параметр	Обязательность	Значение по умолчанию	Описание
primary_key	Да		Первичный ключ таблицы
store_type	Нет	row	Тип таблицы. 'row' - для строковой таблицы и 'column' для колоночной таблицы
auto_partitioning_by_size	Нет		Автоматическое партиционирование по размеру
auto_partitioning_partition_size_mb	Нет		Порог размера партиции
ttl	Нет		Правило Time-To-Live

   Пример модели, материализуемой как таблица на основе другой модели (связаны через ref).
   Заданы первичный ключ, TTL и автоматическое партиционирование по размеру.

   ```
   {{ config(
      primary_key='id, created_at',
      store_type='row',
      auto_partitioning_by_size='ENABLED',
      auto_partitioning_partition_size_mb=256,
      ttl='Interval("P30D") on created_at'
   ) }}

   select
      id,
      name,
      created_at
   from {{ ref('source_table') }}
   ```

3. Инкрементальное представление (incremental model) — создаётся как таблица внутри YDB, но при обновлении не пересоздаётся, а обновляется изменившимися и новыми строками.

   dbt-ydb коннектор поддерживает те же параметры, что определены для табличной материализации, а также уникальные параметры инкрементального представления:

Параметр	Обязательность	Значение по умолчанию	Описание
incremental_strategy	Нет	MERGE	Стратегия инкрементальной материлазации. Поддерживается стратегия MERGE, использующая YDB операцию UPSERT. Поддержка стратегии APPEND находится в разработке

> **Примечание**
>
> Еще один тип материализации, эфемерное представление, в данный момент не поддерживается коннектором.

Снепшоты

Механизм снепшотов в данный момент не поддерживается dbt-ydb.

Тестовые/справочные данные, создаваемые из файлов CSV (сиды)

dbt-ydb коннектор поддерживает возможность dbt определять сиды для загрузки справочных и тестовых данных из CSV-файлов в ваш проект и их использование в других моделях.

Тестирование данных

dbt-ydb поддерживает стандартные тесты данных dbt, а также специфичные тесты в рамках возможностей YQL.

Генерация документации

dbt-ydb поддерживает создание документации из проектов dbt для YDB.

Подготовка к использованию

Зависимости

Для начала работы с dbt на YDB вам понадобятся:

- Python 3.10+;
- dbt Core (1.8+);
- Существующий кластер YDB, однонодовая инсталляция из быстрого старта будет достаточной.

> **Примечание**
>
> Внимание! dbt Fusion (2.0) в данный момент не поддерживается.

Установка

Для установки dbt-ydb выполните

```
pip install dbt-ydb
```

Подключение dbt к YDB кластеру

dbt подключается к YDB через dbt-ydb коннектор стандартным для YDB образом. Для успешного подключения требуется указать эндпоинт, путь к базе данных, а также параметры аутентификации в файле профилей dbt.

Пример файла профилей с возможными вариантами аутентификации, а также значениями по умолчанию (в квадратных скобках)

```
profile_name:
target: dev
outputs:
   dev:
      type: ydb
      host: [localhost] # YDB host
      port: [2136] # YDB port
      database: [/local] # YDB database
      schema: [<empty string>] # Optional subfolder for DBT models
      secure: [False] # If enabled, grpcs protocol will be used
      root_certificates_path: [<empty string>] # Optional path to root certificates file

      # Static Credentials
      username: [<empty string>]
      password: [<empty string>]

      # Access Token Credentials
      token: [<empty string>]

      # Service Account Credentials
      service_account_credentials_file: [<empty string>]
```

Создание проекта "с нуля" через команду dbt init

1. Инициализация проекта

   ```
   dbt init
   ```

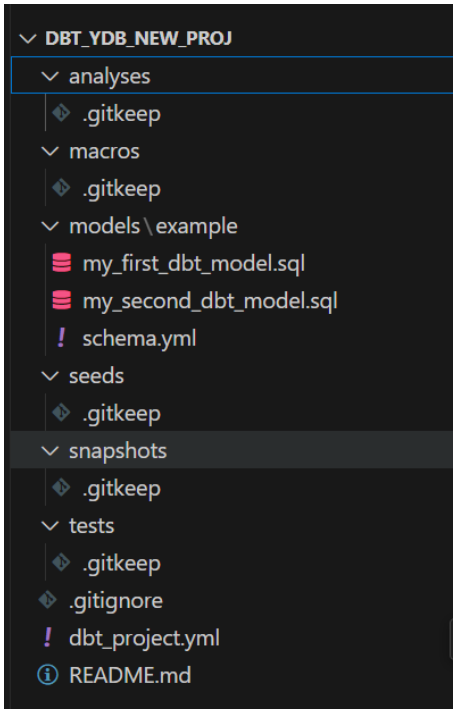
2. Следуйте интерактивным подсказкам dbt для выбора коннектора dbt-ydb и настроек аутентификации, подходящих для вашего кластера YDB
3. В результате директория с вашим проектом, а также файл профилей dbt в домашней директории пользователя будет создан или обновлен новым соединением к YDB

   ```
   ~/.dbt/profiles.yml
   ```

4. Запустите команду dbt debug для проверки соединения

   ```
   dbt debug
   ```

5. Внутри директории вашего проекта, вы увидите следующую структуру



6. Адаптация модели my_first_dbt_model

   В данный момент dbt не поддерживает возможности модификации автоматически генерируемого примера индивидуально под коннектор, поэтому, для запуска данной модели через dbt-ydb требуется ее обновить следующим образом:

```
/*
   Welcome to your first dbt model!
   Did you know that you can also configure models directly within SQL files?
   This will override configurations stated in dbt_project.yml

   Try changing "table" to "view" below
*/

{{ config(materialized='table', primary_key='id') }}

select *
from (
   select 1 as id
   union all
   select null as id
)
```

7. Теперь вы можете запустить ваш проект

```
dbt run
```

Запуск тестового примера

Вместе с коннектором dbt-ydb поставляется пример, который можно использовать для быстрой проверки возможностей dbt для YDB

1. Клонирование репозитория

```
git clone https://github.com/ydb-platform/dbt-ydb.git
cd dbt-ydb/examples/jaffle_shop
```

2. Настройка файла профиля подключения к вашей YDB в файле `profiles.yml`. Для однонодовой инсталляции из быстрого старта файл будет выглядеть следующим образом

```
profile_name:
   target: dev
   outputs:
  dev:
     type: ydb
     host: localhost # Хост YDB
     port: 2136 # Порт YDB
     database: /local # База данных YDB
     schema: jaffle_shop
```

3. Проверьте подключение

```
dbt debug
```

4. Подготовьте тестовые данные через seeds

Эта команда загрузит CSV-файлы из директории `data/` в таблицы `raw_*` YDB.

```
dbt seed
```

5. Запустите выполнение моделей

   Будут созданы таблицы и представления на основе примеров моделей проекта.

   ```
   dbt run
   ```

6. Протестируйте данные в моделях

   Данная команда выполнит стандартные проверки данных, описанные в тестовом примере — проверки на `null`, на допустимые значения списка и другие.

   ```
   dbt test
   ```

7. Сгенерируйте документацию и запустите локальный веб-сервер для её просмотра

   Документация по проекту будет доступна в браузере: http://localhost:8080.

   ```
   dbt docs generate
   dbt docs serve --port 8080
   ```

Дальнейшие шаги

Официальную документацию dbt можно найти по ссылке.
Дополнительно вы можете изучить исходный код коннектора, а также поучаствовать в его развитии в публичном репозитории dbt-ydb на GitHub.

Миграции схемы данных YDB с помощью Flyway

Введение

Flyway - это инструмент для миграции баз данных с открытым исходным кодом. Она имеет расширения для различных систем управления базами данных (СУБД), включая YDB.

Установка

Чтобы использовать Flyway вместе с YDB в Java / Kotlin приложении или в Gradle / Maven плагине, требуется зависимости Flyway core, расширения Flyway для YDB и YDB JDBC Driver:

Maven

```
<!-- Set actual versions -->
<dependency>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-core</artifactId>
    <version>${flyway.core.version}</version>
</dependency>

<dependency>
    <groupId>tech.ydb.jdbc</groupId>
    <artifactId>ydb-jdbc-driver</artifactId>
    <version>${ydb.jdbc.version}</version>
</dependency>

<dependency>
    <groupId>tech.ydb.dialects</groupId>
    <artifactId>flyway-ydb-dialect</artifactId>
    <version>${flyway.ydb.dialect.version}</version>
</dependency>
```

Gradle

```
dependencies {
    // Set actual versions
    implementation "org.flywaydb:flyway-core:$flywayCoreVersion"
    implementation "tech.ydb.dialects:flyway-ydb-dialect:$flywayYdbDialecVersion"
    implementation "tech.ydb.jdbc:ydb-jdbc-driver:$ydbJdbcVersion"
}
```

Для работы с YDB через Flyway CLI требуется установить `flyway` утилиту любым из рекомендованных способов.

Затем утилита должна быть расширена с помощью диалекта YDB и драйвера JDBC:

```
# install flyway
# cd $(which flyway) // prepare this command for your environment

cd libexec
# set actual versions of .jar files

cd drivers && curl -L -o ydb-jdbc-driver-shaded-2.1.0.jar https://repo.maven.apache.org/maven2/tech/ydb/jdbc/
ydb-jdbc-driver-shaded/2.1.0/ydb-jdbc-driver-shaded-2.1.0.jar

cd ..

cd lib && curl -L -o flyway-ydb-dialect.jar https://repo.maven.apache.org/maven2/tech/ydb/dialects/flyway-ydb
-dialect/1.0.0-RC0/flyway-ydb-dialect-1.0.0-RC0.jar
```

Примечание

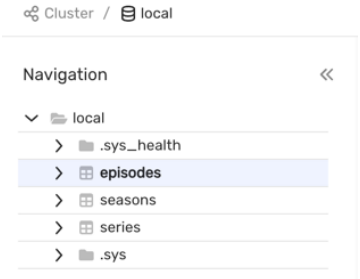
Flyway Desktop в настоящее время не поддерживается.

Управление миграциями с помощью Flyway

baseline

Команда baseline инициализируют Flyway в существующей базе данных, исключающая все миграции вплоть до `baselineVersion` включительно.

Предположим, что мы имеем существующий проект с текущей схемой базы данных:



Запишем наши существующие миграции следующим образом:

```
db/migration:
  V1__create_series.sql
  V2__create_seasons.sql
  V3__create_episodes.sql
```

Содержимое `SQL` файлов:

V1__create_series.sql

```
  CREATE TABLE series -- series is the table name.
  (                          -- Must be unique within the folder.
      series_id    Int64,
      title        Text,
      series_info  Text,
      release_date Int64,
      PRIMARY KEY (series_id) -- The primary key is a column or
      -- combination of columns that uniquely identifies
      -- each table row (contains only
      -- non-repeating values). A table can have
      -- only one primary key. For every table
      -- in YDB, the primary key is required.
  );
```

V2__create_seasons.sql

```
  CREATE TABLE seasons
  (
      series_id Uint64,
      season_id Uint64,
      title Utf8,
      first_aired Uint64,
      last_aired Uint64,
      PRIMARY KEY (series_id, season_id)
  );
```

V3__create_episodes.sql

```
CREATE TABLE episodes
(
    series_id Uint64,
    season_id Uint64,
    episode_id Uint64,
    title Utf8,
    air_date Uint64,
    PRIMARY KEY (series_id, season_id, episode_id)
);
```

Установим `baselineVersion = 3`, затем выполним следующую команду:

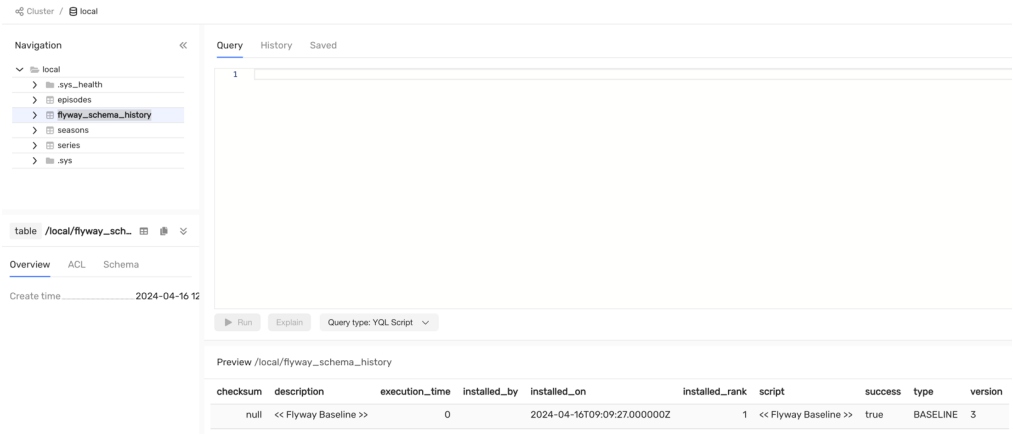
```
flyway -url=jdbc:ydb:grpc://localhost:2136/local -locations=db/migration -baselineVersion=3 baseline
```

Примечание

Все примеры использует докер контейнер, для которого не требуется дополнительных параметров для аутентификации.

Как подключится к YDB можно посмотреть в следующем разделе.

В результате будет создана таблица с именем `flyway_schema_history`, которая будет содержать запись `baseline`:



migrate

Команда migrate эволюционирует схему базы данных до последней версии. Если таблица истории схемы не была создана, то Flyway создаст ее автоматически.

Добавим к предыдущему примеру миграцию загрузки данных:

```
db/migration:
  V1__create_series.sql
  V2__create_seasons.sql
  V3__create_episodes.sql
  V4__load_data.sql
```

Содержимое файла `V4__load_data.sql`

```
INSERT INTO series (series_id, title, release_date, series_info)
VALUES
    -- By default, numeric literals have type Int32
    -- if the value is within the range.
    -- Otherwise, they automatically expand to Int64.
    (1,
     "IT Crowd",
     CAST(Date ("2006-02-03") AS Uint64), -- CAST converts one datatype into another.
        -- You can convert a string
        -- literal into a primitive literal.
        -- The Date() function converts a string
        -- literal in ISO 8601 format into a date.

     "The IT Crowd is a British sitcom produced by Channel 4, written by Graham Linehan, produced by Ash
Atalla and starring Chris O'Dowd, Richard Ayoade, Katherine Parkinson, and Matt Berry."),
    (2,
     "Silicon Valley",
     CAST(Date ("2014-04-06") AS Uint64),
     "Silicon Valley is an American comedy television series created by Mike Judge, John Altschuler and D
ave Krinsky. The series focuses on five young men who founded a startup company in Silicon Valley.")
;

INSERT INTO seasons (series_id, season_id, title, first_aired, last_aired)
VALUES (1, 1, "Season 1", CAST(Date ("2006-02-03") AS Uint64), CAST(Date ("2006-03-03") AS Uint64)),
       (1, 2, "Season 2", CAST(Date ("2007-08-24") AS Uint64), CAST(Date ("2007-09-28") AS Uint64)),
       (1, 3, "Season 3", CAST(Date ("2008-11-21") AS Uint64), CAST(Date ("2008-12-26") AS Uint64)),
       (1, 4, "Season 4", CAST(Date ("2010-06-25") AS Uint64), CAST(Date ("2010-07-30") AS Uint64)),
       (2, 1, "Season 1", CAST(Date ("2014-04-06") AS Uint64), CAST(Date ("2014-06-01") AS Uint64)),
       (2, 2, "Season 2", CAST(Date ("2015-04-12") AS Uint64), CAST(Date ("2015-06-14") AS Uint64)),
       (2, 3, "Season 3", CAST(Date ("2016-04-24") AS Uint64), CAST(Date ("2016-06-26") AS Uint64)),
       (2, 4, "Season 4", CAST(Date ("2017-04-23") AS Uint64), CAST(Date ("2017-06-25") AS Uint64)),
       (2, 5, "Season 5", CAST(Date ("2018-03-25") AS Uint64), CAST(Date ("2018-05-13") AS Uint64))
;

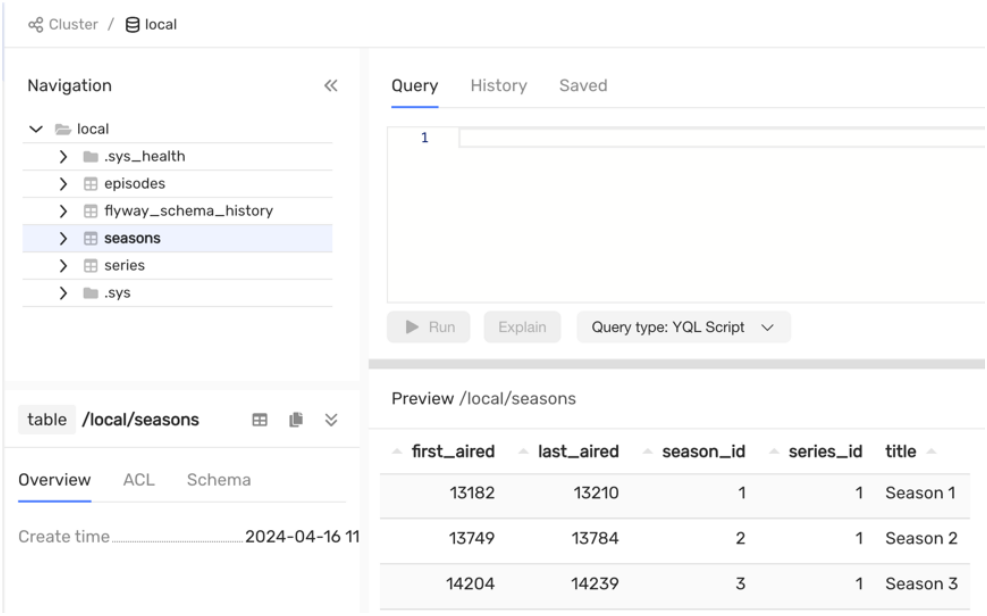
INSERT INTO episodes (series_id, season_id, episode_id, title, air_date)
VALUES (1, 1, 1, "Yesterday's Jam", CAST(Date ("2006-02-03") AS Uint64)),
       (1, 1, 2, "Calamity Jen", CAST(Date ("2006-02-03") AS Uint64)),
       (1, 1, 3, "Fifty-Fifty", CAST(Date ("2006-02-10") AS Uint64)),
       (1, 1, 4, "The Red Door", CAST(Date ("2006-02-17") AS Uint64)),
       (1, 1, 5, "The Haunting of Bill Crouse", CAST(Date ("2006-02-24") AS Uint64)),
       (1, 1, 6, "Aunt Irma Visits", CAST(Date ("2006-03-03") AS Uint64)),
       (1, 2, 1, "The Work Outing", CAST(Date ("2006-08-24") AS Uint64)),
       (1, 2, 2, "Return of the Golden Child", CAST(Date ("2007-08-31") AS Uint64)),
       (1, 2, 3, "Moss and the German", CAST(Date ("2007-09-07") AS Uint64)),
       (1, 2, 4, "The Dinner Party", CAST(Date ("2007-09-14") AS Uint64)),
       (1, 2, 5, "Smoke and Mirrors", CAST(Date ("2007-09-21") AS Uint64)),
       (1, 2, 6, "Men Without Women", CAST(Date ("2007-09-28") AS Uint64)),
       (1, 3, 1, "From Hell", CAST(Date ("2008-11-21") AS Uint64)),
       (1, 3, 2, "Are We Not Men?", CAST(Date ("2008-11-28") AS Uint64)),
       (1, 3, 3, "Tramps Like Us", CAST(Date ("2008-12-05") AS Uint64)),
       (1, 3, 4, "The Speech", CAST(Date ("2008-12-12") AS Uint64)),
       (1, 3, 5, "Friendface", CAST(Date ("2008-12-19") AS Uint64)),
       (1, 3, 6, "Calendar Geeks", CAST(Date ("2008-12-26") AS Uint64)),
       (1, 4, 1, "Jen The Fredo", CAST(Date ("2010-06-25") AS Uint64)),
       (1, 4, 2, "The Final Countdown", CAST(Date ("2010-07-02") AS Uint64)),
       (1, 4, 3, "Something Happened", CAST(Date ("2010-07-09") AS Uint64)),
       (1, 4, 4, "Italian For Beginners", CAST(Date ("2010-07-16") AS Uint64)),
       (1, 4, 5, "Bad Boys", CAST(Date ("2010-07-23") AS Uint64)),
       (1, 4, 6, "Reynholm vs Reynholm", CAST(Date ("2010-07-30") AS Uint64)),
       (2, 1, 1, "Minimum Viable Product", CAST(Date ("2014-04-06") AS Uint64)),
       (2, 1, 2, "The Cap Table", CAST(Date ("2014-04-13") AS Uint64)),
       (2, 1, 3, "Articles of Incorporation", CAST(Date ("2014-04-20") AS Uint64)),
       (2, 1, 4, "Fiduciary Duties", CAST(Date ("2014-04-27") AS Uint64)),
       (2, 1, 5, "Signaling Risk", CAST(Date ("2014-05-04") AS Uint64)),
       (2, 1, 6, "Third Party Insourcing", CAST(Date ("2014-05-11") AS Uint64)),
       (2, 1, 7, "Proof of Concept", CAST(Date ("2014-05-18") AS Uint64)),
       (2, 1, 8, "Optimal Tip-to-Tip Efficiency", CAST(Date ("2014-06-01") AS Uint64)),
       (2, 2, 1, "Sand Hill Shuffle", CAST(Date ("2015-04-12") AS Uint64)),
       (2, 2, 2, "Runaway Devaluation", CAST(Date ("2015-04-19") AS Uint64)),
       (2, 2, 3, "Bad Money", CAST(Date ("2015-04-26") AS Uint64)),
       (2, 2, 4, "The Lady", CAST(Date ("2015-05-03") AS Uint64)),
       (2, 2, 5, "Server Space", CAST(Date ("2015-05-10") AS Uint64)),
       (2, 2, 6, "Homicide", CAST(Date ("2015-05-17") AS Uint64)),
       (2, 2, 7, "Adult Content", CAST(Date ("2015-05-24") AS Uint64)),
       (2, 2, 8, "White Hat/Black Hat", CAST(Date ("2015-05-31") AS Uint64)),
```

```
        (2, 2, 9, "Binding Arbitration", CAST(Date ("2015-06-07") AS Uint64)),
        (2, 2, 10, "Two Days of the Condor", CAST(Date ("2015-06-14") AS Uint64)),
        (2, 3, 1, "Founder Friendly", CAST(Date ("2016-04-24") AS Uint64)),
        (2, 3, 2, "Two in the Box", CAST(Date ("2016-05-01") AS Uint64)),
        (2, 3, 3, "Meinertzhagen's Haversack", CAST(Date ("2016-05-08") AS Uint64)),
        (2, 3, 4, "Maleant Data Systems Solutions", CAST(Date ("2016-05-15") AS Uint64)),
        (2, 3, 5, "The Empty Chair", CAST(Date ("2016-05-22") AS Uint64)),
        (2, 3, 6, "Bachmanity Insanity", CAST(Date ("2016-05-29") AS Uint64)),
        (2, 3, 7, "To Build a Better Beta", CAST(Date ("2016-06-05") AS Uint64)),
        (2, 3, 8, "Bachman's Earnings Over-Ride", CAST(Date ("2016-06-12") AS Uint64)),
        (2, 3, 9, "Daily Active Users", CAST(Date ("2016-06-19") AS Uint64)),
        (2, 3, 10, "The Uptick", CAST(Date ("2016-06-26") AS Uint64)),
        (2, 4, 1, "Success Failure", CAST(Date ("2017-04-23") AS Uint64)),
        (2, 4, 2, "Terms of Service", CAST(Date ("2017-04-30") AS Uint64)),
        (2, 4, 3, "Intellectual Property", CAST(Date ("2017-05-07") AS Uint64)),
        (2, 4, 4, "Teambuilding Exercise", CAST(Date ("2017-05-14") AS Uint64)),
        (2, 4, 5, "The Blood Boy", CAST(Date ("2017-05-21") AS Uint64)),
        (2, 4, 6, "Customer Service", CAST(Date ("2017-05-28") AS Uint64)),
        (2, 4, 7, "The Patent Troll", CAST(Date ("2017-06-04") AS Uint64)),
        (2, 4, 8, "The Keenan Vortex", CAST(Date ("2017-06-11") AS Uint64)),
        (2, 4, 9, "Hooli-Con", CAST(Date ("2017-06-18") AS Uint64)),
        (2, 4, 10, "Server Error", CAST(Date ("2017-06-25") AS Uint64)),
        (2, 5, 1, "Grow Fast or Die Slow", CAST(Date ("2018-03-25") AS Uint64)),
        (2, 5, 2, "Reorientation", CAST(Date ("2018-04-01") AS Uint64)),
        (2, 5, 3, "Chief Operating Officer", CAST(Date ("2018-04-08") AS Uint64)),
        (2, 5, 4, "Tech Evangelist", CAST(Date ("2018-04-15") AS Uint64)),
        (2, 5, 5, "Facial Recognition", CAST(Date ("2018-04-22") AS Uint64)),
        (2, 5, 6, "Artificial Emotional Intelligence", CAST(Date ("2018-04-29") AS Uint64)),
        (2, 5, 7, "Initial Coin Offering", CAST(Date ("2018-05-06") AS Uint64)),
        (2, 5, 8, "Fifty-One Percent", CAST(Date ("2018-05-13") AS Uint64));
```

Применим последнюю миграцию следующей командой:

```
flyway -url=jdbc:ydb:grpc://localhost:2136/local -locations=db/migration migrate
```

В результате будут созданы таблицы `series`, `seasons` и `episodes`, которые будут заполнены данными:



Затем мы изменяем схему, добавляя вторичный индекс в таблицу `series`:

```
db/migration:
  V1__create_series.sql
  V2__create_seasons.sql
  V3__create_episodes.sql
  V4__load_data.sql
  V5__create_series_title_index.sql
```

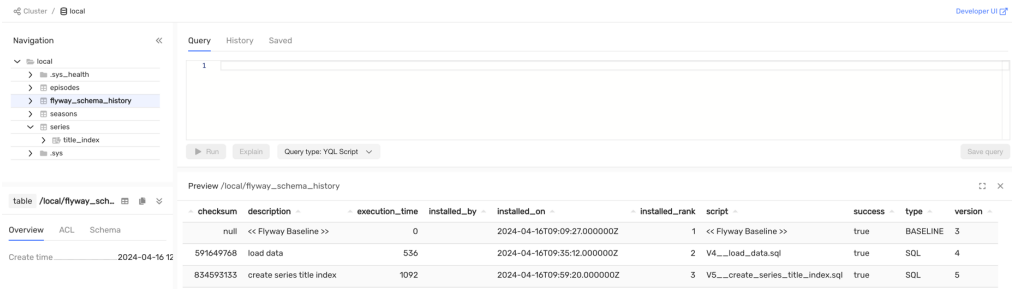
Содержимое файла `V5__create_series_title_index.sql`

```
ALTER TABLE `series` ADD INDEX `title_index` GLOBAL ON (`title`);
```

Применим последнюю миграцию следующей командой:

```
flyway -url=jdbc:ydb:grpc://localhost:2136/local -locations=db/migration migrate
```

В результате будет создан вторичный индекс для таблицы `series`:



info

Команда info печатает подробные сведения и информацию о состоянии всех миграций.

Добавим еще одну миграцию, которая переименовывает раннее добавленный вторичный индекс:

```
db/migration:
  V1__create_series.sql
  V2__create_seasons.sql
  V3__create_episodes.sql
  V4__load_data.sql
  V5__create_series_title_index.sql
  V6__rename_series_title_index.sql
```

Содержимое файла `V6__rename_series_title_index.sql`

```
ALTER TABLE `series` RENAME INDEX `title_index` TO `title_index_new`;
```

Результатом исполнения команды `flyway -url=jdbc:ydb:grpc://localhost:2136/local -locations=db/migration info` будет подробная информация о состоянии миграций:

```
+-----------+---------+---------------------------+----------+---------------------+--------------------+----
------+
| Category  | Version | Description               | Type     | Installed On        | State              | Und
oable |
+-----------+---------+---------------------------+----------+---------------------+--------------------+----
------+
| Versioned | 1       | create series             | SQL      |                     | Below Baseline     | No
|
| Versioned | 2       | create seasons            | SQL      |                     | Below Baseline     | No
|
| Versioned | 3       | create episodes           | SQL      |                     | Ignored (Baseline) | No
|
|           | 3       | << Flyway Baseline >>     | BASELINE | 2024-04-16 12:09:27 | Baseline           | No
|
| Versioned | 4       | load data                 | SQL      | 2024-04-16 12:35:12 | Success            | No
|
| Versioned | 5       | create series title index | SQL      | 2024-04-16 12:59:20 | Success            | No
|
| Versioned | 6       | rename series title index | SQL      |                     | Pending            | No
|
+-----------+---------+---------------------------+----------+---------------------+--------------------+----
------+
```

validate

Команда validate проверяет соответствие примененных миграций к миграциям, которые находятся в файловой системе пользователя.

После применения к текущим миграциям команды `flyway -url=jdbc:ydb:grpc://localhost:2136/local -locations=db/migration validate`, в логах будет написано, что последняя миграция не была применена к нашей базе данных:

```
ERROR: Validate failed: Migrations have failed validation
Detected resolved migration not applied to database: 6.
To fix this error, either run migrate, or set -ignoreMigrationPatterns='*:pending'.
```

Давайте исправим эту ошибку, выполнив `flyaway .. migrate` еще раз. Теперь проверка пройдет успешно, и вторичный индекс будет переименован.

Далее изменим файл уже ранее примененной миграции `V4__load_date.sql`, удалив комментарии в SQL скрипте.

После исполнения команды валидации, получим закономерную ошибку о том, что `checksum` различается в измененной миграции:

```
ERROR: Validate failed: Migrations have failed validation
Migration checksum mismatch for migration version 4
-> Applied to database : 591649768
-> Resolved locally    : 1923849782
```

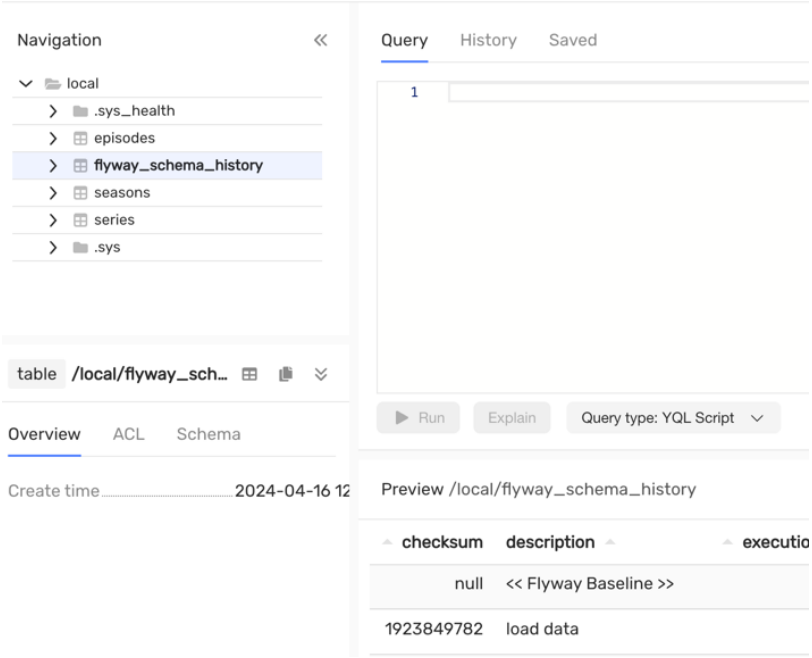
repair

Команда repair пытается исправить выявленные ошибки и несоответствия в таблице истории схемы базы данных.

Устраним проблему с разными `checksum`, выполнив следующую команду:

```
flyway -url=jdbc:ydb:grpc://localhost:2136/local -locations=db/migration repair
```

Результатом будет обновление колонки `checksum` в таблице `flyway_schema_history` у записи, отвечающей за миграцию `V4__load_data.sql`:



После восстановления таблицы лога, валидация проходит успешно.

Также с помощью команды `repair` можно удалить не удавшийся DDL скрипт.

clean

Команда clean удаляет все таблицы в схеме базы данных.

> **Важно**
>
> В отличие от других систем управления базами данных, YDB не имеет такую сущность как `SCHEMA`. Таким образом, команда `clean` удалит все пользовательские таблицы в вашей базе данных.

Удалим все таблицы в нашей базе данных следующей командой:

```
flyway -url=jdbc:ydb:grpc://localhost:2136/local -locations=db/migration -cleanDisabled=false clean
```

Результатом будет пустая база данных:



Версионирование схемы данных и миграции в YDB с использованием "goose"

Введение

Goose – open-source инструмент, который помогает версионировать схему данных в БД и управлять миграциями. Goose поддерживает множество различных баз данных, включая YDB. Goose использует файлы миграций и хранит состояние миграций непосредственно в базе данных в специальной таблице.

Установка goose

Варианты установки goose описаны в документации.

Аргументы запуска goose

Утилита `goose` вызывается командой:

```
$ goose <DB> <CONNECTION_STRING> <COMMAND> <COMMAND_ARGUMENTS>
```

где:

- `<DB>` - движок базы данных, в случае YDB следует писать `goose ydb`
- `<CONNECTION_STRING>` - строка подключения к базе данных.
- `<COMMAND>` - команда, которую требуется выполнить. Полный перечень команд доступен во встроенной справке (`goose help`).
- `<COMMAND_ARGUMENTS>` - аргументы команды.

Строка подключения к YDB

Для подключения к YDB следует использовать строку подключения вида

```
<protocol>://<host>:<port>/<database_path>?go_query_mode=scripting&go_fake_tx=scripting&go_query_bind=declar
e,numeric
```

где:

- `<protocol>` - протокол подключения (`grpc` для незащищенного соединения или `grpcs` для защищенного (`TLS`) соединения). При этом, для защищенного подключения (с `TLS`) следует явно подключить сертификаты YDB, например так: `export YDB_SSL_ROOT_CERTIFICATES_FILE=/path/to/ydb/certs/CA.pem` .
- `<host>` - адрес подключения к YDB.
- `<port>` - порт подключения к YDB.
- `<database_path>` - путь к базе данных в кластере YDB.
- `go_query_mode=scripting` - специальный режим `scripting` выполнения запросов по умолчанию в драйвере YDB. В этом режиме все запросы от goose направляются в YDB сервис `scripting` , который позволяет обрабатывать как `DDL` , так и `DML` инструкции `SQL` .
- `go_fake_tx=scripting` - поддержка эмуляции транзакций в режиме выполнения запросов через сервис YDB `scripting` . Дело в том, что в YDB выполнение `DDL` инструкций `SQL` в транзакции невозможно (или несет значительные накладные расходы). В частности сервис `scripting` не позволяет делать интерактивные транзакции (с явными `Begin` + `Commit` / `Rollback`). Соответственно, режим эмуляции транзакций на деле не делает ничего (`nop`) на вызовах `Begin` + `Commit` / `Rollback` из `goose` . Этот трюк в редких случаях может привести к тому, что отдельный шаг миграции может оказаться в промежуточном состоянии. Команда YDB работает на новым сервисом `query` , который должен помочь убрать этот риск.
- `go_query_bind=declare,numeric` - поддержка биндингов авто-выведения типов YQL из параметров запросов (`declare`) и поддержка биндингов нумерованных параметров (`numeric`). Дело в том, что `YQL` - язык со строгой типизацией, требующий явным образом указывать типы параметров запросов в теле самого `SQL` -запроса с помощью специальной инструкции `DECLARE` . Также `YQL` поддерживает только именованные параметры запроса (напрмиер, `$my_arg`), в то время как ядро `goose` генерирует `SQL` -запросы с нумерованными параметрами (`$1` , `$2` , и т.д.). Биндинги `declare` и `numeric` модифицируют исходные запросы из `goose` на уровне драйвера YDB, что позволило в конечном счете встроиться в `goose` .

В случае подключения к ломальному докер-контейнеру YDB строка подключения должна иметь вид:

```
grpc://localhost:2136/local?go_query_mode=scripting&go_fake_tx=scripting&go_query_bind=declare,numeric
```

Давайте сохраним эту строку в переменную окружения для дальнейшего использования:

```
export YDB_CONNECTION_STRING="grpc://localhost:2136/local?go_query_mode=scripting&go_fake_tx=scripting&go_que
ry_bind=declare,numeric"
```

Далее примеры вызова команд `goose` будут содержать именно эту строку подключения.

Директория с файлами миграций

Создадим директорию migrations и далее все команды `goose` следует выполнять в этой директории:

```
$ mkdir migrations && cd migrations
```

Управление миграциями с помощью goose

Создание файлов миграций и применение их к базе

Файл миграции можно создать командой `goose create` :

```
$ goose ydb $YDB_CONNECTION_STRING create 00001_create_first_table sql
2024/01/12 11:52:29 Created new file: 20240112115229_00001_create_first_table.sql
```

В результате выполнения команды был создан файл `<timestamp>_00001_create_table_users.sql`:

```
$ ls
20231215052248_00001_create_table_users.sql
```

Файл `<timestamp>_00001_create_table_users.sql` был создан со следующим содержимым:

```
-- +goose Up
-- +goose StatementBegin
SELECT 'up SQL query';
-- +goose StatementEnd

-- +goose Down
-- +goose StatementBegin
SELECT 'down SQL query';
-- +goose StatementEnd
```

Такая структура файла миграции помогает держать в контексте внимания ровно одну миграцию - шаги, чтобы обновить состояние базы, и шаги, чтобы откатить назад измения.

Файл миграции состоит из двух секций. Первая секция `+goose Up` содержит SQL-команды обновления схемы. Вторая секция `+goose Down` отвечает за откат изменений, выполненных в секции `+goose Up`. `Goose` заботливо вставил плейсхолдеры:

```
SELECT 'up SQL query';
```

и

```
SELECT 'down SQL query';
```

Мы можем заменить эти выражения на необходимые нам SQL-команды создания таблицы `users` и удаления ее в случае отката миграции:

```
-- +goose Up
-- +goose StatementBegin
CREATE TABLE users (
     id Uint64,
     username Text,
     created_at Timestamp,
     PRIMARY KEY (id)
);
-- +goose StatementEnd

-- +goose Down
-- +goose StatementBegin
DROP TABLE users;
-- +goose StatementEnd
```

Проверим статус миграций:

```
$ goose ydb $YDB_CONNECTION_STRING status
2024/01/12 11:53:50     Applied At                  Migration
2024/01/12 11:53:50     =======================================
2024/01/12 11:53:50     Pending                  -- 20240112115229_00001_create_first_table.sql
```

Статус `Pending` означает, что миграция еще не применена.

Применим миграцию с помощью команды `goose up`:

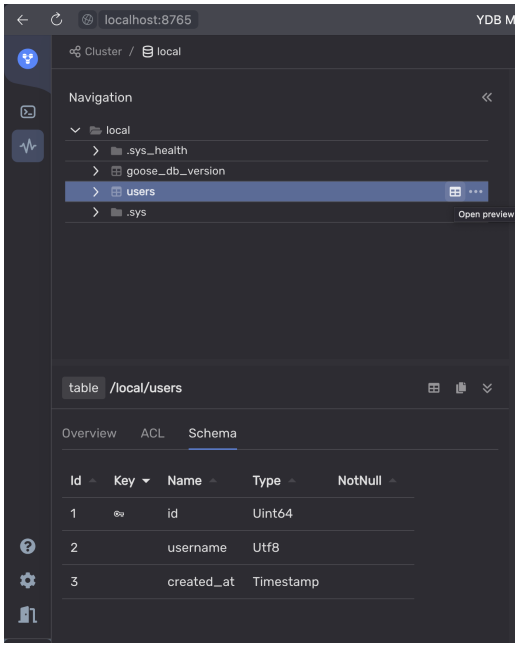
```
$ goose ydb $YDB_CONNECTION_STRING up
2024/01/12 11:55:18 OK   20240112115229_00001_create_first_table.sql (93.58ms)
2024/01/12 11:55:18 goose: successfully migrated database to version: 20240112115229
```

Проверим статус миграций `goose status`:

```
$ goose ydb $YDB_CONNECTION_STRING status
2024/01/12 11:56:00     Applied At                  Migration
2024/01/12 11:56:00     =======================================
2024/01/12 11:56:00     Fri Jan 12 11:55:18 2024 -- 20240112115229_00001_create_first_table.sql
```

Статус `Pending` заменился на временную отметку `Fri Jan 12 11:55:18 2024` - это означает, что миграция успешно применена. Мы также можем убедиться в этом и другими способами:

Используя YDB UI по адресу `http://localhost:8765`



Используя YDB CLI

```
$ ydb -e grpc://localhost:2136 -d /local scheme describe users
<table> users

Columns:
┌────────────┬────────────┬────────┬─────┐
| Name       | Type       | Family | Key |
├────────────┼────────────┼────────┼─────┤
| id         | Uint64?    |        | K0  |
| username   | Utf8?      |        |     |
| created_at | Timestamp? |        |     |
└────────────┴────────────┴────────┴─────┘

Storage settings:
Store large values in "external blobs": false

Column families:
┌─────────┬──────┬─────────────┬─────────────────┐
| Name    | Data | Compression | Keep in memory |
├─────────┼──────┼─────────────┼─────────────────┤
| default |      | None        |                 |
└─────────┴──────┴─────────────┴─────────────────┘

Auto partitioning settings:
Partitioning by size: true
Partitioning by load: false
Preferred partition size (Mb): 2048
Min partitions count: 1
```

Давайте создадим второй файл миграции с добавлением колонки `password_hash` в таблицу `users`:

```
$ goose ydb $YDB_CONNECTION_STRING create 00002_add_column_password_hash_into_table_users sql
2024/01/12 12:00:57 Created new file: 20240112120057_00002_add_column_password_hash_into_table_users.sql
```

Отредактируем файл `<timestamp>_00002_add_column_password_hash_into_table_users.sql` до следующего содержимого:

```
-- +goose Up
-- +goose StatementBegin
ALTER TABLE users ADD COLUMN password_hash Text;
-- +goose StatementEnd

-- +goose Down
-- +goose StatementBegin
ALTER TABLE users DROP COLUMN password_hash;
-- +goose StatementEnd
```

Проверим статус миграций:

```
$ goose ydb $YDB_CONNECTION_STRING status
2024/01/12 12:02:40     Applied At                  Migration
2024/01/12 12:02:40     =======================================
2024/01/12 12:02:40     Fri Jan 12 11:55:18 2024 -- 20240112115229_00001_create_first_table.sql
2024/01/12 12:02:40     Pending                  -- 20240112120057_00002_add_column_password_hash_into_table_
users.sql
```

Мы видим, что первая миграция применена, а вторая только запланирована (`Pending`).

Применим вторую миграцию с помощью команды `goose up-by-one` (в отличие от `goose uo` команда `goose up-by-one` применяет ровно одну "следующую" миграцию):

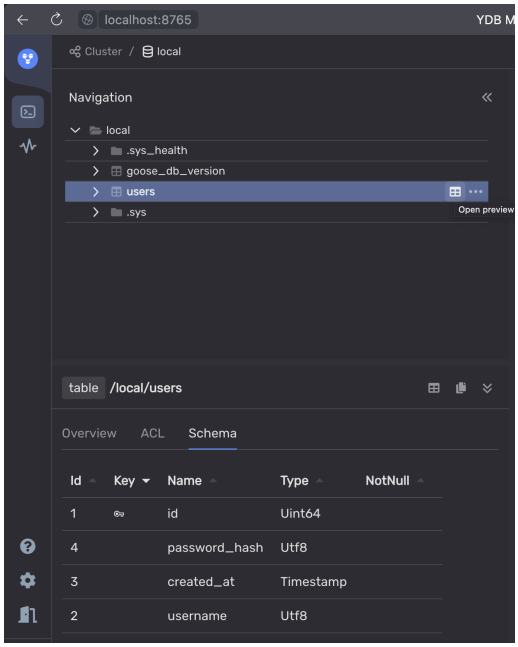
```
$ goose ydb $YDB_CONNECTION_STRING up-by-one
2024/01/12 12:04:56 OK   20240112120057_00002_add_column_password_hash_into_table_users.sql (59.93ms)
```

Проверим статус миграций:

```
$ goose ydb $YDB_CONNECTION_STRING status
2024/01/12 12:05:17     Applied At                  Migration
2024/01/12 12:05:17     =======================================
2024/01/12 12:05:17     Fri Jan 12 11:55:18 2024 -- 20240112115229_00001_create_first_table.sql
2024/01/12 12:05:17     Fri Jan 12 12:04:56 2024 -- 20240112120057_00002_add_column_password_hash_into_table_
users.sql
```

Обе миграции успешно применены. Убедимся в этом альтернативными способами:

Используя YDB UI по адресу `http://localhost:8765`



Используя YDB CLI

```
$ ydb -e grpc://localhost:2136 -d /local scheme describe users
<table> users

Columns:
┌───────────────┬──────────────┬────────┬─────┐
| Name          | Type         | Family | Key |
├───────────────┼──────────────┼────────┼─────┤
| id            | Uint64?      |        | K0  |
| username      | Utf8?        |        |     |
| created_at    | Timestamp?   |        |     |
| password_hash | Utf8?        |        |     |
└───────────────┴──────────────┴────────┴─────┘

Storage settings:
Store large values in "external blobs": false

Column families:
┌─────────┬──────┬─────────────┬──────────────────┐
| Name    | Data | Compression | Keep in memory |
├─────────┼──────┼─────────────┼──────────────────┤
| default |      | None        |                  |
└─────────┴──────┴─────────────┴──────────────────┘

Auto partitioning settings:
Partitioning by size: true
Partitioning by load: false
Preferred partition size (Mb): 2048
Min partitions count: 1
```

Все последующие миграции можно создавать аналогичным образом.

Откат миграции

Откатим последнюю миграцию с помощью команды `goose down`:

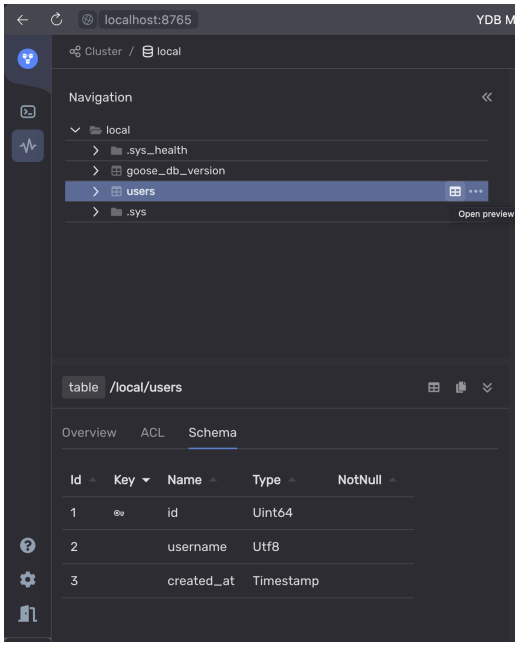
```
$ goose ydb $YDB_CONNECTION_STRING down
2024/01/12 13:07:18 OK   20240112120057_00002_add_column_password_hash_into_table_users.sql (43ms)
```

Проверим статус миграций `goose status`:

```
$ goose ydb $YDB_CONNECTION_STRING status
2024/01/12 13:07:36     Applied At                  Migration
2024/01/12 13:07:36     =======================================
2024/01/12 13:07:36     Fri Jan 12 11:55:18 2024 -- 20240112115229_00001_create_first_table.sql
2024/01/12 13:07:36     Pending                  -- 20240112120057_00002_add_column_password_hash_into_table_
users.sql
```

Статус `Fri Jan 12 12:04:56 2024` заменился на статус `Pending` - это означает, что последняя миграция успешно отменена. Мы также можем убедиться в этом и другими способами:

Используя YDB UI по адресу `http://localhost:8765`



Используя YDB CLI

```
$ ydb -e grpc://localhost:2136 -d /local scheme describe users
<table> users

Columns:
┌────────────┬────────────┬────────┬─────┐
| Name       | Type       | Family | Key |
├────────────┼────────────┼────────┼─────┤
| id         | Uint64?    |        | K0  |
| username   | Utf8?      |        |     |
| created_at | Timestamp? |        |     |
└────────────┴────────────┴────────┴─────┘

Storage settings:
Store large values in "external blobs": false

Column families:
┌─────────┬──────┬─────────────┬──────────────────┐
| Name    | Data | Compression | Keep in memory |
├─────────┼──────┼─────────────┼──────────────────┤
| default |      | None        |                  |
└─────────┴──────┴─────────────┴──────────────────┘

Auto partitioning settings:
Partitioning by size: true
Partitioning by load: false
Preferred partition size (Mb): 2048
Min partitions count: 1
```

"Горячий" список команд "goose"

Утилита `goose` позволяет управлять миграциями через командную строку:

- `goose status` - посмотреть статус применения миграций. Например, `goose ydb $YDB_CONNECTION_STRING status`.
- `goose up` - применить все известные миграции. Например, `goose ydb $YDB_CONNECTION_STRING up`.
- `goose up-by-one` - применить ровно одну "следующую" миграцию. Например, `goose ydb $YDB_CONNECTION_STRING up-by-one`.
- `goose redo` - пере-применить последнюю миграцию. Например, `goose ydb $YDB_CONNECTION_STRING redo`.
- `goose down` - откатить последнюю миграцию. Например, `goose ydb $YDB_CONNECTION_STRING down`.

- `goose reset` - откатить все миграции. Например, `goose ydb $YDB_CONNECTION_STRING reset`.

> **Важно**
>
> Будьте осторожны: команда `goose reset` может удалить все ваши миграции, включая данные в таблицах. Это происходит за счет инструкций в блоке `+goose Down`. Регулярно делайте бекапы и проверяйте их на возможность восстановления, чтобы минимизировать этот риск.

Миграции схемы данных YDB с помощью Liquibase

Введение

Liquibase – это библиотека с открытым исходным кодом для отслеживания, управления и применения изменений схемы базы данных. Она расширяется диалектами для различных систем управления базами данных (СУБД), включая YDB.

Диалект - это основная сущность в фреймворке Liquibase, которая помогает формировать SQL запросы к базе данных, учитывая специфику той или иной СУБД.

Возможности диалекта YDB

Основной функциональностью Liquibase является абстрактное описание схемы базы данных в `.xml`, `.json` или `.yaml` форматах. Что обеспечивает переносимость при смене одной СУБД на другую.

В диалекте поддержаны основные конструкции стандарта описания миграций (changeset).

Создание таблицы

Changeset `createTable` отвечает за создание таблицы. Описание типов из SQL стандарта сопоставляется с примитивными типами YDB. К примеру тип bigint будет конвертирован в Int64.

> **Примечание**
>
> Вы также можете явно указать имя исходного типа, например `Int32`, `Json`, `JsonDocument`, `Bytes` или `Interval`. Однако в этом случае схема не будет переносимой.

Таблица сравнения описаний типов Liquibase с типами YDB:

Liquibase типы	YDB тип
`boolean`, `java.sql.Types.BOOLEAN`, `java.lang.Boolean`, `bit`, `bool`	`Bool`
`blob`, `longblob`, `longvarbinary`, `String`, `java.sql.Types.BLOB`, `java.sql.Types.LONGBLOB`, `java.sql.Types.LONGVARBINARY`, `java.sql.Types.VARBINARY`, `java.sql.Types.BINARY`, `varbinary`, `binary`, `image`, `tinyblob`, `mediumblob`, `long binary`, `long varbinary`	`Bytes` (синоним `String`)
`java.sql.Types.DATE`, `smalldatetime`, `date`	`Date`
`decimal`, `java.sql.Types.DECIMAL`, `java.math.BigDecimal`	`Decimal(22,9)`
`double`, `java.sql.Types.DOUBLE`, `java.lang.Double`	`Double`
`float`, `java.sql.Types.FLOAT`, `java.lang.Float`, `real`, `java.sql.Types.REAL`	`Float`
`int`, `integer`, `java.sql.Types.INTEGER`, `java.lang.Integer`, `int4`, `int32`	`Int32`
`bigint`, `java.sql.Types.BIGINT`, `java.math.BigInteger`, `java.lang.Long`, `integer8`, `bigserial`, `long`	`Int64`
`java.sql.Types.SMALLINT`, `int2`, `smallserial`, `smallint`	`Int16`
`java.sql.Types.TINYINT`, `tinyint`	`Int8`
`char`, `java.sql.Types.CHAR`, `bpchar`, `character`, `nchar`, `java.sql.Types.NCHAR`, `nchar2`, `text`, `varchar`, `java.sql.Types.VARCHAR`, `java.lang.String`, `varchar2`, `character varying`, `nvarchar`, `java.sql.Types.NVARCHAR`, `nvarchar2`, `national`, `clob`, `longvarchar`, `longtext`, `java.sql.Types.LONGVARCHAR`, `java.sql.Types.CLOB`, `nclob`, `longnvarchar`, `ntext`, `java.sql.Types.LONGNVARCHAR`, `java.sql.Types.NCLOB`, `tinytext`, `mediumtext`, `long varchar`, `long nvarchar`	`Text` (синоним `Utf8`)
`timestamp`, `java.sql.Types.TIMESTAMP`, `java.sql.Timestamp`	`Timestamp`
`datetime`, `time`, `java.sql.Types.TIME`, `java.sql.Time`	`Datetime`

> **Важно**
>
> В YDB тип данных `Timestamp` хранит дату с точностью до микросекунд, в то время как `java.sql.Timestamp` или `java.time.Instant` хранят временную метку с точностью до наносекунд. Обратите на это внимание при работе с этим типом данных.
>
> Имена типов не чувствительны к регистру.

`dropTable` - удаление таблицы. Пример: `<dropTable tableName="episodes"/>`

Изменение структуры таблицы

`addColumn` - добавление колонки. Пример:

xml

```
<addColumn tableName="seasons">
    <column name="is_deleted" type="bool"/>
</addColumn>
```

json

```
"changes": [
    {
      "addColumn": {
        "tableName": "seasons",
        "columns": [
          {
            "column": {
              "name": "is_deleted",
              "type": "bool"
            }
          }
        ]
      }
    }
  ]
```

yaml

```
changes:
- addColumn:
    tableName: seasons
    columns:
    - column:
        name: is_deleted
        type: bool
```

`createIndex` - создание вторичного индекса. Пример:

xml

```
<createIndex tableName="episodes" indexName="episodes_index">
    <column name="title"/>
</createIndex>
```

json

```
"changes": [
    {
      "createIndex": {
        "tableName": "episodes",
        "indexName": "episodes_index",
        "columns": {
          "column": {
            "name": "title"
          }
        }
      }
    }
```

yaml

```
changes:
- createIndex:
    tableName: episodes
    indexName: episodes_index
    columns:
    - column:
        name: title
```

Важно

YDB не поддерживает уникальный вторичный индекс.

Примечание

Создание асинхронных индексов нужно делать через нативные SQL миграции.

`dropIndex` - удаление вторичного индекса. Пример:

xml

```
<dropIndex tableName="series" indexName="series_index"/>
```

json

```
"changes": [
  {
    "dropIndex": {
```

```
      "tableName": "series",
      "indexName": "series_index"
    }
  }
]
```

yaml

```
changes:
- dropIndex:
    tableName: series
    indexName: series_index
```

Ввод данных в таблицу

`loadData`, `loadUpdateData` - загрузка данных из `CSV` файла. `loadUpdateData` загружает данные командой UPSERT INTO.

`insert` - changeset, который осуществляет единичный insert в таблицу командой INSERT INTO. Например:

xml

```
<insert tableName="episodes">
    <column name="series_id" valueNumeric="1"/>
    <column name="season_id" valueNumeric="1"/>
    <column name="episode_id" valueNumeric="1"/>
    <column name="title" value="Yesterday's Jam"/>
    <column name="air_date" valueDate="2023-04-03T08:46:23.456"/>
</insert>
```

json

```
"changes": [
  {
    "insert": {
      "tableName": "episodes",
      "columns": [
        {
          "column": {
            "name": "series_id",
            "valueNumeric": "1"
          }
        },
        {
          "column": {
            "name": "season_id",
            "valueNumeric": "1"
          }
        },
        {
          "column": {
            "name": "episode_id",
            "valueNumeric": "1"
          }
        },
        {
          "column": {
            "name": "title",
            "value": "Yesterday's Jam"
          }
        },
        {
          "column": {
            "name": "air_date",
            "valueDate": "2023-04-03T08:46:23.456"
          }
        }
      ]
    }
  }
]
```

yaml

```
changes:
- insert:
    tableName: episodes
    columns:
      - column:
          name: series_id
          valueNumeric: 1
      - column:
          name: season_id
          valueNumeric: 1
      - column:
          name: episode_id
          valueNumeric: 1
      - column:
          name: title
```

```
          value: Yesterday's Jam
      - column:
          name: air_date
          valueDate: 2023-04-03T08:46:23.456
```

Вы также можете указать любое значение в поле `value`. Данные из поля `value` в наборе изменений `insert` или `CSV` файлов будут автоматически преобразованы в требуемые типы с учетом строгой типизации в YDB.

Таблица форматирования типов для загрузки в таблицу:

YDB тип	Формат описания
`Bool`	`true` или `false`
`Int8`, `Int16`, `Int32`, `Int64`	Целочисленное число
`Uint8`, `Uint16`, `Uint32`, `Uint64`	Целое число без знака
`Text`, `Bytes`, `Json`, `JsonDocument`	Строковое описание
`Float`, `Double`, `Decimal(22, 9)`	Вещественное число
`Interval`	ISO-8601, соответствует классу `java.time.Duration` в Java.
`Date`	Шаблон `YYYY-MM-DD` календарной даты из стандарта ISO-8601
`Datetime`	Шаблон `YYYY-MM-DDThh:mm:ss`, timezone будет установлена `UTC`
`Timestamp`	Временная метка из стандарта ISO-8601 соответствует классу `java.time.Instant` в Java, timezone будет установлена `UTC` (точность в микросекундах - ограничение типа данных `Timestamp` в YDB)

Пример `CSV` файла:

```
id,bool,bigint,smallint,tinyint,float,double,decimal,uint8,uint16,uint32,uint64,text,binary,json,jsondocumen
t,date,datetime,timestamp,interval
2,true,123123,13000,112,1.123,1.123123,1.123123,12,13,14,15,kurdyukov-kir,binary,{"asd": "asd"},{"asd": "as
d"},2014-04-06,2023-09-16T12:30,2023-07-31T17:00:00.00Z,PT10S
```

Важно

Чтобы понять, какие SQL-конструкции может выполнять YDB и какие существуют ограничения на типы данных, ознакомьтесь с документацией по языку запросов YQL.

Совет

Важно отметить, что кастомные инструкции YQL могут быть применены с помощью нативных SQL-запросов.

Как воспользоваться?

Есть два способа:

Программно из Java / Kotlin приложения

Как воспользоваться из Java / Kotlin подробно описано в README проекта. Также есть пример приложения Spring Boot, использующего его.

Liquibase CLI

Для начала нужно установить саму утилиту liquibase любым из рекомендуемых способов. Затем нужно подложить актуальные .jar архивы YDB JDBC драйвера и Liquibase диалекта YDB в папку `internal/lib`.

```
# $(which liquibase)
cd ./internal/lib/

# you may need to sudo
# set actual versions of .jar files
curl -L -o ydb-jdbc-driver.jar https://repo.maven.apache.org/maven2/tech/ydb/jdbc/ydb-jdbc-driver-shaded/2.0.
7/ydb-jdbc-driver-shaded-2.0.7.jar
curl -L -o liquibase-ydb-dialect.jar https://repo.maven.apache.org/maven2/tech/ydb/dialects/liquibase-ydb-dia
lect/1.0.0/liquibase-ydb-dialect-1.0.0.jar
```

Более подробное описание в разделе manual library management в документации Liquibase.

Теперь утилиту командной строки liquibase можно использовать с YDB.

Сценарии использования liquibase

Инициализация liquibase на пустой YDB

Основной командой является `liquibase update`, которая применяет миграции, если текущая схема YDB отстает от пользовательского описания.

Далее в примерах предполагается, использование следующего файла настроек `liquibase.properties`:

```
changelog-file=changelogs.xml
url=jdbc:ydb:grpc://localhost:2136/local
```

Применим к пустой базе данных следующий changeset:

```
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog
        xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
                      http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.8.xsd">

    <changeSet id="episodes" author="kurdyukov-kir">
        <comment>Table episodes.</comment>

        <createTable tableName="episodes">
            <column name="series_id" type="bigint">
                <constraints primaryKey="true"/>
            </column>
            <column name="season_id" type="bigint">
                <constraints primaryKey="true"/>
            </column>
            <column name="episode_id" type="bigint">
                <constraints primaryKey="true"/>
            </column>

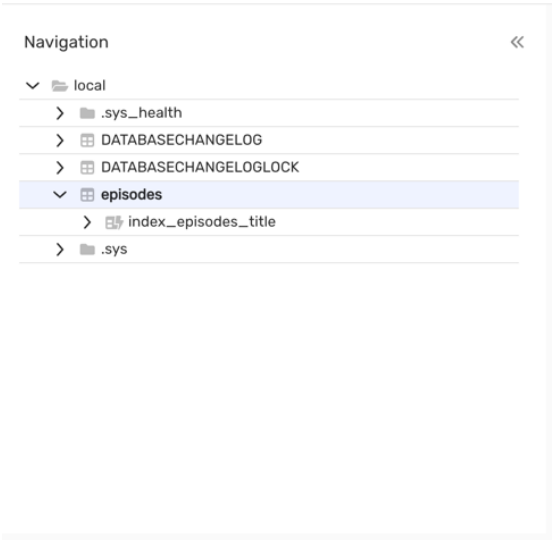
            <column name="title" type="text"/>
            <column name="air_date" type="timestamp"/>
        </createTable>
        <rollback>
            <dropTable tableName="episodes"/>
        </rollback>
    </changeSet>
    <changeSet id="index_episodes_title" author="kurdyukov-kir">
        <createIndex tableName="episodes" indexName="index_episodes_title">
            <column name="title"/>
        </createIndex>
    </changeSet>
</databaseChangeLog>
```

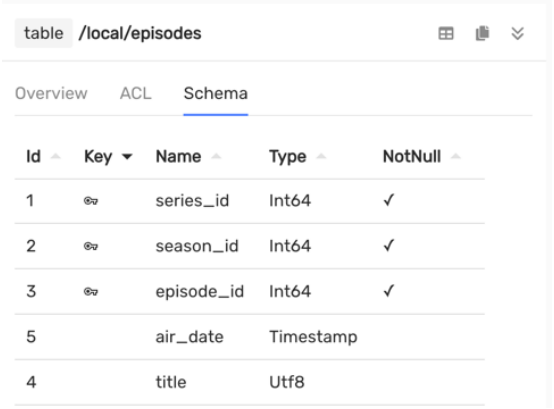
После исполнения команды `liquibase update`, liquibase напечатает следующий лог:

```
UPDATE SUMMARY
Run:                          2
Previously run:               0
Filtered out:                 0
-------------------------------
Total change sets:            2

Liquibase: Update has been successful. Rows affected: 2
Liquibase command 'update' was executed successfully.
```

После применения миграций схемы данных выглядит следующим образом:





Можно увидеть, что Liquibase создал две служебные таблицы `DATABASECHANGELOG` - лог миграций, `DATABASECHANGELOGLOCK` - таблица для взятия распределенной блокировки.

Содержимое таблицы DATABASECHANGELOG:

AUTHOR	COMMENTS	CONTEXTS	DATEEXECUTED	DEPLOYMENT_ID	DESCRIPTION	EXECTYPE	FILENAME
kurdyukov-kir	Table episodes.		12:53:27	1544007500	createTable tableName=episodes	EXECUTED	migration/episodes.xml
kurdyukov-kir	""		12:53:28	1544007500	createIndex indexName=index_episodes_title, tableName=episodes	EXECUTED	migration/episodes.xml

Эволюция схемы базы данных

Допустим нам нужно создать YDB топик и выключить параметр `AUTO_PARTITIONING_BY_SIZE` таблицы. Это можно сделать нативным SQL скриптом:

```
--liquibase formatted sql

--changeset kurdyukov-kir:create-a-topic
CREATE TOPIC `my_topic` (
    CONSUMER my_consumer
) WITH (
     retention_period = Interval('P1D')
);

--changeset kurdyukov-kir:auto-partitioning-disabled
ALTER TABLE episodes SET (AUTO_PARTITIONING_BY_SIZE = DISABLED);
```

Также добавим новую колонку `is_deleted` и удалим индекс `index_episodes_title`:

```
<changeSet id="alter-episodes" author="kurdyukov-kir">
    <comment>Alter table episodes.</comment>

    <dropIndex tableName="episodes" indexName="index_episodes_title"/>

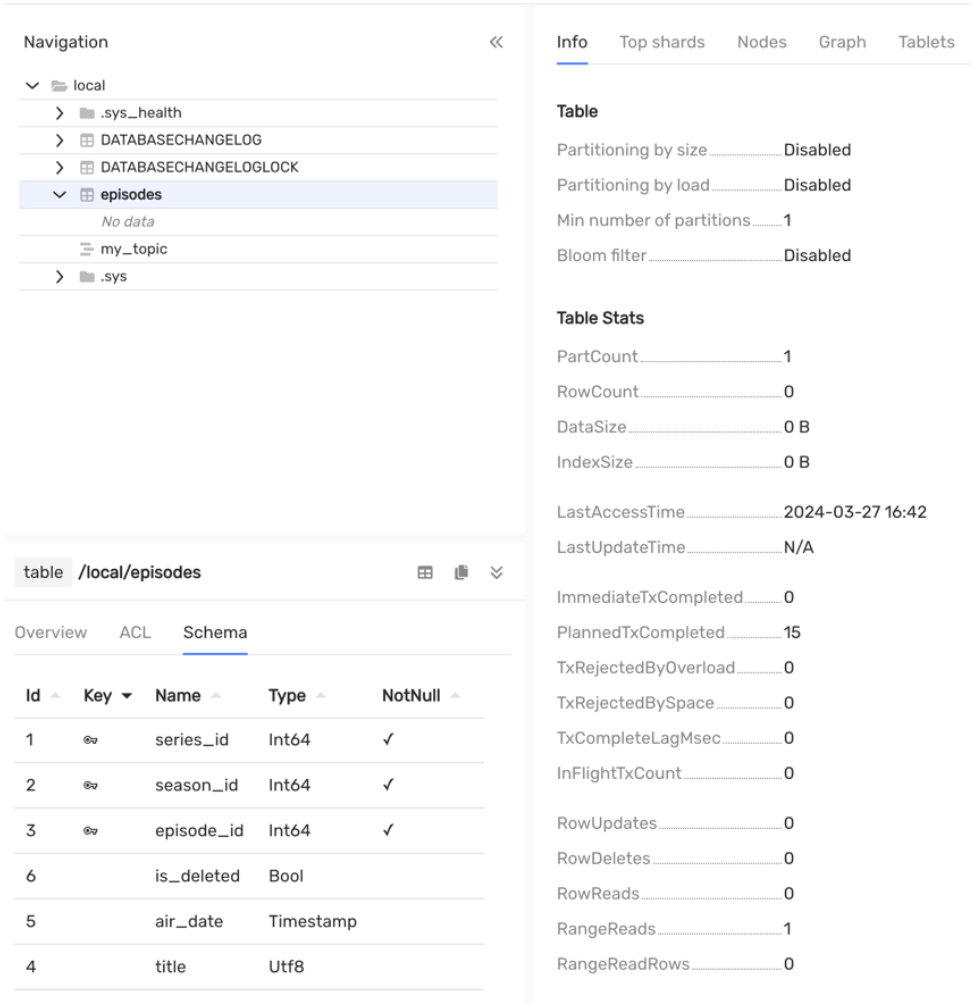
    <addColumn tableName="episodes">
        <column name="is_deleted" type="bool"/>
    </addColumn>
</changeSet>
<include file="/migration/sql/yql.sql" relativeToChangelogFile="true"/>
```

После исполнения `liquibase update` схема базы успешно обновится.

```
UPDATE SUMMARY
Run:                          3
Previously run:               2
Filtered out:                 0
-------------------------------
Total change sets:            5

Liquibase: Update has been successful. Rows affected: 3
Liquibase command 'update' was executed successfully.
```

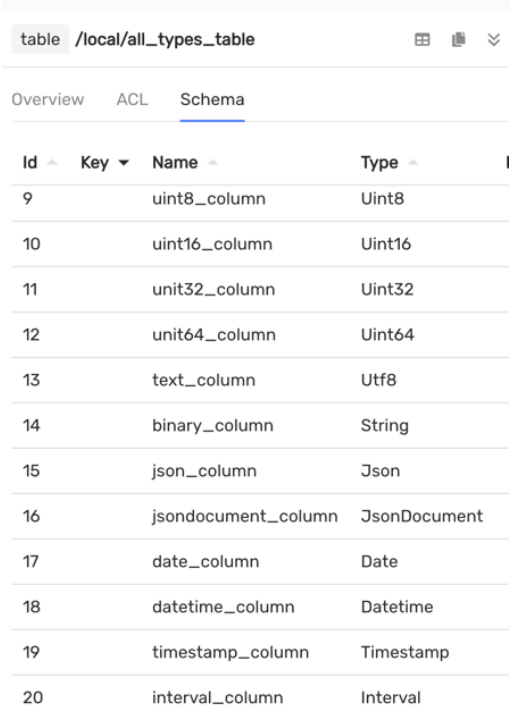
Результатом будет удаление индекса, добавление колонки `is_deleted`, выключение параметра `AUTO_PARTITIONING_BY_SIZE`, а также создание топика:



Инициализация liquibase в проекте с непустой схемой данных

Предположим, что я имею существующий проект с текущей схемой базы данных:





Id	Key	Name	Type
9		uint8_column	Uint8
10		uint16_column	Uint16
11		unit32_column	Uint32
12		unit64_column	Uint64
13		text_column	Utf8
14		binary_column	String
15		json_column	Json
16		jsondocument_column	JsonDocument
17		date_column	Date
18		datetime_column	Datetime
19		timestamp_column	Timestamp
20		interval_column	Interval

Чтобы начать использовать liquibase, требуется выполнить:

```
liquibase generate-changelog --changelog-file=changelog.xml
```

Содержимое сгенерированного changelog.xml:

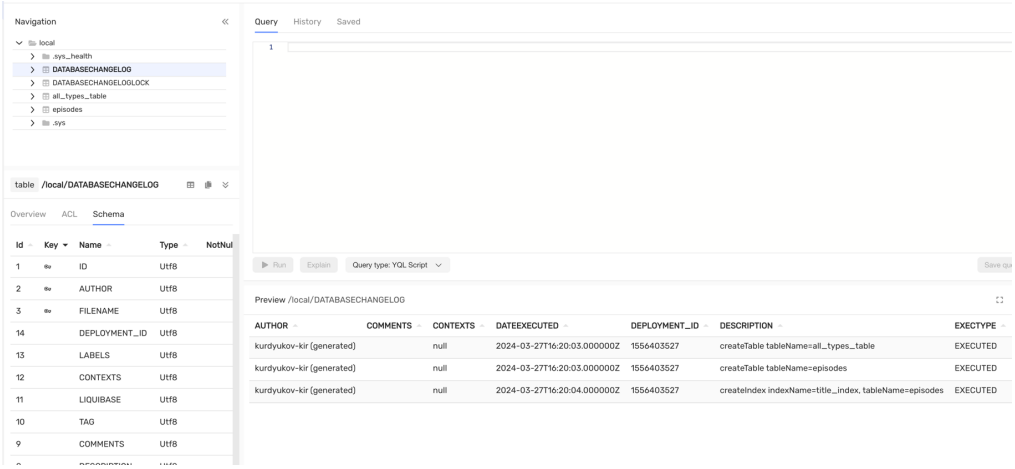
```
<changeSet author="kurdyukov-kir (generated)" id="1711556283305-1">
    <createTable tableName="all_types_table">
        <column name="id" type="INT32">
            <constraints nullable="false" primaryKey="true"/>
        </column>
        <column name="bool_column" type="BOOL"/>
        <column name="bigint_column" type="INT64"/>
        <column name="smallint_column" type="INT16"/>
        <column name="tinyint_column" type="INT8"/>
        <column name="float_column" type="FLOAT"/>
        <column name="double_column" type="DOUBLE"/>
        <column name="decimal_column" type="DECIMAL(22, 9)"/>
        <column name="uint8_column" type="UINT8"/>
        <column name="uint16_column" type="UINT16"/>
        <column name="unit32_column" type="UINT32"/>
        <column name="unit64_column" type="UINT64"/>
        <column name="text_column" type="TEXT"/>
        <column name="binary_column" type="BYTES"/>
        <column name="json_column" type="JSON"/>
        <column name="jsondocument_column" type="JSONDOCUMENT"/>
        <column name="date_column" type="DATE"/>
        <column name="datetime_column" type="DATETIME"/>
        <column name="timestamp_column" type="TIMESTAMP"/>
        <column name="interval_column" type="INTERVAL"/>
    </createTable>
</changeSet>
<changeSet author="kurdyukov-kir (generated)" id="1711556283305-2">
    <createTable tableName="episodes">
        <column name="series_id" type="INT64">
            <constraints nullable="false" primaryKey="true"/>
        </column>
        <column name="season_id" type="INT64">
            <constraints nullable="false" primaryKey="true"/>
        </column>
        <column name="episode_id" type="INT64">
            <constraints nullable="false" primaryKey="true"/>
        </column>
        <column name="title" type="TEXT"/>
        <column name="air_date" type="DATE"/>
```

```
    </createTable>
</changeSet>
<changeSet author="kurdyukov-kir (generated)" id="1711556283305-3">
    <createIndex indexName="title_index" tableName="episodes">
        <column name="title"/>
    </createIndex>
</changeSet>
```

Затем нужно синхронизировать сгенерированный changelog.xml файл, делается это командой:

```
liquibase changelog-sync --changelog-file=dbchangelog.xml
```

Результатом будет синхронизация liquibase в вашем проекте:



Подключение к YDB

В вышеуказанных примерах мы использовали Docker контейнер, для которого не требовалось дополнительных параметров для аутентификации.

Существуют различные варианты настройки аутентификации через параметр URL:

- Docker контейнер (anonymous authentication):
  `jdbc:ydb:grpc://localhost:2136/local`
- Self-hosted кластер:
  `jdbc:ydb:grpcs://<host>:2135/Root/testdb?secureConnectionCertificate=file:~/myca.cer`
- Подключение с использованием токена:
  `jdbc:ydb:grpcs://<host>:2135/path/to/database?token=file:~/my_token`
- Подключение с использованием сервисного аккаунта:
  `jdbc:ydb:grpcs://<host>:2135/path/to/database?saFile=file:~/sa_key.json`

Если ваш кластер настроен с использованием логина и пароля, процесс аутентификации происходит через параметры Liquibase.

За дополнительной информацией о различных настройках аутентификации обратитесь к соответствующему разделу.

Движки выполнения запросов

Движок	Инструкция
Apache Spark™	Инструкция

Apache Spark™

Apache Spark™ — распределённая система обработки данных с открытым исходным кодом, предназначенная для обработки больших объёмов данных. Она использует кеширование в оперативной памяти и обеспечивает оптимальное выполнение запросов для быстрого анализа данных любого размера. Предоставляется API для разработки на Java, Scala, Python и R, а также поддерживается повторное использование кода в различных сценариях: пакетная обработка, интерактивные запросы, аналитика в реальном времени, машинное обучение и обработка графов. Apache Spark™ может работать с YDB с помощью YDB Spark Connector — специального модуля, предоставляющего реализацию основных примитивов Apache Spark™. Поддерживаются:

- Распределение операций по партициям таблиц YDB;
- Параллельная запись и чтение таблиц YDB;
- Автоматическое создание таблиц при их отсутствии.

> **Примечание**
>
> Для более быстрой работы YDB Spark Connector может потребоваться увеличить объем памяти, доступной для каждого исполнителя Apache Spark™. Рекомендуется указывать 4 ГБ или больше на один executor.

Где взять и как использовать

Для использования YDB в Apache Spark™ необходимо добавить YDB Spark Connector в драйвер Apache Spark™. Это можно сделать несколькими способами:

- Загрузить коннектор напрямую из Maven Central с помощью опции `--packages`. Рекомендуется использовать последнюю опубликованную версию:

  Spark Shell

  ```
  ~ $ spark-shell --master <master-url> --packages tech.ydb.spark:ydb-spark-connector-shaded:2.0.1 --conf
  spark.executor.memory=4g
  ```

  PySpark

  ```
  ~ $ pyspark --master <master-url> --packages tech.ydb.spark:ydb-spark-connector-shaded:2.0.1 --conf spar
  k.executor.memory=4g
  ```

  Spark SQL

  ```
  ~ $ spark-sql --master <master-url> --packages tech.ydb.spark:ydb-spark-connector-shaded:2.0.1 --conf sp
  ark.executor.memory=4g
  ```

- Скачать последнюю версию shaded-сборки (вариант коннектора, включающий все зависимости) из GitHub или Maven Central и указать скачанный артефакт в опции `--jars`:

  Spark Shell

  ```
  ~ $ spark-shell --master <master-url> --jars ~/Download/ydb-spark-connector-shaded-2.0.1.jar --conf spar
  k.executor.memory=4g
  ```

  PySpark

  ```
  ~ $ pyspark --master <master-url> --jars ~/Download/ydb-spark-connector-shaded-2.0.1.jar  --conf spark.e
  xecutor.memory=4g
  ```

  Spark SQL

  ```
  ~ $ spark-sql --master <master-url> --jars ~/Download/ydb-spark-connector-shaded-2.0.1.jar  --conf spar
  k.executor.memory=4g
  ```

- Также скачанную shaded-сборку можно скопировать в папку `jars` дистрибутива Apache Spark™. В таком случае никаких опций указывать не требуется.

Работа через DataFrame API

DataFrame API позволяет работать с YDB в интерактивных `spark-shell` или `pyspark`, а также при написании кода на `Java`, `Scala` или `Python` для `spark-submit`.

Для создания `DataFrame` нужно указать формат `ydb`, передать набор опций подключения и путь к таблице YDB:

Scala

```scala
val ydb_df = spark.read.format("ydb").options(<options>).load("<table-path>")
```

Python

```python
ydb_df = spark.read.format("ydb").options(<options>).load("<table-path>")
```

Для сохранения любого `DataFrame` в таблицу YDB аналогично нужно указать формат `ydb`, опции подключения и путь к таблице:

Scala

```
any_dataframe.write.format("ydb").options(<options>).mode("append").save("<table-path>")
```

Python

```
any_dataframe.write.format("ydb").options(<options>).mode("append").save("<table-path>")
```

> **Примечание**
>
> При записи данных в YDB рекомендуется использовать режим `append`, который использует пакетную загрузку данных. Если указанная в методе `save()` таблица не существует, она будет создана в соответствии с опциями автосоздания таблиц.

Более подробный пример приведён в разделе Пример работы с YDB в spark-shell.

Работа через Catalog API

С помощью каталогов возможна работа с YDB в интерактивном режиме `spark-sql` или при выполнении SQL-запросов через метод `spark.sql`.
В этом случае для доступа к YDB нужно добавить каталог, указав следующие опции Apache Spark™ (можно определить несколько каталогов с разными именами для разных баз данных YDB):

```
# Обязательный параметр типа каталога
spark.sql.catalog.<catalog_name>=tech.ydb.spark.connector.YdbCatalog
# Обязательный параметр url
spark.sql.catalog.<catalog_name>.url=<ydb-connection-url>
# Все остальные параметры не обязательны и указываются только при необходимости
spark.sql.catalog.<catalog_name>.<param-name>=<param-value>
```

После задания каталогов можно работать с таблицами YDB через стандартные SQL-запросы Apache Spark™. Обратите внимание, что в качестве разделителя в пути к таблице нужно использовать `.`:

```
SELECT * FROM <catalog_name>.<table-path> LIMIT 10;
```

Более подробный пример приведён в разделе Пример работы с YDB в spark-sql.

Список параметров YDB Spark Connector

Поведение YDB Spark Connector настраивается с помощью опций, которые могут передаваться в виде одного набора с помощью метода `options` или указываться по одной с помощью метода `option`. При этом каждый `DataFrame` и даже каждая отдельная операция над `DataFrame` может иметь свой набор опций.

Опции подключения

- `url` — обязательный параметр с адресом подключения к YDB. Имеет вид `grpc[s]://<endpoint>:<port>/<database>[?<options>]`.
  Примеры использования:
  - Локальный Docker-контейнер с анонимной аутентификацией и без TLS:
    `grpc://localhost:2136/local`
  - Удалённый кластер, размещённый на собственном сервере:
    `grpcs://my-private-cluster:2135/Root/my-database?secureConnectionCertificate=~/myCertificate.cer`
  - Экземпляр облачной базы данных с токеном:
    `grpcs://ydb.my-cloud.com:2135/my_folder/test_database?tokenFile=~/my_token`
  - Экземпляр облачной базы данных с файлом сервисного аккаунта:
    `grpcs://ydb.my-cloud.com:2135/my_folder/test_database?saKeyFile=~/sa_key.json`
- `ca.text` — указывает значение сертификата для установки TLS-соединения.
- `ca.file` — указывает путь к сертификату для установки TLS-соединения. Может быть указан прямо в `url` в виде опции `secureConnectionCertificate`.
- `auth.use_env` — если указано `true`, используется режим аутентификации на основе переменных среды окружения.
- `auth.use_metadata` — если указано `true`, используется режим аутентификации Metadata. Может быть указан прямо в `url` в виде опции `useMetadata`.
- `auth.login` и `auth.password` — логин и пароль для статической аутентификации.
- `auth.token` — аутентификация с использованием указанного Access Token.
- `auth.token.file` — аутентификация с использованием Access Token из указанного файла. Может быть указан прямо в `url` в виде опции `tokenFile`.
- `auth.sakey.text` — можно указать содержимое ключа для аутентификации по ключу сервисного аккаунта.
- `auth.sakey.file` — можно указать путь к файлу ключа для аутентификации по ключу сервисного аккаунта. Может быть указан прямо в `url` в виде опции `saKeyFile`.

Опции автоматического создания таблиц

> **Совет**
>
> Если вам нужно настроить параметры таблицы, создайте её вручную заранее с помощью CREATE TABLE или измените их позже с помощью ALTER TABLE.

- `table.autocreate` — если указано `true`, то при записи в несуществующую таблицу она будет создана автоматически. По умолчанию включено.
- `table.type` — тип автоматически создаваемой таблицы. Возможные варианты:
  - `row` — для создания строчной таблицы (по умолчанию);
  - `column` — для создания колоночной таблицы;
- `table.primary_keys` — разделённый запятой список колонок для использования в качестве первичного ключа. При отсутствии этой опции для ключа будет автоматически создана новая колонка.
- `table.auto_pk_name` — имя колонки для автоматически создаваемого ключа. Эта колонка будет создана с типом `Utf8` и будет заполняться случайными UUID v4 значениями. По умолчанию `_spark_key`.

Пример работы с YDB в spark-shell и pyspark

В качестве примера покажем, как загрузить в YDB список всех постов StackOverflow за 2020 год. Эти данные доступны в виде Parquet-файла, расположенного по адресу https://datasets-documentation.s3.eu-west-3.amazonaws.com/stackoverflow/parquet/posts/2020.parquet:

Spark Shell

```
~ $ spark-shell --master <master-url> --packages tech.ydb.spark:ydb-spark-connector-shaded:2.0.1 --conf spar
k.executor.memory=4g
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.5.4
      /_/

Using Scala version 2.12.18 (OpenJDK 64-Bit Server VM, Java 17.0.15)
Type in expressions to have them evaluated.
Type :help for more information.
```

PySpark

```
~ $ pyspark --master <master-url> --packages tech.ydb.spark:ydb-spark-connector-shaded:2.0.1 --conf spark.exe
cutor.memory=4g
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.5.4
      /_/

Using Python version 3.10.12 (main, May 27 2025 17:12:29)
SparkSession available as 'spark'.
```

Выведем схему файла с данными и посмотрим количество строк в нём:

Spark Shell

```
scala> val so_posts2020 = spark.read.format("parquet").load("/home/username/2020.parquet")
so_posts2020: org.apache.spark.sql.DataFrame = [Id: bigint, PostTypeId: bigint ... 20 more fields]

scala> so_posts2020.printSchema
root
|-- Id: long (nullable = true)
|-- PostTypeId: long (nullable = true)
|-- AcceptedAnswerId: long (nullable = true)
|-- CreationDate: timestamp (nullable = true)
|-- Score: long (nullable = true)
|-- ViewCount: long (nullable = true)
|-- Body: binary (nullable = true)
|-- OwnerUserId: long (nullable = true)
|-- OwnerDisplayName: binary (nullable = true)
|-- LastEditorUserId: long (nullable = true)
|-- LastEditorDisplayName: binary (nullable = true)
|-- LastEditDate: timestamp (nullable = true)
|-- LastActivityDate: timestamp (nullable = true)
|-- Title: binary (nullable = true)
|-- Tags: binary (nullable = true)
|-- AnswerCount: long (nullable = true)
|-- CommentCount: long (nullable = true)
|-- FavoriteCount: long (nullable = true)
|-- ContentLicense: binary (nullable = true)
|-- ParentId: binary (nullable = true)
|-- CommunityOwnedDate: timestamp (nullable = true)
|-- ClosedDate: timestamp (nullable = true)
```

```
scala> so_posts2020.count
res1: Long = 4304021
```

PySpark

```
>>> so_posts2020 = spark.read.format("parquet").load("/home/username/2020.parquet")
>>> so_posts2020.printSchema()
root
|-- Id: long (nullable = true)
|-- PostTypeId: long (nullable = true)
|-- AcceptedAnswerId: long (nullable = true)
|-- CreationDate: timestamp (nullable = true)
|-- Score: long (nullable = true)
|-- ViewCount: long (nullable = true)
|-- Body: binary (nullable = true)
|-- OwnerUserId: long (nullable = true)
|-- OwnerDisplayName: binary (nullable = true)
|-- LastEditorUserId: long (nullable = true)
|-- LastEditorDisplayName: binary (nullable = true)
|-- LastEditDate: timestamp (nullable = true)
|-- LastActivityDate: timestamp (nullable = true)
|-- Title: binary (nullable = true)
|-- Tags: binary (nullable = true)
|-- AnswerCount: long (nullable = true)
|-- CommentCount: long (nullable = true)
|-- FavoriteCount: long (nullable = true)
|-- ContentLicense: binary (nullable = true)
|-- ParentId: binary (nullable = true)
|-- CommunityOwnedDate: timestamp (nullable = true)
|-- ClosedDate: timestamp (nullable = true)

>>> so_posts2020.count()
4304021
```

Добавим к данному DataFrame новую колонку с годом и запишем всё это в колоночную таблицу YDB:

Spark Shell

```
scala> val my_ydb = Map("url" -> "grpcs://ydb.my-host.net:2135/preprod/spark-test?tokenFile=~/.token")
my_ydb: scala.collection.immutable.Map[String,String] = Map(url -> grpcs://ydb.my-host.net:2135/preprod/spark
-test?tokenFile=~/.token)

scala> so_posts2020.withColumn("Year", lit(2020)).write.format("ydb").options(my_ydb).option("table.type", "c
olumn").option("table.primary_keys", "Id").mode("append").save("stackoverflow/posts");
```

PySpark

```
>>> from pyspark.sql.functions import col,lit
>>> my_ydb = {"url": "grpcs://ydb.my-host.net:2135/preprod/spark-test?tokenFile=~/.token"}
>>> so_posts2020.withColumn("Year", lit(2020)).write.format("ydb").options(**my_ydb).option("table.type", "co
lumn").option("table.primary_keys", "Id").mode("append").save("stackoverflow/posts")
```

В итоге мы можем прочитать записанные данные из YDB и, например, подсчитать количество постов с подтверждённым ответом:

Spark Shell

```
scala> val ydb_posts2020 = spark.read.format("ydb").options(my_ydb).load("stackoverflow/posts")
ydb_posts2020: org.apache.spark.sql.DataFrame = [Id: bigint, PostTypeId: bigint ... 21 more fields]

scala> ydb_posts2020.printSchema
root
|-- Id: long (nullable = false)
|-- PostTypeId: long (nullable = true)
|-- AcceptedAnswerId: long (nullable = true)
|-- CreationDate: timestamp (nullable = true)
|-- Score: long (nullable = true)
|-- ViewCount: long (nullable = true)
|-- Body: binary (nullable = true)
|-- OwnerUserId: long (nullable = true)
|-- OwnerDisplayName: binary (nullable = true)
|-- LastEditorUserId: long (nullable = true)
|-- LastEditorDisplayName: binary (nullable = true)
|-- LastEditDate: timestamp (nullable = true)
|-- LastActivityDate: timestamp (nullable = true)
|-- Title: binary (nullable = true)
|-- Tags: binary (nullable = true)
|-- AnswerCount: long (nullable = true)
|-- CommentCount: long (nullable = true)
|-- FavoriteCount: long (nullable = true)
|-- ContentLicense: binary (nullable = true)
|-- ParentId: binary (nullable = true)
|-- CommunityOwnedDate: timestamp (nullable = true)
|-- ClosedDate: timestamp (nullable = true)
|-- Year: integer (nullable = true)

scala> ydb_posts2020.count
```

```
res3: Long = 4304021

scala> ydb_posts2020.filter(col("AcceptedAnswerId") > 0).count
res4: Long = 843780
```

PySpark

```
>>> ydb_posts2020 = spark.read.format("ydb").options(**my_ydb).load("stackoverflow/posts")
>>> ydb_posts2020.printSchema()
root
|-- Id: long (nullable = true)
|-- PostTypeId: long (nullable = true)
|-- AcceptedAnswerId: long (nullable = true)
|-- CreationDate: timestamp (nullable = true)
|-- Score: long (nullable = true)
|-- ViewCount: long (nullable = true)
|-- Body: binary (nullable = true)
|-- OwnerUserId: long (nullable = true)
|-- OwnerDisplayName: binary (nullable = true)
|-- LastEditorUserId: long (nullable = true)
|-- LastEditorDisplayName: binary (nullable = true)
|-- LastEditDate: timestamp (nullable = true)
|-- LastActivityDate: timestamp (nullable = true)
|-- Title: binary (nullable = true)
|-- Tags: binary (nullable = true)
|-- AnswerCount: long (nullable = true)
|-- CommentCount: long (nullable = true)
|-- FavoriteCount: long (nullable = true)
|-- ContentLicense: binary (nullable = true)
|-- ParentId: binary (nullable = true)
|-- CommunityOwnedDate: timestamp (nullable = true)
|-- ClosedDate: timestamp (nullable = true)
|-- Year: integer (nullable = true)

>>> ydb_posts2020.count()
4304021
>>> ydb_posts2020.filter(col("AcceptedAnswerId") > 0).count()
843780
```

Пример работы с YDB в spark-sql

В качестве примера покажем, как загрузить в YDB список всех постов StackOverflow за 2020 год. Эти данные доступны в виде Parquet-файла, расположенного по адресу https://datasets-documentation.s3.eu-west-3.amazonaws.com/stackoverflow/parquet/posts/2020.parquet:

Для начала запустим `spark-sql` с настроенным каталогом `my_ydb`:

```
~ $ spark-sql --master <master-url> --packages tech.ydb.spark:ydb-spark-connector-shaded:2.0.1 \
     --conf spark.sql.catalog.my_ydb=tech.ydb.spark.connector.YdbCatalog \
     --conf spark.sql.catalog.my_ydb.url=grpcs://ydb.my-host.net:2135/preprod/spark-test \
     --conf spark.sql.catalog.my_ydb.auth.token.file=~/.token \
     --conf spark.executor.memory=4g
spark-sql (default)>
```

Просмотрим текущее содержимое подключенной базы данных и убедимся в отсутствии таблицы `stackoverflow/posts`:

```
spark-sql (default)> SHOW NAMESPACES FROM my_ydb;
stackoverflow
Time taken: 0.11 seconds, Fetched 1 row(s)
spark-sql (default)> SHOW TABLES FROM my_ydb.stackoverflow;
Time taken: 0.041 seconds
```

Подсчитаем количество строк в оригинальном файле:

```
spark-sql (default)> SELECT COUNT(*) FROM parquet.`/home/username/2020.parquet`;
4304021
```

Добавим новую колонку с годом и запишем всё это в таблицу YDB:

```
spark-sql (default)> CREATE TABLE my_ydb.stackoverflow.posts OPTIONS(table.primary_keys='Id') AS SELECT *, 20
20 as Year FROM parquet.`/home/username/2020.parquet`;
Time taken: 85.225 seconds
```

Убедимся, что новая таблица появилась в базе данных YDB:

```
spark-sql (default)> SHOW TABLES FROM my_ydb.stackoverflow;
posts
Time taken: 0.07 seconds, Fetched 1 row(s)
spark-sql (default)> DESCRIBE TABLE my_ydb.stackoverflow.posts;
Id                      bigint
PostTypeId              bigint
AcceptedAnswerId        bigint
CreationDate            timestamp
Score                   bigint
ViewCount               bigint
```

```
Body                  binary
OwnerUserId           bigint
OwnerDisplayName      binary
LastEditorUserId      bigint
LastEditorDisplayName binary
LastEditDate          timestamp
LastActivityDate      timestamp
Title                 binary
Tags                  binary
AnswerCount           bigint
CommentCount          bigint
FavoriteCount         bigint
ContentLicense        binary
ParentId              binary
CommunityOwnedDate    timestamp
ClosedDate            timestamp
Year                  int
```

В итоге мы можем прочитать записанные данные из YDB и, например, подсчитать количество постов с подтверждённым ответом:

```
spark-sql (default)> SELECT COUNT(*) FROM my_ydb.stackoverflow.posts;
4304021
Time taken: 19.726 seconds, Fetched 1 row(s)
spark-sql (default)> SELECT COUNT(*) FROM my_ydb.stackoverflow.posts WHERE AcceptedAnswerId > 0;
843780
Time taken: 6.599 seconds, Fetched 1 row(s)
```

Объектно-реляционное отображение (ORM)

Система	Инструкция
Hibernate	Инструкция
Spring Data JDBC	Инструкция
Kotlin Exposed	Инструкция
JOOQ	Инструкция
Dapper	Инструкция
Entity Framework	Инструкция
Linq To DB	Инструкция
SQLAlchemy	Инструкция
Django	Инструкция
Drizzle ORM	Инструкция

Диалект YDB для Hibernate

Введение

Это руководство использования Hibernate с YDB.

Hibernate - это фреймворк объектно-реляционного отображения (ORM) для Java, облегчающий процесс отображения объектно-ориентированных моделей.

Установка диалекта YDB

Примеры для различных систем сборки:

Maven

```
<!-- Set actual versions -->
<dependency>
    <groupId>tech.ydb.jdbc</groupId>
    <artifactId>ydb-jdbc-driver</artifactId>
    <version>${ydb.jdbc.version}</version>
</dependency>

<dependency>
    <groupId>tech.ydb.dialects</groupId>
    <artifactId>hibernate-ydb-dialect</artifactId>
    <version>${hibernate.ydb.dialect.version}</version>
</dependency>
```

Gradle

```
dependencies {
    // Set actual versions
    implementation "tech.ydb.dialects:hibernate-ydb-dialect:$ydbDialectVersion"
    implementation "tech.ydb.jdbc:ydb-jdbc-driver:$ydbJdbcVersion"
}
```

Если вы используете Hibernate версии 5, вам понадобится `<artifactId>hibernate-ydb-dialect-v5</artifactId>` для Maven или `implementation "tech.ydb.dialects:hibernate-ydb-dialect-v5:$version"` для Gradle вместо аналогичного пакета без `-v5` суффикса.

Конфигурация диалекта

Сконфигурируйте Hibernate для использования диалекта YDB, обновив persistence.xml файл:

```
<property name="hibernate.dialect">tech.ydb.hibernate.dialect.YdbDialect</property>
```

Или, если вы используете программную настройку:

Java

```
import org.hibernate.cfg.AvailableSettings;
import org.hibernate.cfg.Configuration;

public static Configuration basedConfiguration() {
    return new Configuration()
            .setProperty(AvailableSettings.JAKARTA_JDBC_DRIVER, YdbDriver.class.getName())
            .setProperty(AvailableSettings.DIALECT, YdbDialect.class.getName());
}
```

Kotlin

```
import org.hibernate.cfg.AvailableSettings
import org.hibernate.cfg.Configuration

fun basedConfiguration(): Configuration = Configuration().apply {
    setProperty(AvailableSettings.JAKARTA_JDBC_DRIVER, YdbDriver::class.name)
    setProperty(AvailableSettings.DIALECT, YdbDialect::class.name)
}
```

Использование

Используйте этот диалект так же, как и любой другой диалект Hibernate. Сопоставьте классы сущностей с таблицами базы данных и используйте фабрику сессий Hibernate для выполнения операций с базой данных.

Таблица сопоставления Java типов с YDB типами:

Java type	YDB type
`bool`, `Boolean`	`Bool`
`String`, enum с аннотацией `@Enumerated(EnumType.STRING)`	`Text` (синоним `Utf8`)
`java.time.LocalDate`	`Date`
`java.math.BigDecimal`, `java.math.BigInteger`	`Decimal(22,9)`

`double`, `Double`	`Double`
`float`, `Float`	`Float`
`int`, `java.lang.Integer`	`Int32`
`long`, `java.lang.Long`	`Int64`
`short`, `java.lang.Short`	`Int16`
`byte`, `java.lang.Byte`, enum с аннотацией `@Enumerated(EnumType.ORDINAL)`	`Int8`
`[]byte`	`Bytes` (синоним `String`)
`java.time.LocalDateTime` (timezone будет установлена в `UTC`)	`Datetime`
`java.time.Instant` (timezone будет установлена в `UTC`)	`Timestamp`

Диалект YDB поддерживает генерацию схемы базы данных на основе объектов Hibernate.

Например, для класса `Group`:

Java

```
@Getter
@Setter
@Entity
@Table(name = "Groups", indexes = @Index(name = "group_name_index", columnList = "GroupName"))
public class Group {

    @Id
    @Column(name = "GroupId")
    private int id;

    @Column(name = "GroupName")
    private String name;

    @OneToMany(mappedBy = "group")
    private List<Student> students;
}
```

Kotlin

```
@Entity
@Table(name = "Groups", indexes = [Index(name = "group_name_index", columnList = "GroupName")])
data class Group(
    @Id
    @Column(name = "GroupId")
    val id: Int,

    @Column(name = "GroupName")
    val name: String,

    @OneToMany(mappedBy = "group")
    val students: List<Student>
)
```

Будет сгенерирована следующая таблица `Groups` и вторичный индекс `group_name_index` к колонке `GroupName`:

```
CREATE TABLE Groups (
    GroupId Int32 NOT NULL,
    GroupName Text,
    PRIMARY KEY (GroupId)
);

ALTER TABLE Groups
  ADD INDEX group_name_index GLOBAL
       ON (GroupName);
```

Если эволюционировать сущность Group путем добавления поля `deparment`:

Java

```
@Column
private String department;
```

Kotlin

```
@Column
val department: String
```

Hibernate при старте приложения обновит схему базы данных, если установлен режим `update`:

```
jakarta.persistence.schema-generation.database.action=update
```

Результат изменения схемы:

```
ALTER TABLE Groups
   ADD COLUMN department Text
```

> **Важно**
>
> Hibernate не предназначен для управления схемами баз данных. Вы можете использовать Liquibase или Flyway для управления схемой базы данных.

Диалект YDB поддерживает `@OneToMany`, `@ManyToOne`, `@ManyToMany` связи.

Например, сгенерированный SQL скрипт для `@OneToMany`:

FetchType.LAZY

```
SELECT
    g1_0.GroupId,
    g1_0.GroupName
FROM
    Groups g1_0
WHERE
    g1_0.GroupName='M3439'

SELECT
    s1_0.GroupId,
    s1_0.StudentId,
    s1_0.StudentName
FROM
    Students s1_0
WHERE
    s1_0.GroupId=?
```

FetchType.EAGER

```
SELECT
    g1_0.GroupId,
    g1_0.GroupName,
    s1_0.GroupId,
    s1_0.StudentId,
    s1_0.StudentName
FROM
    Groups g1_0
JOIN
    Students s1_0
        on g1_0.GroupId=s1_0.GroupId
WHERE
    g1_0.GroupName='M3439'
```

Пример с Spring Data JPA

Настройте Spring Data JPA для использования диалекта YDB, обновив свой `application.properties`:

```
spring.jpa.properties.hibernate.dialect=tech.ydb.hibernate.dialect.YdbDialect
spring.datasource.driver-class-name=tech.ydb.jdbc.YdbDriver
spring.datasource.url=jdbc:ydb:<grpc/grpcs>://<host>:<2135/2136>/path/to/database[?saFile=file:~/sa_key.json]
```

Создадим простую сущность и репозиторий:

Java

```
@Data
@Entity
@Table(name = "employee")
public class Employee {
    @Id
    private long id;

    @Column(name = "full_name")
    private String fullName;

    @Column
    private String email;

    @Column(name = "hire_date")
    private LocalDate hireDate;

    @Column
    private java.math.BigDecimal salary;

    @Column(name = "is_active")
    private boolean isActive;

    @Column
    private String department;
```

```
    @Column
    private int age;
}

public interface EmployeeRepository implements CrudRepository<Employee, Long> {}
```

Kotlin

```
@Entity
@Table(name = "employee")
data class Employee(
    @Id
    val id: Long,

    @Column(name = "full_name")
    val fullName: String,

    @Column
    val email: String,

    @Column(name = "hire_date")
    val hireDate: LocalDate,

    @Column
    val salary: java.math.BigDecimal,

    @Column(name = "is_active")
    val isActive: Boolean,

    @Column
    val department: String,

    @Column
    val age: Int,
)

interface EmployeeRepository : CrudRepository<Employee, Long>

fun EmployeeRepository.findByIdOrNull(id: Long): Employee? = this.findById(id).orElse(null)
```

Пример использования:

Java

```
Employee employee = new Employee(
    1,
    "Example",
    "example@bk.com",
    LocalDate.parse("2023-12-20"),
    BigDecimal("500000.000000000"),
    true,
    "YDB AppTeam",
    23
);

/* The following SQL will be executed:
INSERT INTO employee (age,department,email,full_name,hire_date,is_active,limit_date_password,salary,id)
VALUES (?,?,?,?,?,?,?,?,?)
*/
employeeRepository.save(employee);

assertEquals(employee, employeeRepository.findById(employee.getId()).get());

/* The following SQL will be executed:
DELETE FROM employee WHERE id=?
 */
employeeRepository.delete(employee);

assertNull(employeeRepository.findById(employee.getId()).orElse(null));
```

Kotlin

```
val employee = Employee(
    1,
    "Example",
    "example@bk.com",
    LocalDate.parse("2023-12-20"),
    BigDecimal("500000.000000000"),
    true,
    "YDB AppTeam",
    23
)

/* The following SQL will be executed:
INSERT INTO employee (age,department,email,full_name,hire_date,is_active,limit_date_password,salary,id)
VALUES (?,?,?,?,?,?,?,?,?)
*/
employeeRepository.save(employee)
```

```
assertEquals(employee, employeeRepository.findByIdOrNull(employee.id))

/* The following SQL will be executed:
DELETE FROM employee WHERE id=?
 */
employeeRepository.delete(employee)

assertNull(employeeRepository.findByIdOrNull(employee.id))
```

Пример простого приложения Spring Data JPA можно найти по ссылке.

Диалект YDB для Spring Data JDBC

Это руководство предназначено для использования Spring Data JDBC с YDB.

Spring Data JDBC является частью экосистемы Spring Data, которая предоставляет упрощенный подход к взаимодействию с реляционными базами данных посредством использования SQL и простых Java объектов. В отличие от Spring Data JPA, который построен на базе JPA (Java Persistence API), Spring Data JDBC предлагает более прямолинейный способ работы с базами данных, исключающий сложности, связанные с ORM (Object-Relational Mapping).

Установка диалекта YDB

Для интеграции YDB с вашим проектом Spring Data JDBC потребуется добавить две зависимости: YDB JDBC Driver и расширение Spring Data JDBC для YDB.

Примеры для различных систем сборки:

Maven

```
<!-- Set actual versions -->
<dependency>
    <groupId>tech.ydb.jdbc</groupId>
    <artifactId>ydb-jdbc-driver</artifactId>
    <version>${ydb.jdbc.version}</version>
</dependency>

<dependency>
    <groupId>tech.ydb.dialects</groupId>
    <artifactId>spring-data-jdbc-ydb</artifactId>
    <version>${spring.data.jdbc.ydb}</version>
</dependency>
```

Gradle

```
dependencies {
    // Set actual versions
    implementation "tech.ydb.dialects:spring-data-jdbc-ydb:$ydbDialectVersion"
    implementation "tech.ydb.jdbc:ydb-jdbc-driver:$ydbJdbcVersion"
}
```

Использование

После импорта всех необходимых зависимостей диалект готов к использованию. Рассмотрим простой пример Spring Data JDBC приложения.

```
spring.datasource.driver-class-name=tech.ydb.jdbc.YdbDriver
spring.datasource.url=jdbc:ydb:<grpc/grpcs>://<host>:<2135/2136>/path/to/database[?saFile=file:~/sa_key.json]
```

```
@Table(name = "Users")
public class User implements Persistable<Long> {
    @Id
    private Long id = ThreadLocalRandom.current().nextLong();

    private String login;
    private String firstname;
    private String lastname;

    @Transient
    private boolean isNew;

    // Конструкторы, геттеры и сеттеры

    @Override
    public Long getId() {
        return id;
    }

    @Override
    public boolean isNew() {
        return isNew;
    }

    public void setNew(boolean isNew) {
        this.isNew = isNew;
    }
}
```

Для сущности `User` таблицы `Users` создадим репозиторий:

```
public interface SimpleUserRepository extends CrudRepository<User, Long> {
}
```

Давайте сохраним нового пользователя и проверим, что он был успешно сохранен:

```
@Component
public class UserRepositoryCommandLineRunner implements CommandLineRunner {
```

```
    @Autowired
    private SimpleUserRepository repository;

    @Override
    public void run(String... args) {
        User user = new User();
        user.setLogin("johndoe");
        user.setFirstname("John");
        user.setLastname("Doe");
        user.setNew(true);  // Устанавливаем флаг новой сущности

        // Сохранение пользователя
        User savedUser = repository.save(user);

        // Проверка сохранения пользователя
        assertThat(repository.findById(savedUser.getId())).contains(savedUser);

        System.out.println("User saved with ID: " + savedUser.getId());
    }
}
```

View Index

Для генерации конструкций `VIEW INDEX` из методов репозитория необходимо использовать аннотацию `@ViewIndex`.
Аннотация `@ViewIndex` имеет два поля:

- `indexName` - название индекса.
- `tableName` - название таблицы, к которой привязан `VIEW INDEX`, особенно полезно при использовании аннотаций `@MappedCollection`.

Рассмотрим простой пример с индексом таблицы Users по полю `login`:

```
public interface SimpleUserRepository extends CrudRepository<User, Long> {

    @ViewIndex(indexName = "login_index")
    User findByLogin(String login);
}
```

Запрос, который сгенерирует этот метод, будет выглядеть следующим образом:

```
SELECT `Users`.`id`        AS `id`,
       `Users`.`login`     AS `login`,
       `Users`.`lastname`  AS `lastname`,
       `Users`.`firstname` AS `firstname`
FROM `Users` VIEW login_index AS `Users`
WHERE `Users`.`login` = ?
```

YdbType

Для указания конкретного типа данных в YDB можно использовать аннотацию `@YdbType` над полем сущности.
Пример использования:

```
    @YdbType("Json")
    private String jsonColumn;
    @YdbType("JsonDocument")
    private String jsonDocumentColumn;
    @YdbType("Uint8")
    private byte uint8Column;
    @YdbType("Uint16")
    private short uint16Column;
    @YdbType("Uint32")
    private int uint32Column;
    @YdbType("Uint64")
    private long uint64Column;
```

Использование аннотации `@YdbType` позволяет точно указать типы данных, поддерживаемые YDB, что обеспечивает корректное взаимодействие с базой данных.

Пример простого приложения Spring Data JDBC можно найти по ссылке.

Расширение JOOQ для использования с YDB

Это руководство предназначено для использования JOOQ с YDB.

JOOQ — это библиотека для Java, которая позволяет создавать типобезопасные SQL-запросы путём генерации Java-классов из схемы базы данных и использования удобных конструкторов запросов.

Генерация Java-классов

Генерировать Java-классы можно с помощью любых инструментов, представленных на официальном сайте JOOQ, используя две зависимости: YDB JDBC Driver и расширение JOOQ для YDB. Также необходимо указать два параметра:

- `database.name` : `tech.ydb.jooq.codegen.YdbDatabase` (обязательная настройка)
- `strategy.name` : `tech.ydb.jooq.codegen.YdbGeneratorStrategy` (рекомендуется к использованию)

Рассмотрим на примере `maven` плагина:

```
<plugin>
    <groupId>org.jooq</groupId>
    <artifactId>jooq-codegen-maven</artifactId>
    <version>3.19.11</version>
    <executions>
        <execution>
            <goals>
                <goal>generate</goal>
            </goals>
        </execution>
    </executions>
    <dependencies>
        <dependency>
            <groupId>tech.ydb.jdbc</groupId>
            <artifactId>ydb-jdbc-driver</artifactId>
            <version>${ydb.jdbc.version}</version>
        </dependency>
        <dependency>
            <groupId>tech.ydb.dialects</groupId>
            <artifactId>jooq-ydb-dialect</artifactId>
            <version>${jooq.ydb.version}</version>
        </dependency>
    </dependencies>
    <configuration>
        <jdbc>
            <driver>tech.ydb.jdbc.YdbDriver</driver>
            <url>jdbc:ydb:grpc://localhost:2136/local</url>
        </jdbc>
        <generator>
            <strategy>
                <name>tech.ydb.jooq.codegen.YdbGeneratorStrategy</name>
            </strategy>
            <database>
                <name>tech.ydb.jooq.codegen.YdbDatabase</name>
                <!-- исключение системных таблицы -->
                <excludes>.sys.*</excludes>
            </database>
            <target>
                <packageName>ydb</packageName>
                <directory>./src/main/java</directory>
            </target>
        </generator>
    </configuration>
</plugin>
```

Пример сгенерированных классов из туториала по YQL (их полный код доступен на GitHub):

```
ydb/DefaultCatalog.java
ydb/default_schema
ydb/default_schema/tables
ydb/default_schema/tables/Seasons.java
ydb/default_schema/tables/records
ydb/default_schema/tables/records/SeriesRecord.java
ydb/default_schema/tables/records/EpisodesRecord.java
ydb/default_schema/tables/records/SeasonsRecord.java
ydb/default_schema/tables/Series.java
ydb/default_schema/tables/Episodes.java
ydb/default_schema/Indexes.java
ydb/default_schema/Keys.java
ydb/default_schema/Tables.java
ydb/default_schema/DefaultSchema.java
```

Использование

Для интеграции YDB с JOOQ в ваш проект потребуется добавить две зависимости: YDB JDBC Driver и расширение JOOQ для YDB.

Примеры для различных систем сборки:

Maven

```
<!-- Set actual versions -->
<dependency>
```

```
    <groupId>tech.ydb.jdbc</groupId>
    <artifactId>ydb-jdbc-driver</artifactId>
    <version>${ydb.jdbc.version}</version>
</dependency>

<dependency>
    <groupId>tech.ydb.dialects</groupId>
    <artifactId>jooq-ydb-dialect</artifactId>
    <version>${jooq.ydb.dialect.version}</version>
</dependency>
```

Gradle

```
dependencies {
    // Set actual versions
    implementation "tech.ydb.dialects:jooq-ydb-dialect:$jooqYdbDialectVersion"
    implementation "tech.ydb.jdbc:ydb-jdbc-driver:$ydbJdbcVersion"
}
```

Для получения `YdbDSLContext` (расширение `org.jooq.DSLContext`) используйте класс `tech.ydb.jooq.YDB`. Например:

```
String url = "jdbc:ydb:<schema>://<host>:<port>/path/to/database[?saFile=file:~/sa_key.json]";
Connection conn = DriverManager.getConnection(url);

YdbDSLContext dsl = YDB.using(conn);
```

или

```
String url = "jdbc:ydb:<schema>://<host>:<port>/path/to/database[?saFile=file:~/sa_key.json]";
try(CloseableYdbDSLContext dsl = YDB.using(url)) {
    // ...
}
```

`YdbDSLContext` готов к использованию.

YQL команды

В `YdbDSLContext` доступны следующие команды, специфичные для синтаксиса YQL:

- `UPSERT`:

```
// generated SQL:
// upsert into `episodes` (`series_id`, `season_id`, `episode_id`, `title`, `air_date`)
// values (?, ?, ?, ?, ?)
public void upsert(YdbDSLContext context) {
    context.upsertInto(EPISODES)
            .set(record)
            .execute();
}
```

- `REPLACE`:

```
// generated SQL:
// replace into `episodes` (`series_id`, `season_id`, `episode_id`, `title`, `air_date`)
// values (?, ?, ?, ?, ?)
public void replace(YdbDSLContext context) {
    ydbDSLContext.replaceInto(EPISODES)
            .set(record)
            .execute();
}
```

- `VIEW index_name`:

```
// generated SQL:
// select `series`.`series_id`, `series`.`title`, `series`.`series_info`, `series`.`release_date`
// from `series` view `title_name` where `series`.`title` = ?
var record = ydbDSLContext.selectFrom(SERIES.useIndex(Indexes.TITLE_NAME.name))
        .where(SERIES.TITLE.eq(title))
        .fetchOne();
```

В остальном диалект YDB соответствует документации JOOQ.

Конфигурация Spring Boot

Расширим `JooqAutoConfiguration.DslContextConfiguration` собственным `YdbDSLContext`. Например:

```
@Configuration
public class YdbJooqConfiguration extends JooqAutoConfiguration.DslContextConfiguration {

    @Override
    public YdbDSLContextImpl dslContext(org.jooq.Configuration configuration) {
        return YdbDSLContextImpl(configuration);
```

```
    }
}
```

```
spring.datasource.driver-class-name=tech.ydb.jdbc.YdbDriver
spring.datasource.url=jdbc:ydb:<schema>://<host>:<port>/path/to/database[?saFile=file:~/sa_key.json]
```

Полный пример простого приложения Spring Boot можно найти на GitHub.

Использование Dapper

Dapper - это микро ORM (Object Relational Mapping), который предоставляет простой и гибкий способ для взаимодействия с базами данных. Он работает на основе стандарта ADO.NET и предлагает множество функций, которые облегчают работу с базой данных.

Чтобы начать работу требуется дополнительная зависимость Dapper.

Рассмотрим полный пример:

```
using Dapper;
using Ydb.Sdk.Ado;

await using var connection = await new YdbDataSource().OpenConnectionAsync();

await connection.ExecuteAsync("""
                              CREATE TABLE Users(
                                  Id Int32,
                                  Name Text,
                                  Email Text,
                                  PRIMARY KEY (Id)
                              );
                              """);

await connection.ExecuteAsync("INSERT INTO Users(Id, Name, Email) VALUES (@Id, @Name, @Email)",
    new User { Id = 1, Name = "Name", Email = "Email" });

Console.WriteLine(await connection.QuerySingleAsync<User>("SELECT * FROM Users WHERE Id = @Id", new { Id = 1
}));

await connection.ExecuteAsync("DROP TABLE Users");

internal class User
{
    public int Id { get; init; }
    public string Name { get; init; } = null!;
    public string Email { get; init; } = null!;

    public override string ToString()
    {
        return $"Id: {Id}, Name: {Name}, Email: {Email}";
    }
}
```

За дополнительной информацией обратитесь к официальной документации.

Важные аспекты

Для того чтобы Dapper интерпретировал `DateTime` как YDB тип `Datetime`. Выполните следующий код:

```
SqlMapper.AddTypeMap(typeof(DateTime), DbType.DateTime);
```

По умолчанию, `DateTime` интерпретируется как `Timestamp`.

YDB Entity Framework Core Provider

YDB предоставляет поставщика Entity Framework (EF) Core — объектно-реляционный модуль сопоставления (ORM), который позволяет разработчикам .NET работать с базой данных YDB с помощью объектов .NET. Он ведёт себя так же, как и другие поставщики EF Core (например, SQL Server), поэтому здесь применима общая документация по EF Core. Если вы только начинаете работать с EF Core, эта документация — лучшее место для начала.

Разработка EF Core Provider ведётся в открытом GitHub репозитории, о всех проблемах следует сообщать там.

Настройка поставщика YDB Entity Framework Core

Чтобы начать работу, необходимо добавить необходимые пакеты NuGet в свой проект:

```
dotnet add package EntityFrameworkCore.Ydb
```

Определение моделей и DbContext

Допустим, вы хотите хранить блоги и их записи в своей базе данных; вы можете смоделировать их как типы .NET следующим образом:

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}
```

Затем вы определяете `DbContext`, который будете использовать для взаимодействия с базой данных:

OnConfiguring

Использование `OnConfiguring()` для настройки контекста — самый простой способ начать работу, но для большинства приложений в рабочей среде он не рекомендуется:

```
public class BloggingContext : DbContext
  {
  public DbSet<Blog> Blogs { get; set; }
  public DbSet<Post> Posts { get; set; }

      protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
          => optionsBuilder.UseYdb("<connection string>");
  }

  // At the point where you need to perform a database operation:
  using var context = new BloggingContext();
  // Use the context...
```

DbContext pooling

```
var dbContextFactory = new PooledDbContextFactory<BloggingContext>(
  new DbContextOptionsBuilder<BloggingContext>()
      .UseYdb("<connection string>")
      .Options);

// At the point where you need to perform a database operation:
using var context = dbContextFactory.CreateDbContext();
// Use the context...
```

ASP.NET / DI

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContextPool<BloggingContext>(opt =>
    opt.UseYdb(builder.Configuration.GetConnectionString("BloggingContext")));

public class BloggingContext(DbContextOptions<BloggingContext> options) : DbContext(options)
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
}
```

Для получения дополнительной информации о начале работы с EF обратитесь к документации по началу работы.

Дополнительные конфигурации YDB

Поставщик Entity Framework (EF) Core для YDB имеет собственные дополнительные параметры конфигурации.

Подключение ADO.NET к Yandex Cloud

Подробнее ознакомиться с различными способами аутентификации в Yandex Cloud можно в документации по ADO.NET.

Ниже приведён пример того, как передать необходимые параметры для подключения к Yandex Cloud в Entity Framework:

```
.UseYdb(cmd.ConnectionString, builder => builder
    .WithCredentialsProvider(saProvider)
    .WithServerCertificates(YcCerts.GetYcServerCertificates())
)
```

Миграция схемы

Для корректного выполнения миграций схемы базы данных необходимо отключить стратегию автоматического повтора запросов (`ExecutionStrategy`), которая по умолчанию активирована в пакете `EntityFrameworkCore.Ydb` .

Чтобы отключить `ExecutionStrategy` при выполнении миграций, следует явно переопределить интерфейс `IDesignTimeDbContextFactory` и воспользоваться методом `DisableRetryOnFailure()` .

Пример реализации фабрики контекста данных для миграций:

```
internal class BloggingContextFactory : IDesignTimeDbContextFactory<BloggingContext>
{
    public BloggingContext CreateDbContext(string[] args)
    {
        var optionsBuilder = new DbContextOptionsBuilder<BloggingContext>();

        return new BloggingContext(
            optionsBuilder.UseYdb("Host=localhost;Port=2136;Database=/local",
                builder => builder.DisableRetryOnFailure()
            ).Options
        );
    }
}
```

Примечание

Объект фабрики потребуется для выполнения команд миграции, например:

```
dotnet ef migrations add MyMigration
dotnet ef database update
```

Примеры

Полные примеры использования доступны на GitHub.

Провайдер YDB для LinqToDB

Введение

Это руководство по использованию LinqToDB с YDB.

linq2db — лёгкий и быстрый ORM для .NET, предоставляющий типобезопасные LINQ запросы и точный контроль над SQL. Провайдер YDB формирует корректный YQL, поддерживает типы YDB, генерацию схемы и массовые операции (Bulk Copy).

Установка провайдера YDB

Примеры добавления зависимостей:

dotnet CLI

```
dotnet add package Community.Ydb.Linq2db
dotnet add package linq2db
```

csproj (PackageReference)

```
<ItemGroup>
    <PackageReference Include="Community.Ydb.Linq2db" Version="$(CommunityYdbLinqToDbVersion)" />
    <PackageReference Include="linq2db" Version="$(LinqToDbVersion)" />
</ItemGroup>
```

Конфигурация провайдера

Настройте LinqToDB на использование YDB через код:

C#

```
DataConnection.AddProviderDetector(YdbTools.ProviderDetector);

// Вариант 1: локальный YDB через строку подключения
//
// Пример: локальный YDB, работающий на localhost:2136 с базой "/local"
using var localDb = new DataConnection(
    new DataOptions().UseConnectionString(
        "YDB",
        "Host=localhost;Port=2136;Database=/local;UseTls=false"
    )
);

// Вариант 2: YDB в облаке только через YdbConnectionStringBuilder
//
// Пример: подключение, собранное из явных значений host/port/database.
static async Task<DataConnection> BuildYdbDataConnection()
{
    var ydbConnectionBuilder = new YdbConnectionStringBuilder
    {
        Host     = "server",
        Port     = 2135,
        Database = "/ru-prestable/my-table",
        UseTls   = true
    };

    await using var ydbConnection = new YdbConnection(ydbConnectionBuilder);
    return YdbTools.CreateDataConnection(ydbConnection);
}
```

Использование

Используйте провайдер как и любой другой провайдер Linq To DB: сопоставляйте классы сущностей с таблицами и выполняйте запросы через `DataConnection` / `ITable<T>`. Ниже — типовая таблица соответствий и примеры генерации схемы из атрибутов.

Таблица соответствия .NET типов с типами YDB

.NET тип(ы)	Linq To DB `DataType`	Тип в YDB	Примечание
`bool`	`Boolean`	`Bool`	—
`string`	`NVarChar` / `VarChar` / `Char` / `NChar`	`Utf8`	UTF-8 строка. Text и Utf8 — одно и то же; что выводить, решает генератор DDL
`byte[]`	`VarBinary` / `Binary` / `Blob`	`String`	Бинарные данные. Bytes и String в YDB равнозначны
`Guid`	`Guid`	`Uuid`	128-битный UUID (RFC 4122). Проверка версии (v1/v4/v7) не выполняется — нужную версию генерируйте в приложении

`DateOnly` / `DateTime`	`Date`	`Date`	Хранится как дата в UTC без времени. Диапазон в YDB `1970-01-01..2106-01-01`. Диапазон .NET шире (`0001..9999`), значения вне YDB записать нельзя.
`DateTime`	`DateTime`	`Datetime`	Момент времени с точностью до секунды в UTC. `DateTime.Kind` не сохраняется; рекомендуется передавать UTC. Диапазон YDB `1970-01-01..2106-01-01`
`DateTime` / `DateTimeOffset`	`DateTime2`	`Timestamp`	Момент времени с микросекундной точностью. При записи `DateTimeOffset` смещение отбрасывается, значение конвертируется в UTC. Диапазон ограничен `1970-01-01..2106-01-01`; при попытке записи все диапазона будет ошибка
`TimeSpan`	`Interval`	`Interval`	Интервал с микросекундной точностью. Диапазон YDB меньше диапазона .NET `TimeSpan`; значения вне диапазона не поддерживаются. Доли микросекунды отбрасываются (округление вниз).
`decimal`	`Decimal`	`Decimal(p,s)`	По умолчанию — Decimal(22,9). Чтобы задать свои Precision/Scale, используйте Decimal(p,s). Пример
`float`	`Single`	`Float`	—
`double`	`Double`	`Double`	—
`sbyte` / `byte`	`SByte` / `Byte`	`Int8` / `Uint8`	—
`short` / `ushort`	`Int16` / `UInt16`	`Int16` / `Uint16`	—
`int` / `uint`	`Int32` / `UInt32`	`Int32` / `Uint32`	—
`long` / `ulong`	`Int64` / `UInt64`	`Int64` / `Uint64`	—
`string`	`Json`	`Json`	Текстовый JSON.
`byte[]`	`BinaryJson`	`JsonDocument`	Бинарный JSON.
`DateOnly` / `DateTime`	`Date`	`Date32`	Более широкий диапазон дат. Укажите DbType = "Date32". Пример
`DateTime`	`DateTime`	`Datetime64`	Точность до секунд, расширенный диапазон. Укажите DbType = "Datetime64". Пример
`DateTime` / `DateTimeOffset`	`DateTime2`	`Timestamp64`	Точность до микросекунд, расширенный диапазон. Укажите DbType = "Timestamp64". Пример
`TimeSpan`	`Interval`	`Interval64`	Более широкий диапазон интервалов. Укажите DbType = "Interval64". Пример

Совет

Точное `Precision` / `Scale` задаётся атрибутами: `[Column(DataType = DataType.Decimal, Precision = 22, Scale = 9)]`.

Примечание

По умолчанию (если на колонке не задан `DbType`) применяются базовые типы YDB: `Date`, `Datetime`, `Timestamp`, `Interval`.
Чтобы точечно включить расширенные типы, укажи `DbType`, например: `[Column(DbType = "Date32")]`. Обе семьи типов могут сосуществовать в одной таблице.

Пример кастомного Decimal

```
[Table("amounts")]
public sealed class AmountRow
{
    [PrimaryKey] public long Id { get; set; }

    // Custom precision & scale: Decimal(25,10)
    [Column("amount", DataType = DataType.Decimal, Precision = 25, Scale = 10), NotNull]
    public decimal Amount { get; set; }
}
```

Пример кастомного DbType

```
[Table("events")]
public sealed class EventRow
{
[PrimaryKey] public long Id { get; set; }
```

```
    // Extended-range timestamp
    [Column("happened_at", DbType = "Timestamp64"), NotNull]
    public DateTime HappenedAt { get; set; }

    // Extended-range date
    [Column("due_on", DbType = "Date32"), NotNull]
    public DateTime DueOn { get; set; }

    // Second precision date
    [Column("made_at", DbType = "Datetime64"), NotNull]
    public DateTime MadeAt { get; set; }

    // Extended-range interval
    [Column("duration", DbType = "Interval64"), NotNull]
    public TimeSpan Duration { get; set; }
}
```

Генерация схемы из атрибутов

Опишите сущность с помощью атрибутов Linq To DB; провайдер создаст таблицу и индексы.

C#

```
using LinqToDB.Mapping;

[Table(Name = "Groups")]
[Index("GroupName", Name = "group_name_index")]
public class Group
{
    [PrimaryKey, Column("GroupId")]
    public int Id { get; set; }

    [Column("GroupName")]
    public string? Name { get; set; }
}
```

Сгенерированный DDL (YDB):

```
CREATE TABLE Groups (
    GroupId Int32 NOT NULL,
    GroupName Utf8,
    PRIMARY KEY (GroupId)
);

ALTER TABLE Groups
  ADD INDEX group_name_index GLOBAL
       ON (GroupName);
```

Эволюция схемы (добавление поля)

Добавим поле `Department` к сущности `Group`:

C#

```
[Column] public string? Department { get; set; }
```

Изменение схемы (DDL):

```
ALTER TABLE Groups
   ADD COLUMN Department Utf8;
```

> **Примечание**
>
> LinqToDB не управляет миграциями. DDL ниже — иллюстративный; применяйте его через Liquibase/Flyway (рекомендуется). Для быстрых локальных изменений можно выполнить его напрямую через db.Execute(...) или YDB CLI.

```
using var db = new DataConnection("YDB", connectionString);

// Применяем изменение схемы (добавляем колонку):
db.Execute(@"
ALTER TABLE Groups
   ADD COLUMN Department Utf8;
");
```

Индексы YDB: как задать параметры

Через атрибут [Index] вы задаёте имя и колонки для индекса. Провайдер создаёт вторичный индекс как GLOBAL. Параметры ASYNC/SYNC и COVER(...) через атрибут не задаются — их добавляют отдельной DDL-командой после создания таблицы.

Вариант A — через атрибут (имя индекса)

```
[Table(Name = "Groups", IsColumnAttributeRequired = false)]
[Index("GroupName", Name = "group_name_index")]
public class Group
{
    [PrimaryKey, Column("GroupId")] public int Id { get; set; }
    [Column("GroupName")] public string? Name { get; set; }

    // Колонка, которую при желании можно добавить в COVER отдельной командой
    [Column] public string? Department { get; set; }
}

// При db.CreateTable<Group>() будет создан GLOBAL индекс по GroupName.
```

Сгенерированный DDL будет эквивалентен

```
CREATE TABLE Groups (
    GroupId Int32 NOT NULL,
    GroupName Utf8,
    Department Utf8,
    PRIMARY KEY (GroupId)
);

ALTER TABLE Groups
  ADD INDEX group_name_index GLOBAL
       ON (GroupName);
```

Вариант B — расширенные параметры (ASYNC, COVER) отдельной командой

```
[Table(Name = "Groups", IsColumnAttributeRequired = false)]
public class Group
{
    [PrimaryKey, Column("GroupId")] public int Id { get; set; }
    [Column("GroupName")] public string? Name { get; set; }
    [Column] public string? Department { get; set; }
}

public static class Demo
{
    public static void Main()
    {
        using var db = new DataConnection("YDB", "Host=localhost;Port=2136;Database=/local;UseTls=false");

        // 1) Создаём таблицу из атрибутов
        db.CreateTable<Group>();

        // 2) Добавляем индекс с параметрами ASYNC и COVER
        db.Execute(@"
            ALTER TABLE Groups
                ADD INDEX group_name_index GLOBAL ASYNC
                ON (GroupName)
                COVER (Department);
        ");
    }
}
```

Связи между сущностями

Опишите навигации через атрибуты `[Association]`. Пример «один-ко-многим» между `Group` и `Student`:

C#

```
[Table("Students")]
public class Student
{
    [PrimaryKey, Column("StudentId")]
    public int Id { get; set; }

    [Column("StudentName")]
    public string Name { get; set; } = null!;

    [Column]
    public int GroupId { get; set; }

    // многие-к-одному
    [Association(ThisKey = nameof(GroupId), OtherKey = nameof(Group.Id), CanBeNull = false)]
    public Group Group { get; set; } = null!;
}

[Table("Groups")]
public class Group
{
    [PrimaryKey, Column("GroupId")]
    public int Id { get; set; }

    [Column("GroupName")]
```

```
    public string? Name { get; set; }

    // один-ко-многим
    [Association(ThisKey = nameof(Id), OtherKey = nameof(Student.GroupId))]
    public IEnumerable<Student> Students { get; set; } = null!;
}
```

Создаваемые таблицы:

```
CREATE TABLE Groups (
    GroupId Int32 NOT NULL,
    GroupName Utf8,
    PRIMARY KEY (GroupId)
);

CREATE TABLE Students (
    StudentId Int32 NOT NULL,
    StudentName Utf8,
    GroupId Int32,
    PRIMARY KEY (StudentId)
);
```

Примеры сгенерированного YQL для выборок по связям

“Lazy” loading (two queries)

```
  // 1) SELECT g.GroupId, g.GroupName FROM Groups AS g WHERE g.GroupName = 'M3439';
  var grp = db.GetTable<Group>()
  .Where(g => g.Name == "M3439")
  .Select(g => new { g.Id, g.Name })
  .FirstOrDefault();

  // 2) SELECT s.StudentId, s.StudentName, s.GroupId FROM Students AS s WHERE s.GroupId = ?;
  var students = grp == null
  ? new List<Student>()
  : db.GetTable<Student>()
  .Where(s => s.GroupId == grp.Id)
  .Select(s => new { s.Id, s.Name, s.GroupId })
  .ToList();
```

“Eager” loading (JOIN)

```
  var joined =
  (from g in db.GetTable<Group>()
  join s in db.GetTable<Student>() on g.Id equals s.GroupId
  where g.Name == "M3439"
  select new { GroupId = g.Id, GroupName = g.Name, StudentId = s.Id, StudentName = s.Name, s.GroupId })
  .ToList();
```

«Ленивая» загрузка (две выборки)

```
SELECT
    g.GroupId,
    g.GroupName
FROM
    Groups AS g
WHERE
    g.GroupName = 'M3439';

SELECT
    s.StudentId,
    s.StudentName,
    s.GroupId
FROM
    Students AS s
WHERE
    s.GroupId = ?;
```

«Жадная» загрузка (JOIN)

```
SELECT
    g.GroupId,
    g.GroupName,
    s.StudentId,
    s.StudentName,
    s.GroupId
FROM
    Groups AS g
JOIN
    Students AS s
  ON g.GroupId = s.GroupId
WHERE
    g.GroupName = 'M3439';
```

Пример «прикладной» сущности и генерируемого DDL

Этот раздел показывает практическую сущность. Он демонстрирует:

- реалистичный набор колонок (ФИО, e-mail, дата найма, зарплата, флаги, целочисленные значения);
- точные типы (например, Decimal(22,9) для денежных значений, Date для дат, Utf8/Bool/Int32/Int64);
- GLOBAL вторичный индекс по full_name для быстрых поисков и сортировки;
- сквозной сценарий: маппинг → создание таблицы → CRUD с сгенерированным YQL/DDL.

При вызове `db.CreateTable<Employee>()` Linq To DB создаёт таблицу и применяет атрибут [Index] как GLOBAL-индекс YDB.

C#

```
using System;
using LinqToDB.Mapping;

[Table("employee")]
[Index("full_name", Name = "employee_full_name_idx")]
public class Employee
{
    [PrimaryKey] public long Id { get; set; }

    [Column("full_name"), NotNull] public string FullName { get; set; } = null!;

    [Column, NotNull] public string Email { get; set; } = null!;

    [Column("hire_date", DataType = DataType.Date), NotNull]
    public DateTime HireDate { get; set; }

    [Column(DataType = DataType.Decimal, Precision = 22, Scale = 9), NotNull]
    public decimal Salary { get; set; }

    [Column("is_active"), NotNull] public bool IsActive { get; set; }

    [Column, NotNull] public string Department { get; set; } = null!;

    [Column, NotNull] public int Age { get; set; }
}
```

Сгенерированный DDL:

```
CREATE TABLE employee (
    Id Int64 NOT NULL,
    full_name Utf8 NOT NULL,
    Email Utf8 NOT NULL,
    hire_date Date NOT NULL,
    Salary Decimal(22,9) NOT NULL,
    is_active Bool NOT NULL,
    Department Utf8 NOT NULL,
    Age Int32 NOT NULL,
    PRIMARY KEY (Id)
);

ALTER TABLE employee
  ADD INDEX employee_full_name_idx GLOBAL
       ON (full_name);
```

Пример использования:

```
using System;
using System.Linq;
using LinqToDB;
using LinqToDB.Data;

// Этот пример использует прямую строку подключения и предназначен для локального / тестового YDB.
// Если вам нужно подключиться к YDB в Yandex Cloud,
// используйте YdbConnectionStringBuilder и YdbConnection (см. раздел про конфигурацию провайдера).

using var db = new DataConnection(
    new DataOptions().UseConnectionString(
        "YDB",
        "Host=localhost;Port=2136;Database=/local;UseTls=false"
    )
);

// INSERT
var employee = new Employee
{
    Id         = 1L,
    FullName   = "Example",
    Email      = "example@example.com",
    HireDate   = new DateTime(2023, 12, 20),
    Salary     = 500000.000000000m,
    IsActive   = true,
    Department = "YDB AppTeam",
    Age        = 23,
};
db.Insert(employee);
```

```
// SELECT по первичному ключу
var loaded = db.GetTable<Employee>()
              .FirstOrDefault(e => e.Id == employee.Id);

// Обновим Email/Department/Salary у сотрудника с заданным Id
db.GetTable<Employee>()
  .Where(e => e.Id == employee.Id)
  .Set(e => e.Email,      "example+updated@example.com")
  .Set(e => e.Department, "Analytics")
  .Set(e => e.Salary,     550000.000000000m)
  .Update();

// DELETE по первичному ключу
db.GetTable<Employee>()
  .Where(e => e.Id == employee.Id)
  .Delete();
```

Примеры YQL, формируемые провайдером для простых операций:

- Вставка одной записи

```
INSERT INTO employee (Age,Department,Email,full_name,hire_date,is_active,Salary,Id)
VALUES (?,?,?,?,?,?,?,?);
```

- Чтение по первичному ключу

```
SELECT
    e.Id, e.full_name, e.Email, e.hire_date, e.Salary, e.is_active, e.Department, e.Age
FROM employee AS e
WHERE e.Id = ?;
```

- Обновление по первичному ключу

```
UPDATE employee
SET
    Email      = ?,
    Department = ?,
    Salary     = ?
WHERE Id = ?;
```

> **Примечание**
>
> Провайдер выводит параметры (?), потому что значения и типы привязываются драйвером, а не объявляются в тексте запроса. Когда YQL требует типизированные параметры, провайдер автоматически добавляет соответствующие DECLARE. Для “нестандартных” паттернов вроде upsert-записей используйте UPSERT с параметризацией — провайдер генерирует такие выражения? как обычный YQL с параметрами.

- Удаление по первичному ключу

```
DELETE FROM employee WHERE Id = ?;
```

Массовые операции: вставка, обновление и удаление

Массовая вставка (BulkCopy)

```
var now  = DateTime.UtcNow;
var data = Enumerable.Range(0, 15_000).Select(i => new SimpleEntity
{
    Id      = i,
    IntVal  = i,
    DecVal  = 0m,
    StrVal  = $"Name {i}",
    BoolVal = (i & 1) == 0,
    DtVal   = now,
});
```

Массовое обновление (WHERE IN)

C#

```
var ids = Enumerable.Range(0, 15_000).ToArray();

table.Where(t => ids.Contains(t.Id))
    .Set(_ => _.DecVal, 1.23m)
    .Set(_ => _.StrVal, "updated")
    .Set(_ => _.BoolVal, true)
    .Update();
```

YQL

```
DECLARE $Gen_List_Primitive_1 AS List<Int32>;
UPDATE
    SimpleEntity
```

```
SET
    DecVal = Decimal('1.23', 22, 9),
    StrVal = 'updated'u,
    BoolVal = true
WHERE
    SimpleEntity.Id IN $Gen_List_Primitive_1
```

Массовое удаление (WHERE IN)

C#

```
table.Delete(t => ids.Contains(t.Id));
```

YQL

```
DECLARE $Gen_List_Primitive_1 AS List<Int32>;
DELETE FROM
    SimpleEntity
WHERE
    SimpleEntity.Id IN $Gen_List_Primitive_1
```

SQLAlchemy

SQLAlchemy — это популярная Python-библиотека для работы с базами данных, предоставляющая как ORM (Object-Relational Mapping), так и Core API для выполнения SQL-запросов.

YDB поддерживает интеграцию с SQLAlchemy через специальный диалект `ydb-sqlalchemy`, который обеспечивает полную совместимость с SQLAlchemy 2.0 и частичную поддержку SQLAlchemy 1.4.

Установка

Установите пакет `ydb-sqlalchemy` с помощью pip:

```
pip install ydb-sqlalchemy
```

Быстрый старт

Строка подключения

Для подключения к YDB через SQLAlchemy используйте следующую строку подключения:

```
yql+ydb://<cluster_url>:<port><db_path>
```

Параметры строки подключения:

- `<cluster_url>` — адрес сервера YDB (например, `localhost`)
- `<port>` — порт gRPC-сервиса (по умолчанию `2136`)
- `<db_path>` — абсолютный путь к базе данных (например, `/local` или `/Root/db`)

Пример строки подключения:

```
yql+ydb://localhost:2136/local
```

Подключение

Синхронное подключение

```
import sqlalchemy as sa

# Пример настройки подключения к YDB через SQLAlchemy (синхронный режим)
engine = sa.create_engine("yql+ydb://localhost:2136/local")

# Выполнение простого запроса
with engine.connect() as conn:
    result = conn.execute(sa.text("SELECT 1 AS value"))
    print(result.fetchone())
```

Асинхронное подключение

```
import asyncio
import sqlalchemy as sa
from sqlalchemy.ext.asyncio import create_async_engine

async def main():
    # Создание асинхронного `engine`
    engine = create_async_engine("yql+ydb_async://localhost:2136/local")

    # Выполнение запроса
    async with engine.connect() as conn:
        result = await conn.execute(sa.text("SELECT 1 AS value"))
        print(result.fetchone())

asyncio.run(main())
```

Конфигурация подключения

Методы аутентификации

Анонимная

Для локальной разработки или тестирования:

```
import sqlalchemy as sa

engine = sa.create_engine("yql+ydb://localhost:2136/local")
```

С помощью логина и пароля

Использование имени пользователя и пароля:

```
engine = sa.create_engine(
    "yql+ydb://localhost:2136/local",
    connect_args={
        "credentials": {
            "username": "your_username",
```

```
            "password": "your_password"
        }
    }
)
```

С помощью токена

Использование токена доступа:

```
engine = sa.create_engine(
    "yql+ydb://localhost:2136/local",
    connect_args={
        "credentials": {
            "token": "your_access_token"
        }
    }
)
```

С помощью сервисного аккаунта

Использование JSON-ключа сервисного аккаунта:

```
import json

# Загрузка из файла
with open('service_account_key.json', 'r') as f:
    service_account_json = json.load(f)

engine = sa.create_engine(
    "yql+ydb://localhost:2136/local",
    connect_args={
        "credentials": {
            "service_account_json": service_account_json
        }
    }
)

# Или передача JSON напрямую
engine = sa.create_engine(
    "yql+ydb://localhost:2136/local",
    connect_args={
        "credentials": {
            "service_account_json": {
                "id": "your_key_id",
                "service_account_id": "your_service_account_id",
                "created_at": "2023-01-01T00:00:00Z",
                "key_algorithm": "RSA_2048",
                "public_key": "-----BEGIN PUBLIC KEY-----\n...\n-----END PUBLIC KEY-----",
                "private_key": "-----BEGIN PRIVATE KEY-----\n...\n-----END PRIVATE KEY-----"
            }
        }
    }
)
```

С помощью SDK YDB

Можно использовать любые доступные методы аутентификации из Python SDK для YDB:

```
import ydb.iam

# Метаданные сервиса
engine = sa.create_engine(
    "yql+ydb://localhost:2136/local",
    connect_args={
        "credentials": ydb.iam.MetadataUrlCredentials()
    }
)

# OAuth-токен
engine = sa.create_engine(
    "yql+ydb://localhost:2136/local",
    connect_args={
        "credentials": ydb.iam.OAuthCredentials("your_oauth_token")
    }
)

# Статические учетные данные
engine = sa.create_engine(
    "yql+ydb://localhost:2136/local",
    connect_args={
        "credentials": ydb.iam.StaticCredentials("username", "password")
    }
)
```

Использование шифрования

Если у вас развёрнут кластер YDB с использованием шифрования, необходимо добавить в настройки подключения блок с указанием протокола и пути к сертификату.
Это обеспечит защищённое соединение с кластером YDB.

Например:

```
engine = sa.create_engine(
    "yql+ydb://ydb.example.com:2135/prod",
    connect_args={
        "credentials": {"token": "your_token"},
        "protocol": "grpcs",
        "root_certificates_path": "/path/to/ca-certificates.crt",
        # "root_certificates": crt_string,  # Альтернативно — строка с сертификатами
    }
)
```

Поддерживаемые типы данных

YDB SQLAlchemy предоставляет полную поддержку типов данных YDB через пользовательские типы SQLAlchemy. Для получения подробной информации о системе типов YDB см. документацию по типам данных YDB.

Сводная таблица типов

YDB тип	YDB SQLAlchemy тип	Стандартный SQLAlchemy тип	Python тип	Примечания
`Bool`	`Boolean`	`Boolean`	`bool`	
`Int8`	`Int8`		`int`	-2^7 до 2^7-1
`Int16`	`Int16`		`int`	-2^15 до 2^15-1
`Int32`	`Int32`		`int`	-2^31 до 2^31-1
`Int64`	`Int64`	`Integer`	`int`	-2^63 до 2^63-1
`Uint8`	`UInt8`		`int`	0 до 2^8-1
`Uint16`	`UInt16`		`int`	0 до 2^16-1
`Uint32`	`UInt32`		`int`	0 до 2^32-1
`Uint64`	`UInt64`		`int`	0 до 2^64-1
`Float`	`Float`	`Float`	`float`	
`Double`	`Double`		`float`	Доступно в SQLAlchemy 2.0+
`Decimal(p,s)`	`Decimal`	`DECIMAL`	`decimal.Decimal`	
`String`		`BINARY`	`bytes`	
`Utf8`		`String/Text`	`str`	
`Date`	`YqlDate`	`Date`	`datetime.date`	
`Date32`	`YqlDate32`		`datetime.date`	Расширенный диапазон дат
`Datetime`	`YqlDateTime`	`DATETIME`	`datetime.datetime`	
`Datetime64`	`YqlDateTime64`		`datetime.datetime`	Расширенный диапазон
`Timestamp`	`YqlTimestamp`	`TIMESTAMP`	`datetime.datetime`	
`Timestamp64`	`YqlTimestamp64`		`datetime.datetime`	Расширенный диапазон
`Json`	`YqlJSON`	`JSON`	`dict/list`	
`List<T>`	`ListType`	`ARRAY`	`list`	
`Struct<...>`	`StructType`		`dict`	
`Optional<T>`	`nullable=True`		`None + базовый тип`	

Миграции с Alembic

Конфигурация, специфичная для YDB

YDB требует специальной конфигурации в `env.py` из-за своих уникальных характеристик:

```
# migrations/env.py
from logging.config import fileConfig
import sqlalchemy as sa
from sqlalchemy import engine_from_config, pool
from alembic import context
from alembic.ddl.impl import DefaultImpl

# Импортируйте ваши модели
from myapp.models import Base

config = context.config
```

```
if config.config_file_name is not None:
    fileConfig(config.config_file_name)

target_metadata = Base.metadata

# YDB-специфичная реализация
class YDBImpl(DefaultImpl):
    __dialect__ = "yql"

def run_migrations_offline() -> None:
    """Запуск миграций в 'offline' режиме."""
    url = config.get_main_option("sqlalchemy.url")
    context.configure(
        url=url,
        target_metadata=target_metadata,
        literal_binds=True,
        dialect_opts={"paramstyle": "named"},
    )

    with context.begin_transaction():
        context.run_migrations()

def run_migrations_online() -> None:
    """Запуск миграций в 'online' режиме."""
    connectable = engine_from_config(
        config.get_section(config.config_ini_section, {}),
        prefix="sqlalchemy.",
        poolclass=pool.NullPool,
    )

    with connectable.connect() as connection:
        context.configure(
            connection=connection,
            target_metadata=target_metadata
        )

        with context.begin_transaction():
            context.run_migrations()

if context.is_offline_mode():
    run_migrations_offline()
else:
    run_migrations_online()
```

Полезные ссылки

- Примеры использования на GitHub
- PyPI пакет

Django

Django — это популярный Python веб-фреймворк с мощной ORM для работы с базами данных.

YDB поддерживает интеграцию с Django через специальный бэкенд `django-ydb-backend`, который обеспечивает полную поддержку Django ORM для работы с YDB.

> **Внимание**
>
> Интеграция с Django пока находится на этапе бета-тестирования. Обратите внимание, что сейчас поля ForeignKey, ManyToManyField и OneToOneField ещё не поддерживаются — их реализация находится в процессе разработки.

Установка

Установите пакет django-ydb-backend с помощью pip:

```
pip install django-ydb-backend
```

Конфигурация

Настройки подключения

Для подключения к YDB используйте стандартный механизм настройки баз данных Django через переменную `DATABASES` в файле `settings.py`. Подробнее о конфигурации баз данных в Django можно прочитать в официальной документации.

Необходимые параметры конфигурации подключения:

- `NAME` — имя базы данных Django.
- `ENGINE` — используется значение `ydb_backend.backend`.
- `HOST` — адрес сервера YDB (например, `localhost`).
- `PORT` — порт gRPC YDB (по умолчанию 2136).
- `DATABASE` — абсолютный путь к базе данных YDB (например, `/local` для локальных тестов или `/my_production_db` для production).

Пример настройки подключения:

```
DATABASES = {
    "default": {
        "NAME": "ydb_db",
        "ENGINE": "ydb_backend.backend",
        "HOST": "localhost",
        "PORT": "2136",
        "DATABASE": "/local",
    }
}
```

Методы аутентификации

Анонимная

Для использования анонимной аутентификации не нужно передавать дополнительные параметры.

С помощью логина и пароля

Для использования статических учетных данных нужно указать `username` и `password`:

```
DATABASES = {
    "default": {
        "ENGINE": "ydb_backend.backend",
        "CREDENTIALS": {
            "username": "...",
            "password": "..."
        }
    }
}
```

С помощью токена

Для использования токена доступа нужно указать `token`:

```
DATABASES = {
    "default": {
        "ENGINE": "ydb_backend.backend",
        "CREDENTIALS": {
            "token": "..."
        },
    }
}
```

С помощью сервисного аккаунта

Для использования сервисного аккаунта нужно указать `service_account_json`:

```
DATABASES = {
    "default": {
        "ENGINE": "ydb_backend.backend",
        "CREDENTIALS": {
            "service_account_json": {
                "id": "...",
                "service_account_id": "...",
                "created_at": "...",
                "key_algorithm": "...",
                "public_key": "...",
                "private_key": "..."
            }
        }
    }
}
```

Использование шифрования

Если у вас развернут кластер YDB с использованием шифрования, необходимо добавить в настройки базы данных блок `OPTIONS` с указанием протокола и пути к сертификату. Например:

Это обеспечит защищённое соединение с кластером YDB.

```
DATABASES = {
    "default": {
        ...
        "OPTIONS": {
            "protocol": "grpcs",
            "root_certificates_path": "path/to/cert.crt",
        }
    }
}
```

Поля Django

YDB backend поддерживает встроенные поля Django.

Поддерживаемые поля Django

Класс	YDB тип	Python тип	Комментарии
`SmallAutoField`	`Int16`	`int`	YDB тип SmallSerial будет генерировать значения автоматически. Диапазон -32768 до 32767
`AutoField`	`Int32`	`int`	YDB тип Serial будет генерировать значения автоматически. Диапазон -2147483648 до 2147483647
`BigAutoField`	`Int64`	`int`	YDB тип BigSerial будет генерировать значения автоматически. Диапазон -9223372036854775808 до 9223372036854775807
`CharField`	`UTF-8`	`str`	
`TextField`	`UTF-8`	`str`	
`BinaryField`	`String`	`bytes`	
`SlugField`	`UTF-8`	`str`	
`FileField`	`String`	`bytes`	
`FilePathField`	`UTF-8`	`str`	
`DateField`	`Date`	`datetime.date`	Диапазон значений: с 00:00 01.01.1970 до 00:00 01.01.2106. Внутреннее представление: беззнаковое 16-битное целое
`DateTimeField`	`DateTime`	`datetime.datetime`	Внутреннее представление: беззнаковое 32-битное целое
`DurationField`	`Interval`	`int`	Диапазон значений от -136 лет до +136 лет. Внутреннее представление: знаковое 64-битное целое. Нельзя использовать в первичном ключе
`SmallIntegerField`	`Int16`	`int`	Диапазон -32768 до 32767
`IntegerField`	`Int32`	`int`	Диапазон -2147483648 до 2147483647
`BigIntegerField`	`Int64`	`int`	Диапазон -9223372036854775808 до 9223372036854775807
`PositiveSmallIntegerField`	`UInt16`	`int`	Диапазон 0 до 65535
`PositiveIntegerField`	`UInt32`	`int`	Диапазон 0 до 4294967295
`PositiveBigIntegerField`	`UInt64`	`int`	Диапазон 0 до 18446744073709551615

`FloatField`	`Float`	`float`	Вещественное число с переменной точностью, размер 4 байта. Нельзя использовать в первичном ключе
`DecimalField`	`Decimal`	`Decimal`	Python значения округляются до масштаба поля базы данных. Поддерживает только Decimal(22,9)
`UUIDField`	`UUID`	`uuid.UUID`	
`IPAddressField`	`UTF-8`	`str`	
`GenericIPAddressField`	`UTF-8`	`str`	
`BooleanField`	`Bool`	`boolean`	
`EmailField`	`UTF-8`	`str`	

Быстрый старт

Необходимые условия

Перед началом работы убедитесь, что:

1. Развернут кластер YDB локально или используя Ansible
2. Установлены Python-пакеты: `Django`, `django-ydb-backend`, `djangorestframework`
3. Склонирован репозиторий с примером

Перейдите в рабочую директорию

```
cd examples/bookstore
```

Конфигурация подключения к YDB

Для интеграции YDB с Django укажите параметры подключения в файле `config/settings.py` проекта:

```
DATABASES = {
    "default": {
        "NAME": "ydb_db",
        "ENGINE": "ydb_backend.backend",
        "HOST": "localhost",
        "PORT": "2136",
        "DATABASE": "/local",
        "OPTIONS": {
            "protocol": "grpcs",
            "root_certificates_path": "path/to/cert.crt",
        }
    }
}
```

Запустите миграции

```
python manage.py migrate
```

Запустите сервер

```
python manage.py runserver
```

Проверьте работу приложения

После запуска приложение будет доступно по адресу `http://127.0.0.1:8000` (по умолчанию).

Полезные ссылки

- Примеры использования на GitHub
- PyPI пакет

Диалект YDB для Kotlin Exposed

Введение

Это руководство описывает использование JetBrains Exposed с YDB.

Exposed - это Kotlin-библиотека для работы с базой данных через SQL DSL и DAO API поверх JDBC. Диалект YDB добавляет в Exposed поддержку YDB-совместимого SQL, отображение типов данных YDB, retry-aware транзакции для optimistic concurrency control и YDB-специфичные особенности схемы.

Руководство охватывает два практических сценария:

- основной JDBC-диалект для Exposed;
- пример собственной конфигурации для приложений на Spring Boot 3.

Установка диалекта YDB

Примеры для различных систем сборки:

Maven

```
<!-- Укажите актуальные версии -->
<dependency>
    <groupId>tech.ydb.jdbc</groupId>
    <artifactId>ydb-jdbc-driver</artifactId>
    <version>${ydb.jdbc.version}</version>
</dependency>

<dependency>
    <groupId>tech.ydb.dialects</groupId>
    <artifactId>kotlin-exposed-ydb-dialect</artifactId>
    <version>${exposed.ydb.dialect.version}</version>
</dependency>
```

Gradle

```
dependencies {
    // Укажите актуальные версии
    implementation "tech.ydb.jdbc:ydb-jdbc-driver:$ydbJdbcVersion"
    implementation "tech.ydb.dialects:kotlin-exposed-ydb-dialect:$ydbDialectVersion"
}
```

Актуальные версии артефактов:

- `tech.ydb.dialects:kotlin-exposed-ydb-dialect` на Maven Central;
- `tech.ydb.jdbc:ydb-jdbc-driver` — там же.

Если приложение использует DAO, JSON DSL или другие модули Exposed, подключайте их стандартным способом вместе с основным диалектом.

Подключение и конфигурация

Сначала зарегистрируйте диалект и драйвер, затем откройте `Database` через обычный `Database.connect(...)` и передайте рекомендуемые значения `DatabaseConfig`:

```
import org.jetbrains.exposed.v1.core.DatabaseConfig
import org.jetbrains.exposed.v1.jdbc.Database
import tech.ydb.exposed.dialect.registerYdbDialect
import java.sql.Connection

registerYdbDialect()

val db = Database.connect(
    url = "jdbc:ydb:grpc://localhost:2136/local",
    driver = "tech.ydb.jdbc.YdbDriver",
    databaseConfig = DatabaseConfig {
        useNestedTransactions = false
    }
)
```

`registerYdbDialect()`:

- регистрирует JDBC-драйвер `tech.ydb.jdbc.YdbDriver` для URL с префиксом `jdbc:ydb:`;
- регистрирует `YdbDialect` в Exposed;
- регистрирует metadata bridge для чтения существующих secondary indexes через JDBC metadata.

Рекомендуемая конфигурация Exposed для YDB включает:

- `useNestedTransactions = false` — это настройка на стороне Exposed, она запрещает Exposed эмулировать вложенные транзакции. YDB вложенные транзакции не поддерживает, поэтому такая эмуляция приводила бы к ошибкам;
- регистрацию `YdbDialect` через `registerYdbDialect(...)` до открытия `Database`.

Особенности DDL в YDB

Базовое использование Exposed — определение таблиц, DSL/DAO-запросы, транзакции и т.д. — описано в официальной документации JetBrains Exposed. В этом разделе собраны только YDB-специфичные особенности генерации DDL диалектом.

Создание таблиц

В YDB `CREATE TABLE` должен содержать табличный `PRIMARY KEY (...)`. Стандартный DDL-путь Exposed 1.3.0 для таблиц с одноколоночным PK может сгенерировать inline-форму:

```
id Int32 PRIMARY KEY
```

YDB такую форму не принимает. Поэтому для таблиц, которые должны создаваться через Exposed DDL, необходимо переопределять `createStatement()` и вызывать `createYdbStatement()`:

```
import org.jetbrains.exposed.v1.core.PrimaryKey
import org.jetbrains.exposed.v1.core.Table
import tech.ydb.exposed.dialect.createYdbStatement
import tech.ydb.exposed.dialect.javatime.ydbTimestamp64
import tech.ydb.exposed.dialect.ydbDecimal

object Products : Table("products") {
    val id = integer("id")
    val sku = varchar("sku", 64)
    val name = varchar("name", 255)
    val category = varchar("category", 128)
    val price = ydbDecimal("price", precision = 10, scale = 2)
    val expiresAt = ydbTimestamp64("expires_at")

    override val primaryKey = PrimaryKey(id)

    init {
        index(false, sku)
    }

    override fun createStatement(): List<String> = createYdbStatement()
}
```

`createYdbStatement()`:

- рендерит все колонки без inline `PRIMARY KEY`;
- добавляет табличный `PRIMARY KEY (...)`;
- сохраняет `NOT NULL` и `DEFAULT`;
- сохраняет `storageParameters`, поэтому YDB-специфичные `WITH (...)` можно описывать через стандартный Exposed API.

Создание схемы выполняется обычными средствами Exposed:

```
import org.jetbrains.exposed.v1.jdbc.SchemaUtils
import tech.ydb.exposed.dialect.ydbTransaction

ydbTransaction(db) {
    SchemaUtils.create(Products)
}
```

YDB-специфичные параметры таблицы

TTL и другие YDB-специфичные table options задаются через `storageParameters`:

```
import org.jetbrains.exposed.v1.core.PrimaryKey
import org.jetbrains.exposed.v1.core.RawTableStorageParameter
import org.jetbrains.exposed.v1.core.Table
import org.jetbrains.exposed.v1.core.TableStorageParameter
import tech.ydb.exposed.dialect.createYdbStatement
import tech.ydb.exposed.dialect.javatime.ydbTimestamp64

object Sessions : Table("sessions") {
    val id = integer("id")
    val expireAt = ydbTimestamp64("expire_at")

    override val primaryKey = PrimaryKey(id)

    override val storageParameters: List<TableStorageParameter> =
        listOf(RawTableStorageParameter("TTL = Interval(\"PT1H\") ON expire_at"))

    override fun createStatement(): List<String> = createYdbStatement()
}
```

Secondary indexes, объявленные через стандартный `Table.index(...)`, создаются отдельно через:

```
ALTER TABLE ... ADD INDEX ... GLOBAL
```

Произвольный YQL через `createYdbStatement()`

`createYdbStatement()` возвращает `List<String>`, поэтому к сгенерированному DDL можно добавить произвольные YQL-операторы — например, `CREATE TABLE` колоночной таблицы, `ALTER TABLE ADD INDEX ... LOCAL` или любой другой DDL, который диалект не покрывает first-class.

```
import org.jetbrains.exposed.v1.core.PrimaryKey
import org.jetbrains.exposed.v1.core.Table
import tech.ydb.exposed.dialect.createYdbStatement

object EventsColumn : Table("events_column") {
    val id = integer("id")
    val name = varchar("name", 255)

    override val primaryKey = PrimaryKey(id)

    override fun createStatement(): List<String> = createYdbStatement() + listOf(
        """
        CREATE TABLE events_column_olap (
            id Int32 NOT NULL,
            payload Json,
            PRIMARY KEY (id)
        ) WITH (STORE = COLUMN)
        """.trimIndent()
    )
}
```

Это удобный способ описать таблицы, для которых Exposed DSL не покрывает нужный YDB-синтаксис. Для управления схемой в целом рекомендуется использовать Flyway или Liquibase.

Операции чтения и записи

Стандартные DSL-операции Exposed (`select`, `insert`, `update`, `deleteWhere`) работают без дополнительных изменений.

Диалект также использует native-операции YDB `UPSERT` и `REPLACE` через расширения `Table.upsert(...)`, `Table.batchUpsert(...)` и `Table.replace(...)`:

```
import java.math.BigDecimal

Products.upsert {
    it[id] = 1
    it[sku] = "BOOK-001"
    it[name] = "Kotlin in Action"
    it[category] = "books"
    it[price] = BigDecimal("39.90")
}

Products.replace {
    it[id] = 1
    it[sku] = "BOOK-001"
    it[name] = "Kotlin in Action, 2nd edition"
    it[category] = "books"
    it[price] = BigDecimal("44.90")
}
```

Семантика соответствует YQL-операторам `UPSERT INTO` и `REPLACE INTO`.
Exposed-специфичные параметры `onUpdate` и `keyColumns` для `upsert(...)` игнорируются, а `upsert(where)` и PostgreSQL-подобная логика `ON CONFLICT DO UPDATE` не поддерживаются.

> **Важно**
>
> ANSI `MERGE` диалект намеренно не поддерживает. В сценариях YDB его роль выполняют `UPSERT` и `REPLACE`.

Повторяемые транзакции

YDB использует optimistic concurrency control, поэтому транзакция может завершиться retryable-ошибкой и потребовать повторного выполнения. Для этого диалект предоставляет `ydbTransaction(...)`:

```
import tech.ydb.exposed.dialect.YdbRetryConfig
import tech.ydb.exposed.dialect.ydbTransaction

ydbTransaction(db) {
    // обычная read-write транзакция
}

ydbTransaction(db, retry = YdbRetryConfig.IDEMPOTENT) {
    // безопасно повторяемая операция
}

ydbTransaction(db, readOnly = true, retry = YdbRetryConfig.IDEMPOTENT) {
    // read-only сценарий
}
```

`YdbRetryConfig.IDEMPOTENT` следует использовать только тогда, когда тело транзакции можно безопасно выполнить повторно.

Типы данных

Таблица отображения стандартных Exposed-типов в типы YDB:

Exposed	YDB

`byte` / `ubyte`	`Int8` / `Uint8`
`short` / `ushort`	`Int16` / `Uint16`
`integer` / `uinteger`	`Int32` / `Uint32`
`long`	`Int64`
`float` / `double`	`Float` / `Double`
`bool`	`Bool`
`varchar` / `text`	`Text`
`binary` / `blob`	`Bytes`
`uuid`	`Uuid`
`date`	`Date` или `Date32`
`datetime`	`Datetime` или `Datetime64`
`timestamp`	`Timestamp` или `Timestamp64`
`json`	`Json`
`jsonb`	`JsonDocument`

Особенности:

- `varchar(n)` отображается в `Text`; ограничение длины не кодируется в YDB DDL;
- `jsonb` соответствует `JsonDocument` — это бинарное представление JSON, оптимизированное для эффективного извлечения через `JSON_VALUE` / `JSON_EXISTS` / `JSON_QUERY`. Сам `JsonDocument` нельзя использовать как ключ B-Tree-индекса (в YDB JSON не является comparable-типом);
- `autoIncrement()` отображается в `Serial` или `BigSerial`.

YDB-специфичные типы

Диалект добавляет набор расширений для колонок:

```
import tech.ydb.exposed.dialect.ydbDecimal
import tech.ydb.exposed.dialect.ydbInterval
import tech.ydb.exposed.dialect.ydbInterval64
import tech.ydb.exposed.dialect.ydbJson
import tech.ydb.exposed.dialect.ydbJsonDocument
import tech.ydb.exposed.dialect.ydbUbyte
import tech.ydb.exposed.dialect.ydbUint32
import tech.ydb.exposed.dialect.ydbUint64
import tech.ydb.exposed.dialect.ydbUshort
import tech.ydb.exposed.dialect.ydbUlong
import tech.ydb.exposed.dialect.ydbUuid

val amount = ydbDecimal("amount", precision = 22, scale = 9)
val payload = ydbJson("payload")
val indexedPayload = ydbJsonDocument("indexed_payload")
val uid = ydbUuid("uid")
val flags = ydbUbyte("flags")
val counter = ydbUint32("counter")
val total = ydbUint64("total")
val someField = ydbUshort("some_field")
val totalFullRange = ydbUlong("total_full_range")
val duration = ydbInterval("duration")
val duration64 = ydbInterval64("duration64")
```

Соответствие YDB-типов и Kotlin-типов рантайма:

Функция	YDB-тип	Kotlin-тип
`ydbDecimal(name, p, s)`	`Decimal(p, s)`	`java.math.BigDecimal`
`ydbInterval(name)`	`Interval`	`java.time.Duration`
`ydbInterval64(name)`	`Interval64`	`java.time.Duration`
`ydbJson(name)`	`Json`	`String`
`ydbJsonDocument(name)`	`JsonDocument`	`String`
`ydbUuid(name)`	`Uuid`	`java.util.UUID`
`ydbUbyte(name)`	`Uint8`	`kotlin.UByte`
`ydbUshort(name)`	`Uint16`	`kotlin.UShort`
`ydbUint32(name)`	`Uint32`	`kotlin.UInt`
`ydbUint64(name)`	`Uint64`	`kotlin.Long` (`0..Long.MAX_VALUE`)
`ydbUlong(name)`	`Uint64`	`kotlin.ULong`

Для literal-значений `Decimal` в выражениях обновления доступно `ydbDecimalLiteral(...)`.

Явный выбор temporal-типа на уровне колонки

```
import org.jetbrains.exposed.v1.core.PrimaryKey
import org.jetbrains.exposed.v1.core.Table
import tech.ydb.exposed.dialect.createYdbStatement
import tech.ydb.exposed.dialect.javatime.ydbDate
import tech.ydb.exposed.dialect.javatime.ydbDate32
import tech.ydb.exposed.dialect.javatime.ydbDatetime
import tech.ydb.exposed.dialect.javatime.ydbDatetime64
import tech.ydb.exposed.dialect.javatime.ydbTimestamp
import tech.ydb.exposed.dialect.javatime.ydbTimestamp64

object Events : Table("events") {
    val id = integer("id")
    val legacyDate = ydbDate("legacy_date")
    val signedDate = ydbDate32("signed_date")
    val legacyDatetime = ydbDatetime("legacy_datetime")
    val signedDatetime = ydbDatetime64("signed_datetime")
    val legacyTimestamp = ydbTimestamp("legacy_timestamp")
    val signedTimestamp = ydbTimestamp64("signed_timestamp")

    override val primaryKey = PrimaryKey(id)
    override fun createStatement(): List<String> = createYdbStatement()
}
```

Примечание

`registerYdbDialect(enableSignedDatetimes = true)` меняет DDL-названия только для стандартных Exposed `date`, `datetime` и `timestamp`. Явные `ydbDate(...)`, `ydbDate32(...)`, `ydbDatetime(...)`, `ydbDatetime64(...)`, `ydbTimestamp(...)` и `ydbTimestamp64(...)` всегда генерируют тот тип, который отражён в названии функции.

Важно

`ydbUint64(...)` хранит значение в Kotlin `Long`, поэтому поддерживается только диапазон `0..Long.MAX_VALUE`. Для значений выше `Long.MAX_VALUE` используйте `ydbUlong(...)` — он представлен Kotlin-типом `ULong` и покрывает весь диапазон YQL `Uint64`.

Интеграция со Spring Boot 3

Для Spring Boot 3 рекомендуется использовать официальный сценарий Exposed из документации по интеграции со Spring Boot 3 и добавить в приложение небольшую YDB-специфичную конфигурацию.

Логика ранее существовавшего starter-модуля может служить референсом для такой ручной настройки: в приложении нужно явно зарегистрировать диалект, явно указать YDB JDBC driver, согласовать `forceSignedDatetimes=...` в URL и создать `Database` из Spring-managed `DataSource`.

Зависимости

Maven

```
<!-- Укажите актуальную версию -->
<dependency>
    <groupId>org.jetbrains.exposed</groupId>
    <artifactId>exposed-spring-boot-starter</artifactId>
    <version>${exposed.version}</version>
</dependency>
<dependency>
    <groupId>tech.ydb.dialects</groupId>
    <artifactId>kotlin-exposed-ydb-dialect</artifactId>
    <version>${exposed.ydb.dialect.version}</version>
</dependency>
<dependency>
    <groupId>tech.ydb.jdbc</groupId>
    <artifactId>ydb-jdbc-driver</artifactId>
    <version>${ydb.jdbc.version}</version>
</dependency>
```

Gradle

```
dependencies {
    // Укажите актуальную версию
    implementation "org.jetbrains.exposed:exposed-spring-boot-starter:$exposedVersion"
    implementation "tech.ydb.dialects:kotlin-exposed-ydb-dialect:$ydbDialectVersion"
    implementation "tech.ydb.jdbc:ydb-jdbc-driver:$ydbJdbcVersion"
}
```

Базовая конфигурация Spring Boot

```
spring:
  datasource:
```

```
    url: jdbc:ydb:grpc://localhost:2136/local?forceSignedDatetimes=false
    driver-class-name: tech.ydb.jdbc.YdbDriver
  exposed:
    generate-ddl: false
    show-sql: false
    ydb:
      enable-signed-datetimes: false
```

Стандартные свойства official Exposed starter остаются доступными, в том числе:

- `spring.exposed.generate-ddl`
- `spring.exposed.excluded-packages`
- `spring.exposed.show-sql`

В примере выше используется YDB-специфичное свойство:

- `spring.exposed.ydb.enable-signed-datetimes`

В этом варианте имя свойства `spring.exposed.ydb.enable-signed-datetimes` выбирается самим приложением по соглашению с остальной конфигурацией. Его можно заменить на любое другое, если так удобнее. Если signed-режим включён, значение в этом свойстве нужно согласовать с JDBC URL, то есть использовать `forceSignedDatetimes=true`.

Пример собственной конфигурации

Ниже показан минимальный пример `@Configuration`, который добавляет к стандартной Spring Boot интеграции Exposed YDB-специфичные шаги:

- вызывает `registerYdbDialect(...)`;
- задаёт рекомендуемые значения `DatabaseConfig`;
- создаёт бин `Database` из Spring-managed `DataSource`.

```
import org.jetbrains.exposed.v1.core.DatabaseConfig
import org.jetbrains.exposed.v1.jdbc.Database
import org.springframework.beans.factory.InitializingBean
import org.springframework.beans.factory.annotation.Value
import org.springframework.context.annotation.Bean
import org.springframework.context.annotation.Configuration
import org.springframework.context.annotation.DependsOn
import tech.ydb.exposed.dialect.registerYdbDialect
import java.sql.Connection
import javax.sql.DataSource

@Configuration
class YdbExposedConfiguration(
    @Value("\${spring.exposed.ydb.enable-signed-datetimes:false}")
    private val enableSignedDatetimes: Boolean
) {
    @Bean
    fun ydbDialectRegistration(): InitializingBean = InitializingBean {
        registerYdbDialect(enableSignedDatetimes = enableSignedDatetimes)
    }

    @Bean
    fun databaseConfig(): DatabaseConfig = DatabaseConfig {
        defaultIsolationLevel = Connection.TRANSACTION_SERIALIZABLE
        defaultReadOnly = false
        useNestedTransactions = false
    }

    @Bean
    @DependsOn("ydbDialectRegistration")
    fun database(dataSource: DataSource, databaseConfig: DatabaseConfig): Database =
        Database.connect(
            datasource = dataSource,
            databaseConfig = databaseConfig
        )
}
```

Если в приложении уже есть собственная логика создания `DataSource` или `Database`, те же шаги можно перенести и в неё. Ключевые YDB-специфичные действия здесь остаются теми же: зарегистрировать диалект, вручную согласовать `forceSignedDatetimes` в JDBC URL, явно указать `tech.ydb.jdbc.YdbDriver` и передать Exposed рекомендуемый `DatabaseConfig`.

Таким образом, в типовом случае достаточно:

1. подключить `exposed-spring-boot-starter`, `kotlin-exposed-ydb-dialect` и `ydb-jdbc-driver`;
2. указать `spring.datasource.url`, начинающийся с `jdbc:ydb:` и содержащий `forceSignedDatetimes=...`;
3. добавить собственный `@Configuration`, аналогичный примеру выше;
4. при необходимости включить `spring.exposed.ydb.enable-signed-datetimes=true`.

То есть YDB-интеграция для Spring Boot здесь описывается как явная конфигурация приложения, а не как отдельный starter-артефакт.

Автоматическая генерация схемы

`spring.exposed.generate-ddl=true` работает так же, как и в official Exposed starter: при старте приложения Exposed создаёт схему на основе обнаруженных классов `Table`.

Для YDB здесь есть важный нюанс: таблицы, которые должны создаваться через DDL, нужно объявлять через обычный `Table` с переопределением:

```
override fun createStatement(): List<String> = createYdbStatement()
```

> **Важно**
>
> Если включить `spring.exposed.generate-ddl=true` и оставить таблицы на plain `Table` или `IdTable` без этого override, Exposed сгенерирует inline `PRIMARY KEY`, который YDB не принимает.

Транзакции в Spring

Для стандартной интеграции со Spring используйте `@Transactional`:

```
import org.springframework.stereotype.Service
import org.springframework.transaction.annotation.Transactional

@Service
class ProductService {
    @Transactional
    fun renameProduct(id: Int, newName: String) {
        // Exposed DSL
    }
}
```

`@Transactional` даёт обычную Spring-интеграцию Exposed, но не добавляет YDB retry policy для optimistic concurrency conflicts.

Если требуется retry-aware путь с автоматическими повторами на retryable-ошибках YDB, в своём Spring-сервисе можно вызывать `ydbTransaction(...)` напрямую, передавая бин `Database`:

```
import org.springframework.stereotype.Service
import org.jetbrains.exposed.v1.jdbc.Database
import tech.ydb.exposed.dialect.YdbRetryConfig
import tech.ydb.exposed.dialect.ydbTransaction

@Service
class RetryableProductService(
    private val database: Database
) {
    fun saveOrUpdate() =
        ydbTransaction(database, retry = YdbRetryConfig.IDEMPOTENT) {
            // Exposed DSL
        }
}
```

Обычно для одного участка кода выбирают один из двух путей:

- либо `@Transactional`;
- либо `ydbTransaction(database, ...)`, если нужны автоматические повторы.

Управление схемой

Для production-сценариев рекомендуется рассматривать DDL-генерацию Exposed как вспомогательный, а не основной путь. Обычно схема:

1. описывается отдельными SQL-скриптами;
2. мигрируется внешним инструментом с версионированием — например, Flyway или Liquibase;
3. сверяется с моделью Exposed на стороне приложения.

Если приложение использует schema validation или migration diff generation через Exposed, добавьте:

Maven

```
<dependency>
    <groupId>org.jetbrains.exposed</groupId>
    <artifactId>exposed-migration-core</artifactId>
    <version>${exposed.version}</version>
</dependency>
<dependency>
    <groupId>org.jetbrains.exposed</groupId>
    <artifactId>exposed-migration-jdbc</artifactId>
    <version>${exposed.version}</version>
</dependency>
```

Gradle

```
dependencies {
    implementation "org.jetbrains.exposed:exposed-migration-core:$exposedVersion"
    implementation "org.jetbrains.exposed:exposed-migration-jdbc:$exposedVersion"
}
```

В Exposed 1.3.0 полный путь `MigrationUtils.statementsRequiredForDatabaseMigration(...)` безусловно читает metadata по `CHECK`-constraint'ам из `INFORMATION_SCHEMA.CHECK_CONSTRAINTS`. Текущий JDBC-драйвер YDB эту часть metadata не предоставляет, поэтому полный generic diff Exposed для YDB работает не полностью.

Практический YDB-совместимый путь:

- накатывать схему внешними миграциями;
- использовать Exposed table definitions как описание клиентской модели;
- валидировать drift через совместимые metadata-пути Exposed:
  - существующие колонки;
  - существующие secondary indexes.

В репозитории диалекта есть integration coverage для сценария, где схема создаётся raw SQL, а затем сверяется с Exposed-моделью без падения на unsupported metadata.

> **Примечание**
>
> Если схема была только что изменена raw SQL, выполняйте validation в новой транзакции, чтобы Exposed не работал со stale metadata cache.

Signed temporal types в DDL

`enableSignedDatetimes` относится исключительно к генерации схемы через Exposed DDL: он меняет DDL-имена для стандартных Exposed-типов `date()`, `datetime()` и `timestamp()`. На уже существующие таблицы и на runtime-binding отдельных колонок этот флаг сам по себе ничего не меняет.

> **Примечание**
>
> Поскольку Exposed DDL — это test/dev-путь, флаг `enableSignedDatetimes` тоже относится в первую очередь к тестам и локальной разработке. В production temporal-типы фиксируются в версионируемых SQL-миграциях.

По умолчанию (`enableSignedDatetimes = false`) стандартные Exposed temporal-типы попадают в сгенерированный `CREATE TABLE` как:

- `Date`
- `Datetime`
- `Timestamp`

Если в Exposed DDL нужен signed-режим, согласуйте его в двух местах: включите `enableSignedDatetimes` у диалекта (это меняет DDL стандартных temporal-типов) и добавьте `forceSignedDatetimes=true` в JDBC URL (это меняет binding на стороне драйвера):

```
registerYdbDialect(enableSignedDatetimes = true)

val db = Database.connect(
    url = "jdbc:ydb:grpc://localhost:2136/local?forceSignedDatetimes=true",
    driver = "tech.ydb.jdbc.YdbDriver",
    databaseConfig = DatabaseConfig {
        defaultIsolationLevel = Connection.TRANSACTION_SERIALIZABLE
        defaultReadOnly = false
        useNestedTransactions = false
    }
)
```

В этом режиме стандартные Exposed-типы `date`, `datetime` и `timestamp` будут попадать в сгенерированный `CREATE TABLE` как `Date32`, `Datetime64` и `Timestamp64`.

Если нужен явный контроль типа отдельной колонки, используйте расширения `ydbDate(...)`, `ydbDate32(...)`, `ydbDatetime(...)`, `ydbDatetime64(...)`, `ydbTimestamp(...)` и `ydbTimestamp64(...)`. Они всегда генерируют DDL и привязывают параметр через JDBC vendor code именно того типа, который указан в имени функции, независимо от `enableSignedDatetimes`.

Если в JDBC URL уже есть другие параметры, флаг нужно добавлять через `&forceSignedDatetimes=true`.

Ограничения и особенности

- Exposed 1.3.0 не предоставляет dialect hook для рендера single-column `PRIMARY KEY` внутри `CREATE TABLE`;
- поэтому DDL-путь для YDB реализован как workaround через `createStatement()` и `createYdbStatement()`;
- каждая таблица, создаваемая через Exposed DDL в YDB, должна иметь `PRIMARY KEY`;
- `SchemaUtils.createMissingTablesAndColumns(...)` и `spring.exposed.generate-ddl=true` требуют такого же override `createStatement()`;
- ANSI `MERGE` не поддерживается;
- `upsert(where)` и PostgreSQL-подобная логика `ON CONFLICT DO UPDATE` не поддерживаются;
- functional indexes не поддерживаются;
- `GLOBAL UNIQUE` через `ALTER TABLE ... ADD INDEX ...` зависит от ограничений конкретной версии YDB;
- `LOCAL` индексы и column-oriented таблицы (`STORE = COLUMN`) first-class в диалекте не поддерживаются; при необходимости их можно навесить через произвольный YQL в `createYdbStatement()` (test-only) — диалект ориентирован на row-oriented OLTP-таблицы;
- полный generic diff через `MigrationUtils.statementsRequiredForDatabaseMigration(...)` в Exposed 1.3.0 для YDB работает не полностью из-за неподдерживаемой metadata по `CHECK`-ограничениям.

Drizzle ORM

Drizzle ORM — это лёгкая TypeScript ORM для Node.js с типобезопасным построителем запросов и DSL для описания схемы.

YDB поддерживает интеграцию с Drizzle ORM через адаптер `@ydbjs/drizzle-adapter`, который входит в YDB JavaScript/TypeScript SDK. Адаптер предоставляет привычный для Drizzle API и учитывает особенности YDB: первичные ключи, вторичные и векторные индексы, TTL, партиционирование и семейства колонок.

Эта страница — краткий обзор интеграции. Подробное руководство со всеми возможностями (построители запросов, реляционный API, миграции, DDL, векторный поиск) доступно на ydb.js.org.

Требования

- Node.js 20.19 или новее
- Пакеты `drizzle-orm` и `@ydbjs/drizzle-adapter`
- Доступный экземпляр YDB

Установка

```
npm install @ydbjs/drizzle-adapter drizzle-orm
```

Подключение

Строка подключения

Самый простой способ — передать строку подключения в `createDrizzle()`. Адаптер сам создаст и будет управлять драйвером YDB.

```
import { createDrizzle } from '@ydbjs/drizzle-adapter'

const db = createDrizzle({
  connectionString: process.env['YDB_CONNECTION_STRING']!,
})
```

Строка подключения имеет вид `grpc://localhost:2136/local` (или `grpcs://` для защищённого соединения), где указываются адрес сервера, порт gRPC и путь к базе данных.

Существующий драйвер SDK

Если приложение уже использует драйвер из `@ydbjs/core`, его можно обернуть в `YdbDriver` и переиспользовать общее соединение.

```
import { Driver } from '@ydbjs/core'
import { YdbDriver, createDrizzle } from '@ydbjs/drizzle-adapter'

const driver = new Driver('grpc://localhost:2136/local')
await driver.ready()
const db = createDrizzle({
  client: new YdbDriver(driver),
})
```

После завершения работы освободите ресурсы, если драйвер был создан адаптером:

```
db.$client.close?.()
```

Описание схемы

Таблицы описываются с помощью `ydbTable()`. Для каждой таблицы обязателен первичный ключ.

```
import { integer, text, timestamp, ydbTable } from '@ydbjs/drizzle-adapter/schema'

export const users = ydbTable('users', {
  id: integer('id').primaryKey(),
  email: text('email').notNull().unique(),
  name: text('name'),
  createdAt: timestamp('created_at')
    .notNull()
    .$defaultFn(() => new Date()),
})
```

Адаптер поддерживает все основные типы данных YDB (`Bool`, целочисленные и вещественные типы, `Utf8`, `String`, `Uuid`, `Json`, типы даты и времени и др.), а также YDB-специфичные возможности схемы: вторичные и векторные индексы, TTL, партиционирование и семейства колонок. Полный перечень типов и опций приведён в руководстве по схеме.

CRUD-операции

```
import { and, eq, like } from 'drizzle-orm'

// Вставка
await db.insert(users).values({ id: 1, email: 'alice@example.com', name: 'Alice' }).execute()

// Выборка с фильтрацией
const filteredUsers = await db
  .select({ userId: users.id, userName: users.name })
  .from(users)
```

```
  .where(and(eq(users.id, 1), like(users.email, '%@example.com')))
  .execute()

// Обновление
await db.update(users).set({ name: 'Alice Cooper' }).where(eq(users.id, 1)).execute()

// Удаление
await db.delete(users).where(eq(users.id, 1)).execute()
```

Для операции «вставить или обновить» по первичному ключу используется `onDuplicateKeyUpdate()`:

```
await db
  .insert(users)
  .values({ id: 1, email: 'alice_new@example.com', name: 'Alice Updated' })
  .onDuplicateKeyUpdate({ set: { name: 'Alice Updated' } })
  .execute()
```

Транзакции

Транзакции запускаются через `db.transaction()`. Поддерживаются режимы доступа и уровни изоляции YDB, а также флаг идемпотентности для автоматических повторов при сетевых ошибках.

```
import { eq } from 'drizzle-orm'
import { TransactionRollbackError } from 'drizzle-orm/errors'

try {
  await db.transaction(
    async (tx) => {
      // Проверяем предусловие перед изменением данных
      const inviter = await tx.select().from(users).where(eq(users.id, 1)).execute()
      if (inviter.length === 0) {
        tx.rollback()
      }

      await tx.insert(users).values({ id: 4, email: 'delta@example.com' }).execute()
    },
    {
      accessMode: 'read write',
      isolationLevel: 'serializableReadWrite',
      idempotent: true,
    }
  )
} catch (error) {
  if (error instanceof TransactionRollbackError) {
    console.log('transaction was rolled back')
  }
}
```

Ограничения

При использовании адаптера учитывайте особенности YDB и адаптера:

- Адаптер распространяется только в формате ESM.
- Вложенные транзакции не поддерживаются.
- Допустимы только уровни изоляции YDB (`serializableReadWrite`, `snapshotReadOnly`); эмуляции других уровней нет.
- `references()` хранится как метаданные для реляционного API — YDB не контролирует внешние ключи на уровне СУБД.

Полезные ссылки

- Подробное руководство по адаптеру
- YDB JavaScript/TypeScript SDK на GitHub
- npm-пакет `@ydbjs/drizzle-adapter`

Интеграции YDB со Spring

В этом разделе собраны интеграции YDB со Spring и Spring Boot.

Компонент	Инструкция
spring-ydb-retry	Автоматический повтор транзакций

Интеграция YDB со Spring Data JDBC описана в разделе Объектно-реляционное отображение (ORM).

Автоматический повтор транзакций YDB для Spring

spring-ydb-retry — это модуль автоконфигурации Spring Boot, который добавляет автоматический повтор транзакционных методов при возникновении повторяемых ошибок YDB.

> **Важно**
>
> Это экспериментальный модуль. Его API и поведение могут измениться в будущих версиях без сохранения обратной совместимости.

YDB является распределённой базой данных, поэтому часть ошибок носит временный (transient) характер: например, может устареть сессия, возникнуть кратковременная перегрузка, временно стать недоступным узел или транзакция может быть прервана из-за конкурентного изменения данных. Корректная обработка таких ошибок требует повторного выполнения транзакции целиком, а не отдельного запроса. Модуль `spring-ydb-retry` берёт эту логику на себя: он перехватывает транзакционные методы, классифицирует ошибки по кодам статуса YDB и при необходимости повторяет транзакцию с экспоненциальной задержкой и джиттером.

Возможности

- Автоматический повтор методов `@Transactional` при повторяемых кодах статуса YDB.
- Аннотация `@YdbTransactional` с настройками повтора на уровне отдельного метода (максимальное число попыток, backoff, идемпотентность).
- Двухуровневая стратегия задержки (быстрая/медленная) с джиттером, подобранная под семантику ошибок YDB.
- Режим идемпотентности для расширенного покрытия повторами на недетерминированных кодах статуса.
- Полная конфигурация через `application.properties` или `application.yaml`.

Требования

- Java 17 или новее.
- Spring Boot 3.4+ (основан на Spring Framework 6.2+).
- YDB JDBC Driver.
- Доступ к экземпляру базы данных YDB.

Установка

Для подключения модуля потребуются две зависимости: YDB JDBC Driver и сам модуль `spring-ydb-retry`.

Примеры для различных систем сборки:

Maven

```
<!-- Set actual versions -->
<dependency>
    <groupId>tech.ydb.jdbc</groupId>
    <artifactId>ydb-jdbc-driver</artifactId>
    <version>${ydb.jdbc.version}</version>
</dependency>

<dependency>
    <groupId>tech.ydb</groupId>
    <artifactId>spring-ydb-retry</artifactId>
    <version>${spring.ydb.retry.version}</version>
</dependency>
```

Gradle

```
dependencies {
    // Set actual versions
    implementation "tech.ydb.jdbc:ydb-jdbc-driver:$ydbJdbcVersion"
    implementation "tech.ydb:spring-ydb-retry:$springYdbRetryVersion"
}
```

Использование

Модуль настраивается автоматически через механизм автоконфигурации Spring Boot. Как только зависимость оказывается в classpath, модуль заменяет стандартный перехватчик транзакций Spring (бин `transactionInterceptor`) на свою реализацию и автоматически оборачивает логикой повтора все методы, помеченные `@Transactional` и `@YdbTransactional`. Никаких дополнительных аннотаций или явного подключения не требуется — достаточно настроить источник данных YDB:

```
spring.datasource.driver-class-name=tech.ydb.jdbc.YdbDriver
spring.datasource.url=jdbc:ydb:<grpc/grpcs>://<host>:<2135/2136>/path/to/database[?saFile=file:~/sa_key.json]
```

> **Примечание**
>
> Повтор применяется только к внешней границе транзакции: метод, который присоединяется к уже открытой транзакции (`PROPAGATION_REQUIRED` при активной транзакции), не повторяется отдельно — повторяется транзакция целиком на верхнем уровне.

Аннотация @YdbTransactional

`@YdbTransactional` нужна, когда для конкретного метода требуются собственные настройки повтора. Для обычного повтора достаточно `@Transactional`.

```java
@Service
public class UserService {

    private final JdbcTemplate jdbcTemplate;

    public UserService(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    @YdbTransactional(maxAttempts = 5, idempotent = true)
    public void save(String login) {
        jdbcTemplate.update("UPSERT INTO users (login) VALUES (?)", login);
    }
}
```

Параметры повтора:

Атрибут	По умолчанию	Описание
`enabled`	`true`	Включает или отключает повтор для метода. Локально повтор можно только отключить — включить его на методе нельзя, если он выключен глобально.
`maxAttempts`	`0`	Максимальное число попыток, включая первую. Например, `maxAttempts = 5` означает до пяти выполнений метода всего: первую попытку и не более четырёх повторов, а `maxAttempts = 1` — ровно один запуск без повторов. `0` — взять значение из `ydb.transaction.retry.max-attempts`. Отрицательные значения недопустимы.
`idempotent`	`false`	Помечает метод как идемпотентный. Часть кодов статуса YDB повторяется только в этом режиме.
`slowBackoffBaseMs`	`0`	Базовая задержка медленного backoff, мс. `0` — из глобальной конфигурации.
`fastBackoffBaseMs`	`0`	Базовая задержка быстрого backoff, мс. `0` — из глобальной конфигурации.
`slowCapBackoffMs`	`0`	Потолок медленного backoff, мс. `0` — из глобальной конфигурации.
`fastCapBackoffMs`	`0`	Потолок быстрого backoff, мс. `0` — из глобальной конфигурации.

Идемпотентность

Повтор транзакции означает её повторное выполнение целиком. Это безопасно только для идемпотентных операций — таких, повторный запуск которых приводит к тому же результату (например, `UPSERT` с детерминированным ключом или операции чтения).

- Без `idempotent = true` повторяются только заведомо временные коды статуса, для которых известно, что транзакция гарантированно не была применена.
- С `idempotent = true` дополнительно повторяются недетерминированные коды статуса (например, когда результат коммита неизвестен). Включайте этот режим только для операций, повторное выполнение которых безопасно.

```java
@YdbTransactional(idempotent = true)
public String findPayload(String guid, int id) {
    return jdbcTemplate.queryForObject(
            "SELECT payload FROM slo_test_table WHERE guid = ? AND id = ?",
            String.class, guid, id);
}
```

Стратегия повтора

Модуль извлекает код статуса YDB из цепочки исключений и принимает решение о повторе. Сначала код проверяется по политике повтора (повторяется всегда или только для идемпотентных операций), затем для повторяемого кода выбирается уровень backoff:

Код статуса	Когда повторяется	Уровень backoff
`ABORTED`	всегда	быстрый
`UNAVAILABLE`	всегда	быстрый
`OVERLOADED`	всегда	медленный
`CLIENT_RESOURCE_EXHAUSTED`	всегда	медленный
`BAD_SESSION`	всегда	нулевой
`SESSION_BUSY`	всегда	нулевой
`UNDETERMINED`	только при `idempotent = true`	быстрый
`TRANSPORT_UNAVAILABLE` (транспортная ошибка)	только при `idempotent = true`	быстрый
`CLIENT_GRPC_ERROR`	только при `idempotent = true`	быстрый
`SESSION_EXPIRED`	только при `idempotent = true`	нулевой

Уровни backoff:

- **Быстрый** — экспоненциальная задержка с базой `fastBackoffBaseMs` и потолком `fastCapBackoffMs`.

- **Медленный** — экспоненциальная задержка с базой `slowBackoffBaseMs` и потолком `slowCapBackoffMs`.
- **Нулевой** — повтор без задержки, поскольку он связан лишь с пересозданием сессии.

Остальные коды статуса (например, `TIMEOUT`, `PRECONDITION_FAILED`, `NOT_FOUND`) не повторяются.

Конфигурация

Глобальное поведение повтора настраивается в `application.properties` (приведены значения по умолчанию):

```
# Включение/отключение повтора
ydb.transaction.retry.enabled=true

# Максимальное число попыток, включая первую
ydb.transaction.retry.max-attempts=10

# Backoff для медленного уровня (OVERLOADED, CLIENT_RESOURCE_EXHAUSTED)
ydb.transaction.retry.slow-backoff-base-ms=50
ydb.transaction.retry.slow-cap-backoff-ms=5000

# Backoff для быстрого уровня (ABORTED, UNAVAILABLE, транспортные ошибки)
ydb.transaction.retry.fast-backoff-base-ms=5
ydb.transaction.retry.fast-cap-backoff-ms=500
```

Свойство	По умолчанию	Описание
`ydb.transaction.retry.enabled`	`true`	Глобальное включение/отключение повтора.
`ydb.transaction.retry.max-attempts`	`10`	Максимальное число попыток, включая первую. Например, значение `10` означает до десяти выполнений метода всего: первую попытку и не более девяти повторов. Допустимый минимум — `0`.
`ydb.transaction.retry.slow-backoff-base-ms`	`50`	Базовая задержка медленного backoff, мс.
`ydb.transaction.retry.slow-cap-backoff-ms`	`5000`	Потолок медленного backoff, мс.
`ydb.transaction.retry.fast-backoff-base-ms`	`5`	Базовая задержка быстрого backoff, мс.
`ydb.transaction.retry.fast-cap-backoff-ms`	`500`	Потолок быстрого backoff, мс.

Параметры `@YdbTransactional` переопределяют глобальные для конкретного метода. Значение `0` в аннотации означает «взять из глобальной конфигурации», отрицательные значения недопустимы.

Смотрите также

- Обработка ошибок
- Диалект YDB для Spring Data JDBC

Векторный поиск

Инструмент	Инструкция
LangChain	Инструкция
Mem0	Инструкция

LangChain

Интеграция YDB с LangChain позволяет использовать YDB в качестве векторного хранилища для RAG приложений.

Эта интеграция позволяет разработчикам эффективно управлять, запрашивать и извлекать векторизованные данные, что является основой для современных приложений, связанных с обработкой естественного языка, поиском и анализом данных. Используя модели эмбеддингов, пользователи могут создавать сложные системы, способные понимать и извлекать информацию на основе семантического сходства.

Интеграция доступна для Python и JavaScript.

Установка

Для использования этой интеграции установите локальный YDB. Для получения дополнительной информации см. Установите и запустите YDB.

Также установите пакеты LangChain и модель эмбеддингов для нужного языка:

Python

```
pip install -qU langchain-ydb
pip install -qU langchain-huggingface
```

JavaScript

```
npm install @ydbjs/langchain @langchain/core
npm install @langchain/community @huggingface/transformers
```

Инициализация

Для создания векторного хранилища YDB требуется указать модель эмбеддингов и параметры подключения:

Python

```
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_ydb.vectorstores import YDB, YDBSearchStrategy, YDBSettings

embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/all-mpnet-base-v2")

settings = YDBSettings(
    host="localhost",
    port=2136,
    database="/local",
    table="ydb_example",
    strategy=YDBSearchStrategy.COSINE_SIMILARITY,
)
vector_store = YDB(embeddings, config=settings)
```

JavaScript

```
import { HuggingFaceTransformersEmbeddings } from "@langchain/community/embeddings/huggingface_transformers";
import { YDBSearchStrategy, YDBVectorStore } from "@ydbjs/langchain";

const embeddings = new HuggingFaceTransformersEmbeddings({
  model: "sentence-transformers/all-mpnet-base-v2",
});

const vectorStore = new YDBVectorStore(embeddings, {
  connectionString: "grpc://localhost:2136/local",
  table: "ydb_example",
  strategy: YDBSearchStrategy.CosineSimilarity,
});
```

Управление векторным хранилищем

Создав векторное хранилище, можно взаимодействовать с ним, добавляя и удаляя различные элементы.

Добавление элементов

Подготовьте документы для работы:

Python

```
from uuid import uuid4

from langchain_core.documents import Document

uuids = [str(uuid4()) for _ in range(10)]
documents = [
    Document(
        page_content="I had chocolate chip pancakes and scrambled eggs for breakfast this morning.",
        metadata={"source": "tweet"},
        id=uuids[0],
    ),
    Document(
        page_content="The weather forecast for tomorrow is cloudy and overcast, with a high of 62 degrees.",
        metadata={"source": "news"},
```

```
        id=uuids[1],
    ),
    Document(
        page_content="Building an exciting new project with LangChain - come check it out!",
        metadata={"source": "tweet"},
        id=uuids[2],
    ),
    Document(
        page_content="Robbers broke into the city bank and stole $1 million in cash.",
        metadata={"source": "news"},
        id=uuids[3],
    ),
    Document(
        page_content="Wow! That was an amazing movie. I can't wait to see it again.",
        metadata={"source": "tweet"},
        id=uuids[4],
    ),
    Document(
        page_content="Is the new iPhone worth the price? Read this review to find out.",
        metadata={"source": "website"},
        id=uuids[5],
    ),
    Document(
        page_content="The top 10 soccer players in the world right now.",
        metadata={"source": "website"},
        id=uuids[6],
    ),
    Document(
        page_content="LangGraph is the best framework for building stateful, agentic applications!",
        metadata={"source": "tweet"},
        id=uuids[7],
    ),
    Document(
        page_content="The stock market is down 500 points today due to fears of a recession.",
        metadata={"source": "news"},
        id=uuids[8],
    ),
    Document(
        page_content="I have a bad feeling I am going to get deleted :(",
        metadata={"source": "tweet"},
        id=uuids[9],
    ),
]
```

Добавьте документы в векторное хранилище:

```
ids = vector_store.add_documents(documents=documents)
```

JavaScript

```
import { Document } from "@langchain/core/documents";
import { randomUUID } from "node:crypto";

const uuids = Array.from({ length: 10 }, () => randomUUID());
const documents = [
  new Document({
    pageContent: "I had chocolate chip pancakes and scrambled eggs for breakfast this morning.",
    metadata: { source: "tweet" },
    id: uuids[0],
  }),
  new Document({
    pageContent: "The weather forecast for tomorrow is cloudy and overcast, with a high of 62 degrees.",
    metadata: { source: "news" },
    id: uuids[1],
  }),
  new Document({
    pageContent: "Building an exciting new project with LangChain - come check it out!",
    metadata: { source: "tweet" },
    id: uuids[2],
  }),
  new Document({
    pageContent: "Robbers broke into the city bank and stole $1 million in cash.",
    metadata: { source: "news" },
    id: uuids[3],
  }),
  new Document({
    pageContent: "Wow! That was an amazing movie. I can't wait to see it again.",
    metadata: { source: "tweet" },
    id: uuids[4],
  }),
  new Document({
    pageContent: "Is the new iPhone worth the price? Read this review to find out.",
    metadata: { source: "website" },
    id: uuids[5],
  }),
  new Document({
    pageContent: "The top 10 soccer players in the world right now.",
    metadata: { source: "website" },
    id: uuids[6],
  }),
  new Document({
```

```
    pageContent: "LangGraph is the best framework for building stateful, agentic applications!",
    metadata: { source: "tweet" },
    id: uuids[7],
  }),
  new Document({
    pageContent: "The stock market is down 500 points today due to fears of a recession.",
    metadata: { source: "news" },
    id: uuids[8],
  }),
  new Document({
    pageContent: "I have a bad feeling I am going to get deleted :(",
    metadata: { source: "tweet" },
    id: uuids[9],
  }),
];
```

Добавьте документы в векторное хранилище:

```
const ids = await vectorStore.addDocuments(documents);
```

Удаление элементов

Элементы удаляются из векторного хранилища по идентификатору с использованием функции `delete`:

Python

```
vector_store.delete(ids=[ids[-1]])
```

JavaScript

```
await vectorStore.delete({ ids: [ids.at(-1)] });
```

Запросы в векторное хранилище

После создания векторного хранилища и добавления в него необходимых документов появляется возможность выполнения поисковых запросов в процессе выполнения цепочки или агента.

Прямой запрос

Поиск по сходству

Простой поиск по сходству можно выполнить следующим образом:

Python

```
results = vector_store.similarity_search(
    "LangChain provides abstractions to make working with LLMs easy",
    k=2,
)
for res in results:
    print(f"* {res.page_content} [{res.metadata}]")
```

JavaScript

```
const results = await vectorStore.similaritySearch(
  "LangChain provides abstractions to make working with LLMs easy",
  2,
);
for (const res of results) {
  console.log(`* ${res.pageContent} [${JSON.stringify(res.metadata)}]`);
}
```

Результат:

```
* Building an exciting new project with LangChain - come check it out! [{'source': 'tweet'}]
* LangGraph is the best framework for building stateful, agentic applications! [{'source': 'tweet'}]
```

Поиск по сходству с оценкой

Также можно выполнить поиск с оценкой:

Python

```
results = vector_store.similarity_search_with_score(
    "Will it be hot tomorrow?",
    k=3,
)
for res, score in results:
    print(f"* [SIM={score:.3f}] {res.page_content} [{res.metadata}]")
```

JavaScript

```
const results = await vectorStore.similaritySearchWithScore(
  "Will it be hot tomorrow?",
  3,
);
for (const [res, score] of results) {
  console.log(`* [SIM=${score.toFixed(3)}] ${res.pageContent} [${JSON.stringify(res.metadata)}]`);
}
```

Результат:

```
* [SIM=0.595] The weather forecast for tomorrow is cloudy and overcast, with a high of 62 degrees. [{'sourc
e': 'news'}]
* [SIM=0.212] I had chocolate chip pancakes and scrambled eggs for breakfast this morning. [{'source': 'twee
t'}]
* [SIM=0.118] Wow! That was an amazing movie. I can't wait to see it again. [{'source': 'tweet'}]
```

Фильтрация

Поиск с использованием фильтров выполняется следующим образом:

Python

```
results = vector_store.similarity_search_with_score(
    "What did I eat for breakfast?",
    k=4,
    filter={"source": "tweet"},
)
for res, _ in results:
    print(f"* {res.page_content} [{res.metadata}]")
```

JavaScript

```
const results = await vectorStore.similaritySearchWithScore(
  "What did I eat for breakfast?",
  4,
  { source: "tweet" },
);
for (const [res] of results) {
  console.log(`* ${res.pageContent} [${JSON.stringify(res.metadata)}]`);
}
```

Результат:

```
* I had chocolate chip pancakes and scrambled eggs for breakfast this morning. [{'source': 'tweet'}]
* Wow! That was an amazing movie. I can't wait to see it again. [{'source': 'tweet'}]
* Building an exciting new project with LangChain - come check it out! [{'source': 'tweet'}]
* LangGraph is the best framework for building stateful, agentic applications! [{'source': 'tweet'}]
```

Запрос через трансформацию в ретривер

Векторное хранилище можно трансформировать в поисковик (retriever) для упрощённого использования в цепочках.

Пример представлен ниже:

Python

```
retriever = vector_store.as_retriever(
    search_kwargs={
        "k": 2,
        "filter": {"source": "news"},
    },
)
results = retriever.invoke("Stealing from the bank is a crime")
for res in results:
    print(f"* {res.page_content} [{res.metadata}]")
```

JavaScript

```
const retriever = vectorStore.asRetriever({
  k: 2,
  filter: { source: "news" },
});
const results = await retriever.invoke("Stealing from the bank is a crime");
for (const res of results) {
  console.log(`* ${res.pageContent} [${JSON.stringify(res.metadata)}]`);
}
```

Результат:

```
* Robbers broke into the city bank and stole $1 million in cash. [{'source': 'news'}]
* The stock market is down 500 points today due to fears of a recession. [{'source': 'news'}]
```

Mem0

Mem0 — это слой долговременной памяти для AI-агентов и LLM-приложений. Mem0 извлекает из диалогов значимые факты о пользователе, сохраняет их в виде эмбеддингов и при следующих обращениях находит релевантные воспоминания по семантическому сходству. Благодаря этому агент «помнит» предпочтения и контекст пользователя между сессиями.

Интеграция с YDB позволяет использовать YDB в качестве векторного хранилища для памяти Mem0 — без отдельной специализированной векторной базы данных. Интеграция работает поверх LangChain: Mem0 поддерживает векторные хранилища LangChain в качестве бэкенда, а YDB предоставляет совместимое хранилище через пакет `langchain-ydb`.

> **Примечание**
>
> Помимо векторного хранилища, Mem0 использует языковую модель (LLM) для извлечения фактов из диалога и эмбеддинг-модель для их векторизации. В примерах ниже используются модели OpenAI (требуется переменная окружения `OPENAI_API_KEY`), но подойдёт любой поддерживаемый Mem0 провайдер.

Установка

Для использования этой интеграции установите локальный YDB. Для получения дополнительной информации см. Установите и запустите YDB.

Установите Mem0 и пакет интеграции YDB с LangChain:

```
pip install -qU mem0ai langchain-ydb langchain-openai
```

Инициализация

Создайте векторное хранилище YDB средствами `langchain-ydb` и передайте его в конфигурацию Mem0 с провайдером `langchain`:

Ключ к моделям OpenAI ожидается в переменной окружения `OPENAI_API_KEY` — задайте её заранее в окружении или менеджере секретов.

```
from langchain_openai import OpenAIEmbeddings
from langchain_ydb.vectorstores import YDB, YDBSearchStrategy, YDBSettings
from mem0 import Memory

embeddings = OpenAIEmbeddings(model="text-embedding-3-small")

vector_store = YDB(
    embeddings,
    config=YDBSettings(
        host="localhost",
        port=2136,
        database="/local",
        table="mem0_memories",
        strategy=YDBSearchStrategy.COSINE_SIMILARITY,
    ),
)

memory = Memory.from_config({
    "vector_store": {
        "provider": "langchain",
        "config": {"client": vector_store},
    },
    "embedder": {
        "provider": "langchain",
        "config": {"model": embeddings},
    },
})
```

Сохранение фактов

Передайте Mem0 сообщения диалога — модель сама выделит из них факты и сохранит в таблицу YDB. Память разделяется между пользователями с помощью `user_id`:

```
memory.add(
    [
        {"role": "user", "content": "Привет! Меня зовут Алиса, я работаю архитектором."},
        {"role": "assistant", "content": "Рада знакомству, Алиса!"},
        {"role": "user", "content": "Предпочитаю примеры на Python."},
    ],
    user_id="alice",
)
```

Поиск по памяти

Перед формированием ответа агент ищет релевантные воспоминания — Mem0 выполняет векторный поиск в YDB и возвращает факты, упорядоченные по сходству:

```
results = memory.search("На каком языке показывать примеры?", user_id="alice")
for item in results["results"]:
    print(f"{item['score']:.3f}  {item['memory']}")
```

Результат:

```
0.512  Предпочитает примеры на Python
0.231  Работает архитектором
```

Просмотр и удаление

Воспоминания пользователя можно получить или удалить целиком:

```python
# все воспоминания пользователя
for item in memory.get_all(user_id="alice")["results"]:
    print(item["memory"])

# удалить все воспоминания пользователя
memory.delete_all(user_id="alice")
```

Конвертация SQL-диалектов

Инструмент	Инструкция
Плагин ydb-sqlglot-plugin для SQLGlot	Инструкция
Конвертер SQL-диалектов в YQL	Инструкция

SQLGlot

SQLGlot — это написанный на чистом Python парсер, транспилер, оптимизатор и форматировщик SQL, поддерживающий более двадцати диалектов (PostgreSQL, MySQL, ClickHouse, BigQuery, Snowflake, Spark SQL и других). SQLGlot разбирает запрос в абстрактное синтаксическое дерево (AST), которое можно анализировать и преобразовывать программно, а затем сгенерировать обратно в SQL на любом из поддерживаемых диалектов.

Плагин ydb-sqlglot-plugin добавляет в SQLGlot диалект YDB. После его установки SQLGlot умеет как разбирать запросы на YQL, так и генерировать YQL из запросов, написанных на других диалектах. Преобразование двунаправленное: любой поддерживаемый диалект ↔ YQL.

В отличие от готового конвертера SQL-диалектов, который работает как внешний сервис и встроен в графические инструменты, плагин — это библиотека, которую вы подключаете в собственный Python-код. Это даёт два важных преимущества:

- **Локальная обработка.** Запросы не покидают вашу машину — никакие данные не отправляются на внешний HTTPS-сервис. Подходит для работы с конфиденциальными запросами.
- **Программный доступ к AST.** Кроме транспиляции, доступны разбор запроса, анализ происхождения столбцов (column lineage), оптимизация и форматирование — всё, что умеет SQLGlot.

Возможности

- Транспиляция запросов из других диалектов в YQL и обратно.
- Разбор YQL-запроса в AST для программного анализа и модификации.
- Анализ происхождения столбцов (column lineage) — отслеживание того, из каких таблиц и колонок получается каждый результирующий столбец.
- Оптимизация и форматирование запросов средствами SQLGlot.
- Поддержка специфичных для YQL конструкций: именованных выражений (`$variable`), модульных функций (`DateTime::GetYear()`), `FLATTEN` , лямбда-выражений, контейнерных типов и других.

> **Примечание**
>
> Полный список поддерживаемого функционала и актуальные ограничения приведены в README репозитория плагина.

Установка

Плагин публикуется в PyPI под именем `ydb-sqlglot-plugin` :

```
pip install ydb-sqlglot-plugin
```

Требования:

- Python 3.9 или новее.
- SQLGlot версии 28.6.0 или новее (устанавливается автоматически как зависимость).

Плагин распространяется под лицензией Apache 2.0.

Быстрый старт

После установки диалект YDB доступен в SQLGlot автоматически — дополнительные импорты не нужны. У него два эквивалентных имени, `ydb` и `yql` ; любое из них можно указывать в аргументах `read` и `write` .

Преобразование запроса из MySQL в YQL:

```
import sqlglot

result = sqlglot.transpile(
    "SELECT * FROM users WHERE id = 1",
    read="mysql",
    write="ydb",
)[0]

print(result)
# SELECT * FROM `users` WHERE id = 1
```

Обратное преобразование — из YQL в PostgreSQL. Именованное выражение YQL (`$t = (...)`) превращается в CTE (`WITH ... AS`):

```
import sqlglot

result = sqlglot.transpile(
    "$t = (SELECT id FROM users); SELECT * FROM $t AS t",
    read="ydb",
    write="postgres",
)[0]

print(result)
# WITH t AS (SELECT id FROM users) SELECT * FROM t AS t
```

> **Совет**
>
> Функция `sqlglot.transpile()` возвращает список строк — по одной на каждый оператор в исходном тексте. Если в запросе один оператор, берите первый элемент (`[0]`).

Примеры использования

Перенос запросов из других СУБД

При миграции в YDB плагин автоматически приводит конструкции исходного диалекта к правилам YQL. Например, имена с указанием схемы (`schema.table`) преобразуются в путь YDB с обратными кавычками:

```
import sqlglot

print(sqlglot.transpile("SELECT * FROM analytics.events", read="postgres", write="ydb")[0])
# SELECT * FROM `analytics/events`
```

Коррелированные подзапросы, не поддерживаемые в YDB напрямую, по возможности переписываются в `JOIN`, а конструкции `WITH` — в именованные выражения YQL.

Разбор запроса в AST

Если нужна не только транспиляция, но и анализ структуры запроса, разберите его в дерево с помощью `sqlglot.parse_one()`:

```
import sqlglot
from sqlglot import exp

tree = sqlglot.parse_one("SELECT id, name FROM `users` WHERE age > 18", dialect="ydb")

# Перечислить все таблицы, к которым обращается запрос
for table in tree.find_all(exp.Table):
    print(table.name)
# users

# Сгенерировать запрос обратно — на любом диалекте
print(tree.sql(dialect="clickhouse"))
```

Анализ происхождения столбцов

Поскольку YQL разбирается в стандартное AST SQLGlot, для запросов на YQL работает встроенный в SQLGlot анализ происхождения данных (column lineage). Это позволяет проследить, из каких исходных таблиц и колонок получается каждый результирующий столбец:

```
from sqlglot.lineage import lineage

node = lineage(
    "total",
    "SELECT SUM(amount) AS total FROM orders",
    dialect="ydb",
)

print(node.name)
# total
```

Анализ происхождения полезен при построении инструментов документирования данных, оценке влияния изменений схемы (impact analysis) и аудите запросов.

Сопоставление функций и типов

Плагин содержит таблицы соответствий между стандартными SQL-конструкциями и их аналогами в YDB.

Типы данных приводятся к эквивалентам YDB:

Тип в исходном диалекте	YDB
`TINYINT`	`Int8`
`INT`	`Int32`
`BIGINT`	`Int64`
`VARCHAR` , `TEXT`	`Utf8`
`TIMESTAMP`	`Timestamp`

Функции сопоставляются по семейству значений:

- **Дата и время:** `DATE_TRUNC` , `EXTRACT` , операции с интервалами.
- **Строки:** `CONCAT` , `UPPER` , `LOWER` , `LENGTH` и другие.
- **Коллекции:** `ARRAY` , `ARRAY_FILTER` , `UNNEST` → `FLATTEN BY` .
- **Условные и числовые:** `NULLIF` , `ROUND` , `COUNT()` .
- **JSON:** `JSON_VALUE` , `JSON_QUERY` с поддержкой сопутствующих конструкций.

См. также

- Конвертер SQL-диалектов в YQL
- ydb-sqlglot-plugin на GitHub
- SQLGlot
- YQL Reference

Конвертер SQL-диалектов в YQL

Конвертер SQL-диалектов — это сервис, преобразующий SQL-запросы, написанные на других диалектах (PostgreSQL, MySQL, ClickHouse и других), в синтаксис YQL. Сервис предназначен для упрощения переноса существующих запросов в YDB и снижения порога входа для разработчиков, уже знакомых с другими СУБД.

Конвертер доступен как отдельное веб-приложение по адресу ydb-dialect-converter.website.yandexcloud.net, а также встроен в плагин DBeaver для YDB и плагин VS Code для YDB. Пользователь вставляет запрос на исходном диалекте, выбирает диалект из списка и получает версию того же запроса на YQL.

Как пользоваться

1. Откройте веб-интерфейс конвертера.
2. Выберите исходный диалект SQL в выпадающем списке.
3. Вставьте запрос в поле **Входной SQL-запрос**. При необходимости воспользуйтесь готовыми примерами (`CTE Example`, `Create Table Example` и другими).
4. Нажмите **Конвертировать** — результат на YQL появится в поле **Результат конвертации (YDB)**.

Принцип работы

Преобразование выполняется в три этапа:

1. **Парсинг исходного SQL.** Исходный запрос разбирается парсером конкретного диалекта в абстрактное синтаксическое дерево (AST) — структуру, описывающую смысл запроса независимо от синтаксиса.
2. **Трансформация AST.** Специфичные для исходного диалекта конструкции заменяются на эквивалентные в YQL. Например, функции приведения типов, работа с датами и строками, операторы `LIMIT` / `OFFSET` и конструкции `WITH` адаптируются под правила YQL.
3. **Генерация YQL.** По изменённому AST генерируется текстовое представление запроса на YQL.

Этапы выполняются на базе библиотеки SQLGlot и плагина YDB для неё, который добавляет в SQLGlot отдельный диалект YDB. SQLGlot — SQL-парсер и транспилер с открытым исходным кодом, поддерживающий более двадцати диалектов. Он выполняет роль как парсера исходного запроса, так и генератора YQL.

Если нужна не разовая конвертация в графическом интерфейсе, а встраивание преобразования в собственный код или конвейер обработки данных, используйте плагин напрямую — см. статью SQLGlot.

> **Важно**
>
> Исходный запрос передаётся на внешний HTTPS-сервис. Не отправляйте в конвертер запросы, содержащие конфиденциальные данные (персональные данные, коммерческие тайны, идентификаторы реальных объектов в проде).

Поддерживаемые диалекты

Список исходных диалектов определяется набором, поддерживаемым SQLGlot. Среди основных:

- PostgreSQL;
- MySQL;
- ClickHouse;
- Microsoft SQL Server (T-SQL);
- Oracle;
- Snowflake;
- BigQuery;
- Presto / Trino;
- Spark SQL / Databricks;
- SQLite.

Актуальный полный список диалектов можно получить через API конвертера либо посмотреть в исходниках SQLGlot. Целевой диалект всегда один — YQL.

Поддерживаемые конструкции

Конвертер покрывает типовые конструкции аналитических и OLTP-запросов:

- Операторы выборки: `SELECT`, `JOIN` (все виды), `WHERE`, `GROUP BY`, `HAVING`, `ORDER BY`, `LIMIT`, `OFFSET`.
- Оконные функции (`OVER`, `PARTITION BY`, `ROWS BETWEEN`).
- Подзапросы и CTE (`WITH`).
- Приведение типов (`CAST`, `::`), арифметические и логические операции.
- Функции для строк, чисел, дат и времени — отображаются на ближайший эквивалент в YQL.
- Агрегатные функции (`COUNT`, `SUM`, `AVG`, `MIN`, `MAX` и другие).
- DML-операции: `INSERT`, `UPDATE`, `DELETE` (с учётом особенностей транзакций YDB).

Ограничения

Полная автоматическая конвертация возможна не всегда, поскольку YQL и другие SQL-диалекты различаются в моделях данных и семантике операций. В частности:

- **Специфичные для диалекта функции.** Функции, не имеющие прямого аналога в YQL (например, массивы PostgreSQL, `JSON_EXTRACT` MySQL с особенным синтаксисом путей), переводятся на приближённые эквиваленты либо оставляются для ручной доработки.
- **Хранимые процедуры и триггеры.** Не поддерживаются, так как YDB использует другую модель выполнения кода в базе.
- **Специфичный DDL.** `CREATE TABLE` для сложных типов может требовать ручной корректировки с учётом колоночных и строковых таблиц в YDB.
- **Подсказки оптимизатору (hints).** Игнорируются: оптимизатор YDB имеет собственные механизмы управления.

Результат преобразования следует рассматривать как черновик: сложные запросы требуют ручной проверки и адаптации перед выполнением.

См. также

- Веб-интерфейс конвертера SQL-диалектов
- SQLGlot
- Подключение к YDB с помощью плагина DBeaver
- YQL Reference
- SQLGlot

YDB CLI

Интерфейс командной строки YDB CLI — программа для управления вашими данными в YDB.

Для того чтобы воспользоваться YDB CLI, выполните установку, а также настройте соединение и аутентификацию.

Полное описание команд YDB CLI опубликовано в других статьях раздела, включая:

- Список объектов.
- Получение информации об объекте схемы.
- Работа с директориями.
- Выполнение запросов.
- Потоковое чтение строковых таблиц.
- Работа со вторичными индексами.
- Получение списка эндпоинтов для базы данных.
- Нагрузочное тестирование.

YDB CLI

Интерфейс командной строки YDB CLI — программа для управления вашими данными в YDB.

Для того чтобы воспользоваться YDB CLI, выполните установку, а также настройте соединение и аутентификацию.

Полное описание команд YDB CLI опубликовано в других статьях раздела, включая:

- Список объектов.
- Получение информации об объекте схемы.
- Работа с директориями.
- Выполнение запросов.
- Потоковое чтение строковых таблиц.
- Работа со вторичными индексами.
- Получение списка эндпоинтов для базы данных.
- Нагрузочное тестирование.

Установка YDB CLI

Linux

Чтобы установить YDB CLI, выполните команду:

```
curl -sSL https://install.ydb.tech/cli | bash
```

Скрипт установит YDB CLI и добавит путь к исполняемому файлу в переменную окружения `PATH`. Также скрипт сгенерирует файлы автодополнения и выведет инструкцию по включению автодополнения команд и опций в вашей оболочке.

> **Примечание**
>
> Скрипт дополнит переменную `PATH`, только если его запустить в командной оболочке bash или zsh. Если вы запустили скрипт в другой оболочке, добавьте путь до CLI в переменную `PATH` самостоятельно.
>
> Для работы автодополнения в bash необходим установленный пакет `bash-completion`.

> **Совет**
>
> При нестандартной конфигурации оболочки можно подключить сгенерированные файлы автодополнения из любого rc-файла. Файлы расположены в `~/.local/share/ydb/completion.bash.inc` и `~/.local/share/ydb/completion.zsh.inc` и автоматически обновляются при `ydb update`.

Чтобы обновить переменные окружения, перезапустите командную оболочку.

macOS

Чтобы установить YDB CLI, выполните команду:

```
curl -sSL https://install.ydb.tech/cli | bash
```

Скрипт установит YDB CLI и добавит путь до исполняемого файла в переменную окружения `PATH`. Также скрипт сгенерирует файлы автодополнения и выведет инструкцию по включению автодополнения команд и опций в вашей оболочке.

Чтобы обновить переменные окружения, перезапустите командную оболочку.

Windows

YDB CLI можно установить с помощью:

- PowerShell. Для этого выполните команду:

  ```
  iex (New-Object System.Net.WebClient).DownloadString('https://install.ydb.tech/cli-windows')
  ```

  Укажите, нужно ли добавить путь к исполняемому файлу в переменную окружения `PATH`:

  ```
  Add ydb installation dir to your PATH? [Y/n]
  ```

- cmd. Для этого выполните команду:

  ```
  @"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -Command "iex ((New-Object System.Net.Web
  Client).DownloadString('https://install.ydb.tech/cli-windows'))"
  ```

  Укажите, нужно ли добавить путь к исполняемому в переменную окружения `PATH`:

  ```
  Add ydb installation dir to your PATH? [Y/n]
  ```

  Чтобы обновить переменные окружения, перезапустите командную оболочку.

> **Примечание**
>
> YDB CLI использует символы Юникода в выводе некоторых команд. При некорректном отображении таких символов в консоли Windows, переключите кодировку на UTF-8:
>
> ```
> chcp 65001
> ```

Соединение CLI с базой данных и аутентификация

Большинство команд YDB CLI относится к операциям над базой данных YDB и требует соединения с ней для исполнения.

YDB CLI определяет, с какой БД необходимо соединиться и какой режим аутентификации использовать из следующих источников (в порядке убывания приоритета):

1. Командной строки
2. Профиля, выбранного опцией командной строки `--profile`
3. Переменных окружения
4. Активированного профиля

Для того чтобы YDB CLI попробовал соединиться с БД, в результате исполнения данных шагов должен быть определен эндпоинт и путь базы данных.

Если все шаги пройдены, но YDB CLI не определил режим аутентификации, то запросы будут отправляться на сервер YDB без добавления аутентификационных данных. Это может позволить успешно работать с локально развернутыми кластерами YDB, не требующими аутентификации. Для всех доступных по сети баз данных такие запросы будут отклонены сервером с выдачей ошибки аутентификации.

О возможных ситуациях, когда YDB CLI не будет пытаться соединиться с БД, читайте ниже в секции Сообщения об ошибках.

Параметры командной строки

Опции соединения с БД в командной строке указываются перед определением команды и её параметров:

```
ydb <опции_соединения> <команда> <опции_команды>
```

Параметры соединения с БД

- `-e, --endpoint <endpoint>` — эндпоинт - основной параметр соединения, позволяющий найти сервер YDB в сети. Если порт не указан, применяется 2135. Если протокол не указан, то в публичных сборках YDB CLI применяется gRPCs (с шифрованием).
- `-d, --database <database>` — путь базы данных.
- `--no-discovery` — пропустить этап discovery, на котором запрашивается список адресов для подключения к кластеру YDB. Если эта опция установлена, подключение будет производиться напрямую к эндпоинту, указанному пользователем (с помощью опции `-e`).

Параметры аутентификации

Режим и параметры аутентификации выбираются заданием одной из перечисленных ниже опций:

- `--token-file <filepath>` : используется режим аутентификации **Access Token**, в качестве которого используется содержимое файла, указанное в значении данной опции
- `--yc-token-file <filepath>` : используется режим аутентификации **Refresh Token**, в качестве которого используется содержимое файла, указанное в значении данной опции
- `--use-metadata-credentials` : используется режим аутентификации **Metadata**
- `--sa-key-file <filepath>` : используется режим аутентификации **Service Account Key**, а ключ и другие параметры загружаются из JSON-файла, указанного в значении данной опции
- `--user <username>` : используется режим аутентификации по логину и паролю, в качестве логина используется значение данной опции. Дополнительно могут быть указаны:
  - `--password-file <filename>` : пароль читается из контента указанного файла
  - `--no-password` : определяет пустой пароль

  Пароль будет запрошен в интерактивном режиме в случае, если в параметрах командной строки не указана ни одна из опций определения пароля из перечисленных выше.
- `--oauth2-key-file <filepath>` : используется режим аутентификации **OAuth 2.0 token exchange**, а ключ и другие параметры загружаются из JSON файла, указанного в значении данной опции. Опция `--iam-endpoint` используется для задания endpoint для обмена токена в формате `<schema>://<host>:<port>/<path>` (через параметр YDB CLI, профиль либо переменную окружения).

Если в командной строке задано одновременно несколько из перечисленных выше опций, то CLI выдаст ошибку с просьбой указать ровно одну:

```
$ ydb --use-metadata-credentials --iam-token-file ~/.ydb/token scheme ls
More than one auth method were provided via options. Choose exactly one of them
Try "--help" option for more info.
```

При использовании режимов аутентификации, предусматривающих ротацию токена с его периодическим перезапросом в IAM (**Refresh Token**, **Service Account Key**, **OAuth 2.0 token exchange**), может быть установлен специальный параметр, указывающий где размещен сервис IAM:

- `--iam-endpoint <URL>` : Задает URL сервиса IAM для обращения за новыми токенами в режимах аутентификации, подразумевающих ротацию токенов. Значение по умолчанию — `"iam.api.cloud.yandex.net"`.

Параметры TLS-соединения

При подключении к эндпоинту по gRPCs (с шифрованием) клиент проверяет серверный сертификат по цепочке доверия. Если сервер использует корневой CA, которого нет в стандартном наборе доверенных сертификатов CLI, то путь к нему указывается в параметре:

- `--ca-file <filename>` : файл корневого PEM-сертификата (CA)

Если в конфигурации кластера в секции client_certificate_authorization включен параметр `request_client_certificate: true`, сервер запрашивает клиентский сертификат при аутентификации устройств. Если нужна аутентификация устройств или аутентификация по клиентскому сертификату, на клиенте указываются:

- `--client-cert-file <filename>` : файл клиентского сертификата (PEM или PKCS#12)

- `--client-cert-key-file <filename>` : файл закрытого ключа клиентского сертификата (указывается, если сертификат и ключ содержатся в отдельных файлах)
- `--client-cert-key-password-file <filename>` : файл с паролем к закрытому ключу, если ключ зашифрован; если ключ зашифрован, а опция не указана, пароль будет запрошен интерактивно

Перечисленные выше опции используются и при аутентификации по клиентскому сертификату: на стороне сервера должен быть включён `request_client_certificate: true`, а на клиенте не должны быть заданы другие опции аутентификации. Если одновременно указаны `--client-cert-file` и другой режим аутентификации, сертификат передаётся при TLS-handshake, но для идентификации запросов используется аутентификационный токен, а не сертификат.

После успешной аутентификации клиенту назначается идентификатор защиты SID, который обладает всеми назначенными соответствующему идентификатору правами. Способ формирования SID зависит от режима аутентификации. Права на схему и данные не выдаются автоматически — см. авторизацию.

Параметры из профиля, выбранного опцией командной строки

Если в командной строке при вызове YDB CLI не указан какой-либо параметр соединения, то CLI производит попытку его определения по профилю, выбранному опцией командной строки `--profile`.

В профиле может быть определено большинство переменных, аналогичных опциям из раздела параметры командной строки. Обработка их значений производится также как и параметров командной строки.

Параметры из переменных окружения

Если в командной строке не был явно указан профиль или в нём не заданы отвечающие за аутентификацию параметры, то YDB CLI пытается определить режим и параметры аутентификации по окружению YDB CLI по следующему алгоритму:

- Если задано значение переменной окружения `IAM_TOKEN`, то используется режим аутентификации **Access Token**, в который передается значение данной переменной
- Иначе, если задано значение переменной окружения `YC_TOKEN`, то используется режим аутентификации **Refresh Token**, а токен для передачи на IAM endpoint при перезапросе берется из значения этой переменной
- Иначе, если задано значение переменной окружения `USE_METADATA_CREDENTIALS`, равное 1, то используется режим аутентификации **Metadata**
- Иначе, если задано значение переменной окружения `SA_KEY_FILE`, то используется режим аутентификации **Service Account Key**, а ключ загружается из файла, имя которого указано в данной переменной
- Если задано значение переменной окружения `YDB_USER` или `YDB_PASSWORD`, то используется режим аутентификации по логину и паролю. Логин считывается из переменной `YDB_USER`; если он не задан, то выдается ошибка `User password was provided without user name`. Пароль считывается из переменной `YDB_PASSWORD`; если она не задана, то в зависимости от наличия опции командной строки `--no-password` применяется либо пустой пароль, либо пароль будет запрошен интерактивно.
- Иначе, если задано значение переменной окружения `YDB_OAUTH2_KEY_FILE`, то используется режим аутентификации **OAuth 2.0 token exchange**, а параметры для обмена токена загружаются из JSON файла, имя которого указано в данной переменной. Опция `--iam-endpoint` используется для задания endpoint для обмена токена в формате `<schema>://<host>:<port>/<path>` (через параметр YDB CLI, профиль либо переменную окружения).

Параметры из активированного профиля

Если на предыдущих шагах не удалось определить какой-либо параметр соединения и в командной строке не был явно указан профиль опцией `--profile`, то YDB CLI пробует использовать параметры соединения из активированного профиля.

Сообщения об ошибках

Ошибки до попытки установки соединения с БД

Если все шаги из приведенных в начале данной статьи пройдены, но не удалось определить эндпоинт, то выполнение команды будет прервано с выдачей сообщения `Missing required option 'endpoint'`.

Если все шаги из приведенных в начале данной статьи пройдены, но не удалось определить путь базы данных, то выполнение команды будет прервано с выдачей сообщения `Missing required option 'database'`.

Если режим аутентификации определить удалось, но не удалось определить необходимые дополнительные параметры, то выполнение команды будет прервано с выдачей сообщения, описывающего проблему:

```
(No such file or directory) util/system/file.cpp:857:
can't open "<filepath>" with mode RdOnly|Seq (0x00000028)
```

— не удалось открыть на чтение файл `<filepath>`, указанный в одном из параметров, где передается имя файла с путем

Проверка аутентификации

Сервисная команда YDB CLI `discovery whoami` позволяет проверить, под какой учетной записью вы фактически аутентифицировались на сервере.

Команды YDB CLI

Общий синтаксис вызова команд YDB CLI:

```
ydb [global options] <command> [<subcommand> ...] [command options]
```

, где:

- `ydb` - команда запуска YDB CLI из командной строки операционной системы
- `[global options]` - глобальные опции, одинаковые для всех команд YDB CLI
- `<command>` - команда
- `[<subcomand> ...]` - подкоманды, указываемые в случае если выбранная команда содержит подкоманды
- `[command options]` - опции команды, специфичные для каждой команды и подкоманд

Команды

Вы можете ознакомиться с нужными командами выбрав тематический раздел в меню слева, или воспользовавшись алфавитным перечнем ниже.

Любая команда может быть вызвана в командной строке с опцией `--help` для получения справки по ней. Перечень всех поддерживаемых YDB CLI команд может быть получен запуском YDB CLI с опцией `--help` без указания команды.

Команда / подкоманда	Краткое описание
admin cluster bootstrap	Инициализация подготовленного кластера
admin cluster bridge failover	Аварийное переключение при недоступности pile
admin cluster bridge list	Список состояния каждого pile в режиме bridge
admin cluster bridge rejoin	Возвращение pile в кластер после обслуживания или восстановления
admin cluster bridge switchover	Плановая смена `PRIMARY` pile
admin cluster bridge takedown	Вывод pile из кластера для обслуживания
admin cluster config fetch	Получение текущей динамической конфигурации кластера
admin cluster config generate	Генерация динамической конфигурации из статической конфигурации запуска
admin cluster config replace	Замена динамической конфигурации кластера
admin cluster config resolve	Вычисление итоговой динамической конфигурации кластера на основе базовой конфигурации и селекторов переопределения
admin cluster config version	Отображение версии конфигурации кластера на узлах
admin cluster dump	Выгрузка метаданных кластера в файловую систему
admin cluster restore	Восстановление метаданных кластера из файловой системы
admin database config fetch	Получение текущей динамической конфигурации базы данных
admin database config generate	Генерация динамической конфигурации базы данных из статической конфигурации запуска
admin database config replace	Замена динамической конфигурации базы данных
admin database config resolve	Вычисление итоговой динамической конфигурации базы данных на основе базовой конфигурации и селекторов переопределения
admin database config version	Отображение версии конфигурации базы данных
admin database dump	Выгрузка метаданных и данных базы данных в файловую систему
admin database restore	Восстановление метаданных и данных базы данных из файловой системы
admin node config init	Инициализация конфигурации узла
auth get-token	Получение аутентификационного токена из параметров аутентификации
config info	Просмотр параметров соединения
config profile activate	Активация профиля
config profile create	Создание профиля
config profile delete	Удаление профиля

config profile deactivate	Деактивация текущего активного профиля
config profile get	Получение параметров профиля
config profile list	Список профилей
config profile replace	Создание или замена профиля с новыми значениями параметров
config profile set	Активация профиля
config profile update	Обновление существующего профиля
debug latency	Проверка базовой задержки с переменным количеством параллельных запросов
debug ping	Проверка доступности YDB
discovery list	Список эндпоинтов
discovery whoami	Проверка аутентификации
export s3	Экспорт данных в хранилище S3
import file csv	Импорт данных из CSV-файла
import file json	Импорт данных из JSON-файла
import file parquet	Импорт данных из Parquet-файла
import file tsv	Импорт данных из TSV-файла
import s3	Импорт данных из хранилища S3
init	Инициализация CLI, создание профиля
monitoring healthcheck	Проверка состояния базы
operation cancel	Прерывание исполнения фоновой операции
operation forget	Удаление фоновой операции из списка
operation get	Статус фоновой операции
operation list	Список фоновых операций
scheme describe	Описание объекта схемы данных
scheme ls	Список объектов схемы данных
scheme mkdir	Создание директории
scheme permissions chown	Изменение владельца объекта
scheme permissions clear	Очистка разрешений
scheme permissions grant	Предоставление разрешения
scheme permissions revoke	Удаление разрешения
scheme permissions set	Установка разрешений
scheme permissions list	Просмотр разрешений
scheme permissions clear-inheritance	Запрет наследования разрешений
scheme permissions set-inheritance	Установка наследования разрешений
scheme rmdir	Удаление директории
scripting yql	Выполнение YQL-скрипта (команда устарела, используйте `ydb sql`)
sql	Выполнение любого запроса
table attribute add	Добавление атрибута для строковой или колоночной таблицы
table attribute drop	Удаление атрибута у строковой или колоночной таблицы
table drop	Удаление строковой или колоночной таблицы
table index add global-async	Добавление асинхронного вторичного индекса для строковых таблиц
table index add global-sync	Добавление синхронного вторичного индекса для строковых таблиц
table index drop	Удаление вторичного индекса у строковых таблиц
table index rename	Переименование вторичного индекса для указанной таблицы

table query execute	Исполнение YQL-запроса (команда устарела, используйте `ydb sql`)
table query explain	Получение плана исполнения YQL-запроса (команда устарела, используйте `ydb sql --explain`)
table read	Потоковое чтение строковой таблицы
table ttl set	Установка параметров TTL для строковых и колоночных таблиц
table ttl reset	Сброс параметров TTL для строковых и колоночных таблиц
tools copy	Копирование таблиц
tools dump	Выгрузка отдельных схемных объектов в файловую систему
tools infer csv	Генерация текста запроса `CREATE TABLE SQL` из CSV файла
tools rename	Переименование строковых таблиц
tools restore	Восстановление отдельных схемных объектов из файловой системы
topic create	Создание топика
topic alter	Модификация параметров топика и перечня читателей
topic drop	Удаление топика
topic consumer add	Добавление читателя в топик
topic consumer describe	Описание читателя топика
topic consumer drop	Удаление читателя из топика
topic consumer offset commit	Сохранение смещения чтения
topic read	Чтение сообщений из топика
topic write	Запись сообщений в топик
update	Обновление YDB CLI
version	Вывод информации о версии YDB CLI
workload clickbench init	Создание и инициализация таблиц для нагрузки `Clickbench`
workload clickbench import files	Загрузка набора данных `Clickbench` из файлов
workload clickbench run	Выполнение бенчмарка `Clickbench`
workload clickbench clean	Удаление таблиц, созданных на этапе инициализации нагрузки `Clickbench`
workload kv init	Создание и инициализация таблиц для `Key-Value` нагрузки
workload kv run upsert	Вставка случайных кортежей в таблицу при помощи конструкции `UPSERT` в `Key-Value` нагрузке
workload kv run insert	Вставка случайных кортежей в таблицу при помощи конструкции `INSERT` в `Key-Value` нагрузке
workload kv run mixed	Одновременная вставка и чтение кортежей с проверкой успешности чтения записанных данных в `Key-Value` нагрузке
workload kv run read-rows	Выполнение ReadRows запросов, возвращающих строки по точному совпадению первичного ключа в `Key-Value` нагрузке
workload kv run select	Выборка данных, возвращающих строки по точному совпадению первичного ключа в `Key-Value` нагрузке
workload kv clean	Удаление таблиц, созданных на этапе инициализации `Key-Value` нагрузки
workload log init	Создание и инициализация таблиц для `Log` нагрузки
workload log import generator	Генератор случайных данных в `Log` нагрузке
workload log run bulk-upsert	Массовая вставка случайных строк в таблицу около текущего времени в `Log` нагрузке
workload log run delete	Удаление случайных строк из таблицы около текущего времени в `Log` нагрузке
workload log run insert	Вставка случайных строк в таблицу около текущего времени в `Log` нагрузке с помощью команды `INSERT`
workload log run upsert	Вставка случайных строк в таблицу около текущего времени в `Log` нагрузке с помощью команды `UPSERT`
workload log run select	Выполнение набора аналитических запросов для анализа логов: подсчет записей, агрегация по уровням, сервисам и компонентам, анализ метаданных и временные диапазоны в `Log` нагрузке
workload log clean	Удаление таблиц, созданных на этапе инициализации `Log` нагрузки

workload mixed init	Создание и инициализация таблиц для `Mixed` нагрузки
workload mixed run bulk-upsert	Массовая вставка случайных строк в таблицу около текущего времени с помощью команды `BULK_UPSERT` в `Mixed` нагрузке
workload mixed run insert	Вставка случайных строк в таблицу около текущего времени с помощью команды `INSERT` в `Mixed` нагрузке
workload mixed run upsert	Обновление случайных строк в таблице около текущего времени с помощью команды `UPSERT` в `Mixed` нагрузке
workload mixed run select	Выборка случайных строк из таблицы в `Mixed` нагрузке
workload mixed clean	Удаление таблиц, созданных на этапе инициализации `Mixed` нагрузки
workload query init	Инициализация таблиц и их конфигураций для `Query` нагрузки
workload query import	Заполнение таблиц данными для `Query` нагрузки
workload query run	Запуск нагрузочного тестирования `Query` нагрузки
workload query clean	Удаление таблиц, используемых для `Query` нагрузки
workload stock init	Создание и инициализация таблиц для `Stock` нагрузки
workload stock run add-rand-order	Вставка заказов со случайным ID без их обработки в `Stock` нагрузке
workload stock run put-rand-order	Отправка случайных заказов с обработкой в `Stock` нагрузке
workload stock run put-same-order	Отправка заказов с одинаковыми продуктами в `Stock` нагрузке
workload stock run rand-user-hist	Выборка заказов случайного клиента в `Stock` нагрузке
workload stock run user-hist	Выборка заказов 10000-го клиента в `Stock` нагрузке
workload stock clean	Удаление таблиц, созданных на этапе инициализации `Stock` нагрузки
workload topic init	Создание и инициализация топика для `Topic` нагрузки
workload topic run full	Выполнение полной нагрузки на топик с одновременным чтением и записью сообщений в `Topic` нагрузке
workload topic run read	Выполнение нагрузки на чтение сообщений из топика в `Topic` нагрузке
workload topic run write	Выполнение нагрузки на запись сообщений в топик в `Topic` нагрузке
workload topic clean	Удаление топика, созданного на этапе инициализации `Topic` нагрузки
workload tpcc init	Создание и инициализация таблиц для бенчмарка `TPC-C`
workload tpcc import	Заполнение таблиц начальными данными бенчмарка `TPC-C`
workload tpcc check	Проверка согласованности данных `TPC-C`
workload tpcc run	Запуск бенчмарка `TPC-C`
workload tpcc clean	Удаление таблиц, созданных бенчмарком `TPC-C`
workload tpcds init	Создание и инициализация таблиц для бенчмарка `TPC-DS`
workload tpcds import generator	Генерация набора данных `TPC-DS` с помощью встроенного генератора
workload tpcds run	Выполнение бенчмарка `TPC-DS`
workload tpcds clean	Удаление таблиц, созданных на этапе инициализации `TPC-DS`
workload tpch init	Создание и инициализация таблиц для бенчмарка `TPC-H`
workload tpch import generator	Генерация набора данных `TPC-H` с помощью встроенного генератора
workload tpch run	Выполнение бенчмарка `TPC-H`
workload tpch clean	Удаление таблиц, созданных на этапе инициализации `TPC-H`
workload transfer topic-to-table init	Создание и инициализация топика с консьюмерами и таблиц для нагрузки на передачу данных из топика в таблицу
workload transfer topic-to-table run	Запуск нагрузки с чтением сообщений из топика и записью в таблицу в транзакциях
workload transfer topic-to-table clean	Удаление топика и таблиц, созданных на этапе инициализации

workload vector init	Создание и инициализация таблиц для `Vector` нагрузки
workload vector run select	Получение топ-K векторов в `Vector` нагрузке
workload vector run upsert	Upsert векторных строк в таблицу в `Vector` нагрузке
workload vector clean	Удаление таблиц, созданных на этапе инициализации `Vector` нагрузки
workload fulltext init	Создание и инициализация таблиц для `Fulltext` нагрузки
workload fulltext import files	Импорт полнотекстовых данных из файлов в `Fulltext` нагрузке
workload fulltext import generator	Генерация синтетических текстовых данных с помощью модели цепи Маркова в `Fulltext` нагрузке
workload fulltext run select	Выполнение полнотекстовых поисковых запросов в `Fulltext` нагрузке
workload fulltext run upsert	Вставка или обновление документов в таблице в `Fulltext` нагрузке
workload fulltext clean	Удаление таблиц, созданных на этапе инициализации `Fulltext` нагрузки
workload fulltext model	Построение модели цепи Маркова из текстового датасета для `Fulltext` нагрузки
yql	Выполнение YQL-скрипта с поддержкой стриминга (команда устарела, используйте `ydb sql`)

Сервисные команды

Данные команды относятся к самому клиентскому приложению YDB CLI и не предполагают установки соединения с БД. Они могут быть выражены как в виде параметра, так и в виде опции.

Имя	Описание
`-?`, `-h`, `--help`	Вывод справки о синтаксисе YDB CLI
`version`	Вывод информации о версии YDB CLI (для публичных сборок)
`update`	Обновление YDB CLI до последней версии (для публичных сборок)
`config info`	Просмотр параметров соединения
`--license`	Показать лицензию (для публичных сборок)
`--credits`	Показать лицензии сторонних продуктов (для публичных сборок)

Если неизвестно, является ли используемая сборка YDB CLI публичной, то уточнить поддержку той или иной сервисной команды в ней можно вызовом справки.

Глобальные параметры

Опции соединения с БД

Опции соединения с БД описаны в статье Соединение с БД и аутентификация.

Сервисные опции

- `--profile <name>` — указывает на использование профиля соединения с БД с заданным именем при выполнении какой-либо команды YDB CLI. В профиле может быть сохранено большинство параметров соединения.
- `-v, --verbose` — вывод детальной информации обо всех выполняемых операциях. Указание данной опции полезно для локализации проблем при соединении с БД.
- `--profile-file` — использовать профили из указанного файла. По умолчанию используются профили из файла `~/.ydb/config/config.yaml`.

table attribute add

С помощью команды `table attribute add` вы можете добавить пользовательский атрибут указанной таблице.

Общий вид команды:

```
ydb [global options...] table attribute add [options...] <table path>
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.
- `table path` — путь таблицы.

Посмотрите описание команды добавления пользовательских атрибутов:

```
ydb table attribute add --help
```

Параметры подкоманды

Имя	Описание
`--attribute`	Пользовательский атрибут в формате `<ключ>=<значение>`. Вы можете использовать `--attribute` несколько раз, чтобы добавить несколько атрибутов в одной команде.

Примеры

Добавьте пользовательский атрибуты с ключами `attr_key1`, `attr_key2` и значениями `attr_value1`, `attr_value2` соответственно таблице `my-table`:

```
ydb table attribute add --attribute attr_key1=attr_value1 --attribute attr_key2=attr_value2 my-table
```

table attribute drop

С помощью команды `table attribute drop` вы можете удалить пользовательский атрибут указанной таблицы.

Общий вид команды:

```
ydb [global options...] table attribute drop [options...] <table path>
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.
- `table path` — путь таблицы.

Посмотрите описание команды добавления пользовательских атрибутов:

```
ydb table attribute drop --help
```

Параметры подкоманды

Имя	Описание
`--attributes`	Ключ пользовательского атрибута, который нужно удалить. Вы можете указать несколько ключей, используя `,` в качестве разделителя.

Примеры

Удалите пользовательские атрибуты с ключами `attr_key1` и `attr_key2` таблицы `my-table`:

```
ydb table attribute drop --attributes attr_key1,attr_key2 my-table
```

Список объектов

Команда `scheme ls` позволяет получить список схемных объектов в базе данных:

```
ydb [connection options] scheme ls [path] [-lR1]
```

, где `[connection options]` — опции соединения с БД

При запуске без параметров выводится перечень имен объектов в корневой директории базы данных в сжатом формате.

Параметром `path` можно задать директорию, для которой нужно вывести перечень объектов.

Для команды доступны следующие опции:

- `-l` — полная информация об атрибутах каждого объекта;
- `-R` — рекурсивный обход всех поддиректорий;
- `-1` — выводить по одному объекту схемы на строку (например, для последующей обработки в скрипте).

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

- Получение объектов в корневой директории базы данных в сжатом формате

```
ydb --profile quickstart scheme ls
```

- Получение объектов во всех директориях базы данных в сжатом формате

```
ydb --profile quickstart scheme ls -R
```

- Получение объектов в заданной директории базы данных в сжатом формате

```
ydb --profile quickstart scheme ls dir1
ydb --profile quickstart scheme ls dir1/dir2
```

- Получение объектов во всех поддиректориях заданной директории базы данных в сжатом формате

```
ydb --profile quickstart scheme ls dir1 -R
ydb --profile quickstart scheme ls dir1/dir2 -R
```

- Получение полной информации по объектам в корневой директории базы данных

```
ydb --profile quickstart scheme ls -l
```

- Получение полной информации по объектам в заданной директории базы данных

```
ydb --profile quickstart scheme ls dir1 -l
ydb --profile quickstart scheme ls dir2/dir3 -l
```

- Получение полной информации по объектам во всех директориях базы данных

```
ydb --profile quickstart scheme ls -lR
```

- Получение полной информации по объектам во всех поддиректориях заданной директории базы данных

```
ydb --profile quickstart scheme ls dir1 -lR
ydb --profile quickstart scheme ls dir2/dir3 -lR
```

Получение информации об объекте схемы

Получите информацию об объекте схемы:

```
ydb scheme describe episodes --stats
```

Результат:

```
<table> episodes

┌─────────────┬──────────┬────────┬─────┐
| Name        | Type     | Family | Key |
├─────────────┼──────────┼────────┼─────┤
| air_date    | Uint64? |        |     |
| episode_id | Uint64? |        | K2  |
| season_id   | Uint64? |        | K1  |
| series_id   | Uint64? |        | K0  |
| title       | Utf8?    |        |     |
└─────────────┴──────────┴────────┴─────┘

Storage settings:
Internal channel 0 commit log storage pool: ssd
Internal channel 1 commit log storage pool: ssd
Store large values in "external blobs": false

Column families:
┌──────────┬──────┬──────────────┬──────────────────┐
| Name     | Data | Compression | Keep in memory |
├──────────┼──────┼──────────────┼──────────────────┤
| default | ssd  | None         |                  |
└──────────┴──────┴──────────────┴──────────────────┘

Auto partitioning settings:
Partitioning by size: true
Partitioning by load: false
Preferred partition size (Mb): 2048

Table stats:
Partitions count: 1
Approximate number of rows: 70
Approximate size of table: 11.05 Kb
Last modified: Thu, 17 Jun 2021 11:01:06 UTC
Created: Thu, 17 Jun 2021 11:00:29 UTC
```

Разрешения

Общий перечень команд

Получить список доступных команд можно через интерактивную справку:

```
ydb scheme permissions --help
```

```
Usage: ydb [global options...] scheme permissions [options...] <subcommand>

Description: Modify permissions

Subcommands:
permissions                Modify permissions
├─ chown                   Change owner
├─ clear                   Clear permissions
├─ grant                   Grant permission (aliases: add)
├─ list                    List permissions
├─ revoke                  Revoke permission (aliases: remove)
├─ set                     Set permissions
├─ clear-inheritance       Set to do not inherit permissions from the parent
└─ set-inheritance         Set to inherit permissions from the parent
```

Все команды имеют дополнительный параметр, который для них не критичен:
`--timeout ms` - технологический параметр, задающий таймаут на отклик сервера.

grant, revoke

Команды `grant` и `revoke` позволяют установить и отозвать, соответственно, права доступа к объектам схемы для пользователя или группы пользователей. По сути, это аналоги соответствующих YQL-команд GRANT и REVOKE.

Синтаксис команд YDB CLI выглядит следующим образом:

```
ydb [connection options] scheme permissions grant  [options...] <path> <subject>
ydb [connection options] scheme permissions revoke [options...] <path> <subject>
```

Параметры:

`<path>` — полный путь от корня кластера до объекта, права на который необходимо модифицировать;
`<subject>` — имя пользователя или группы, для которых изменяются права доступа.

Дополнительные параметры `[options...]`:

`{-p|--permission} NAME` — список прав, которые необходимо предоставить (grant) или отозвать (revoke) у пользователя.

Каждое право нужно передавать отдельным параметром, например:

```
ydb scheme permissions grant -p "ydb.access.grant" -p "ydb.generic.read" '/Root/db1/MyApp/Orders' testuser
```

set

Команда `set` позволяет установить права доступа к объектам схемы для пользователя или группы пользователей.

Синтаксис команды:

```
ydb [connection options] scheme permissions set  [options...] <path> <subject>
```

Значения всех параметров полностью идентичны командам `grant`, `revoke`. Однако ключевое отличие команды `set` от `grant` и `revoke` заключается в установке на указанный объект ровно тех прав доступа, которые перечислены в параметрах `-p (--permission)`. Другие права для указанного пользователя или группы будут отозваны.

Например, ранее пользователю `testuser` на объект `'/Root/db1'` были выданы права `"ydb.granular.select_row"`, `"ydb.granular.update_row"`, `"ydb.granular.erase_row"`, `"ydb.granular.read_attributes"`, `"ydb.granular.write_attributes"`, `"ydb.granular.create_directory"`.
Тогда в результате выполнения команды все права на указанный объект будут отозваны (как будто для каждого из прав вызывали `revoke`) и останется только право `"ydb.granular.select_row"` — непосредственно указанное в команде `set`:

```
ydb scheme permissions set -p "ydb.granular.select_row" '/Root/db1' testuser
```

list

Команда `list` позволяет получить текущий список прав доступа к объектам схемы.

Синтаксис команды:

```
ydb [connection options] scheme permissions list [options...] <path>
```

Параметры:
`<path>` — полный путь от корня кластера до объекта, права на который необходимо получить.

Пример результата выполнения `list`:

```
ydb scheme permissions list '/Root/db1/MyApp'
```

```
Owner: root

Permissions:
user1:ydb.generic.read

Effective permissions:
USERS:ydb.database.connect
METADATA-READERS:ydb.generic.list
DATA-READERS:ydb.granular.select_row
DATA-WRITERS:ydb.tables.modify
DDL-ADMINS:ydb.granular.create_directory,ydb.granular.write_attributes,ydb.granular.create_table,ydb.granula
r.remove_schema,ydb.granular.alter_schema
ACCESS-ADMINS:ydb.access.grant
DATABASE-ADMINS:ydb.generic.manage
user1:ydb.generic.read
```

Структура результата состоит из трёх блоков:

- `Owner` — показывает владельца объекта схемы.
- `Permissions` — отображает список прав, выданных непосредственно на данный объект.
- `Effective permissions` — отображает список прав, фактически действующих на данный схемный объект с учётом правил наследования прав. Данный список также включает все права, отображаемые в секции `Permissions`.

clear

Команда `clear` позволяет отозвать все ранее выданные права на схемный объект. Права, действующие на него по правилам наследования, продолжат действовать.

```
ydb [global options...] scheme permissions clear [options...] <path>
```

Параметры:
`<path>` — полный путь от корня кластера до объекта, права на который необходимо отозвать.

Например, если над состоянием базы данных из предыдущего примера `list` выполнить команду:

```
ydb scheme permissions clear '/Root/db1/MyApp'
```

И затем заново выполнить команду `list` на объект `/Root/db1/MyApp`, то получим следующий результат:

```
Owner: root

Permissions:
none

Effective permissions:
USERS:ydb.database.connect
METADATA-READERS:ydb.generic.list
DATA-READERS:ydb.granular.select_row
DATA-WRITERS:ydb.tables.modify
DDL-ADMINS:ydb.granular.create_directory,ydb.granular.write_attributes,ydb.granular.create_table,ydb.granula
r.remove_schema,ydb.granular.alter_schema
ACCESS-ADMINS:ydb.access.grant
DATABASE-ADMINS:ydb.generic.manage
```

Обратите внимание, что секция `Permissions` теперь пуста. То есть все права на данный объект были отозваны. Также произошли изменения в содержании секции `Effective permissions`: в ней более не указаны права, которые были выданы непосредственно на объект `/Root/db1/MyApp`.

chown

Команда `chown` позволяет сменить владельца для схемного объекта.

Синтаксис команды:

```
ydb [connection options] scheme permissions chown [options...] <path> <owner>
```

Параметры:
`<path>` — полный путь от корня кластера до объекта, права на который необходимо модифицировать;
`<owner>` — имя нового владельца указанного объекта, может быть пользователем или группой.

Пример команды `chown`:

```
ydb scheme permissions chown '/Root/db1' testuser
```

> **Примечание**
>
> В текущей версии YDB есть ограничение, что только непосредственно пользователь - текущий владелец схемного объекта может сменить владельца.

clear-inheritance

Команда `clear-inheritance` позволяет запретить наследование разрешений для схемного объекта.

Синтаксис команды:

```
ydb [connection options] scheme permissions clear-inheritance [options...] <path>
```

Параметры:
`<path>` — полный путь от корня кластера до объекта, права на который необходимо модифицировать.

Пример команды `clear-inheritance`:

```
ydb scheme permissions clear-inheritance '/Root/db1'
```

set-inheritance

Команда `set-inheritance` позволяет включить наследование разрешений для схемного объекта.

Синтаксис команды:

```
ydb [connection options] scheme permissions set-inheritance [options...] <path>
```

Параметры:
`<path>` — полный путь от корня кластера до объекта, права на который необходимо модифицировать.

Пример команды `set-inheritance`:

```
ydb scheme permissions set-inheritance '/Root/db1'
```

Директории

База данных YDB поддерживает внутри иерархическую структуру директорий, в которых могут размещаться объекты БД.

YDB CLI поддерживает операции изменения структуры директорий, а также указание директории при обращении к объектам схемы.

Создание директории

Команда `scheme mkdir` создает директории:

```
ydb [connection options] scheme mkdir <path>
```

, где `[connection options]` — опции соединения с БД

В параметре `path` указывается относительный путь создаваемой директории от корня директорий базы данных. Будут созданы все директории на этом пути, которые не существовали до момента вызова команды.

Если конечная директория на пути уже существовала, то выполнение команды завершится успешно (код результата 0) с выдачей предупреждения о том, что никаких изменений не было произведено:

```
Status: SUCCESS
Issues:
<main>: Error: dst path fail checks, path: /<database>/<path>: path exist, request accepts it,
pathId: [OwnerId: <some>, LocalPathId: <some>], path type: EPathTypeDir, path state: EPathStateNoChanges
```

Также поддерживается синтаксис указания полного пути, начинающегося с символа `/`, который должен в этом случае содержать в начале путь базы данных, указанной в параметрах соединения, или с которой разрешаются операции через установленное соединение с кластером.

Примеры:

- Создание директории в корне базы данных

  ```
  ydb --profile quickstart scheme mkdir dir1
  ```

- Создание директорий на указанном пути от корня базы данных

  ```
  ydb --profile quickstart scheme mkdir dir1/dir2/dir3
  ```

Удаление директории

Команда `scheme rmdir` удаляет директорию:

```
ydb [global options...] scheme rmdir [options...] <path>
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.
- `path` — путь до удаляемой директории.

Посмотрите описание команды для удаления директории:

```
ydb scheme rmdir --help
```

Параметры подкоманды

Имя	Описание
`-r`, `--recursive`	Рекурсивное удаление директории вместе с дочерними объектами (поддиректориями, таблицами, топиками). При указании этого параметра по умолчанию будет запрошено подтверждение.
`-f`, `--force`	Не запрашивать никаких подтверждений.
`-i`	Запрашивать подтверждение на удаление каждого объекта.
`-I`	Однократно запросить подтверждение.
`--timeout <значение>`	Таймаут операции, мс.

При попытке удалить непустую директорию без указания параметра `-r` или `--recursive` команда не будет выполнена с выдачей ошибки:

```
Status: SCHEME_ERROR
Issues:
<main>: Error: path table fail checks, path: /<database>/<path>: path has children, request
doesn't accept it, pathId: [OwnerId: <some>, LocalPathId: <some>], path type:
EPathTypeDir, path state: EPathStateNoChanges, alive children: <count>
```

Примеры

- Удаление пустой директории:

```
ydb scheme rmdir dir1
```

- Удаление пустой директории с запросом подтверждения:

```
ydb scheme rmdir -I dir1
```

- Рекурсивное удаление непустой директории с запросом подтверждения:

```
ydb scheme rmdir -r dir1
```

- Рекурсивное удаление непустой директории без запроса подтверждения:

```
ydb scheme rmdir -rf dir1
```

- Рекурсивное удаление непустой директории с запросом подтверждения на каждый объект:

```
ydb scheme rmdir -ri dir1
```

Использование директорий в других командах CLI

Во всех командах CLI, в которые передается параметром имя объекта, оно может быть указано с директорией, например в `scheme describe`:

```
ydb --profile quickstart scheme describe dir1/table_a
```

Команда `scheme ls` поддерживает передачу пути к директории в качестве параметра:

```
ydb --profile quickstart scheme ls dir1/dir2
```

Использование директорий в YQL

Имена объектов в запросах на языке YQL могут содержать путь к директории, где находится этот объект. Этот путь будет конкатенирован с базовым путем, задаваемым прагмой `TablePathPrefix`. Если прагма не указана, имя объекта разрешается относительно корня базы данных.

Неявное создание директорий при импорте

При выполнении команды импорта данных создается дерево директорий по образцу каталога, из которого выполняется импорт.

Создание и удаление вторичных индексов

Важно

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Команда `table index` позволяет создавать и удалять вторичные индексы:

```
ydb [connection options] table index [subcommand] [options]
```

, где `[connection options]` — опции соединения с БД

Также добавить или удалить вторичный индекс можно с помощью директив ADD INDEX и DROP INDEX операции YQL ALTER TABLE.

О назначении и применении вторичных индексов при разработке приложений можно прочитать в статье Вторичные индексы.

Создание вторичного индекса

Создание вторичного индекса выполняется командой `table index add`:

```
ydb [connection options] table index add <sync-async> <table> \
  --index-name STR --columns STR [--cover STR]
```

Параметры:

`<sync-async>` : Тип вторичного индекса. Укажите `global-sync` для построения индекса с синхронным обновлением или `global-async` для индекса с асинхронным обновлением.

`<table>`: Путь и имя таблицы, для которой выполняется построение индекса

`--index-name STR`: Обязательный параметр, в котором задается имя индекса. Рекомендуется определять такие имена индексов, чтобы по ним можно было понять, какие колонки в них включены. Имена индексов уникальны в контексте таблицы.

`--columns STR`: Обязательный параметр, в котором определяются состав и порядок включения колонок в ключ индекса. Перечисляются имена колонок через запятую, без пробелов. Ключ индекса будет состоять из этих колонок с добавлением колонок первичного ключа таблицы.

`--cover STR`: Необязательный параметр, в котором определяется состав покрывающих колонок индекса. Их значения не будут включены в состав ключа индекса, но будут скопированы в записи в индексе для получения их значений при поиске по индексу без необходимости обращения к таблице.

В результате успешного исполнения команды запускается фоновая операция построения индекса, и в выделенном псевдографикой поле `id` возвращается идентификатор операции для дальнейшего получения информации о ее статусе командой `operation get`. Незаконченное построение индекса может быть прервано командой `operation cancel`.

После того, как построение индекса завершено (успешно или прервано), запись об операции построения может быть удалена командой `operation forget`.

Получить информацию о статусе всех операций построения индекса можно командой `operation list buildindex`.

Примеры

Примечание

В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Добавление синхронного индекса по колонке `air_date` в созданную ранее таблицу `episodes`:

```
ydb -p quickstart table index add global-sync episodes \
  --index-name idx_aired --columns air_date
```

Добавление асинхронного индекса по колонкам `release_date` и `title`, и с копированием в индекс значения колонки `series_info`, для созданной ранее таблицы `series`:

```
ydb -p quickstart table index add global-async series \
  --index-name idx_rel_title --columns release_date,title --cover series_info
```

Вывод (id операции при фактическом запуске будет другой):

```
┌──────────────────────────────────────┬───────┬────────┐
| id                                   | ready | status |
├──────────────────────────────────────┼───────┼────────┤
| ydb://buildindex/7?id=2814749869 | false |        |
└──────────────────────────────────────┴───────┴────────┘
```

Получение информации о статусе операции (подставьте фактический id операции):

```
ydb -p quickstart operation get ydb://buildindex/7?id=281474976866869
```

Возвращаемое значение:

```
┌──────────────────────────────────────┬────────┬─────────┬───────┬──────────┬──────────────────┬────────────┐
| id                                   | ready | status  | state | progress | table            | index      |
├──────────────────────────────────────┼────────┼─────────┼───────┼──────────┼──────────────────┼────────────┤
| ydb://buildindex/7?id=2814749869 | true  | SUCCESS | Done  | 100.00%  | /local/episodes | idx_aired |
└──────────────────────────────────────┴────────┴─────────┴───────┴──────────┴──────────────────┴────────────┘
```

Удаление информации о построении индекса (подставьте фактический id операции):

```
ydb -p quickstart operation forget ydb://buildindex/7?id=2814749869
```

Удаление индекса

Удаление индекса выполняется командой `table index drop`:

```
ydb [connection options] table index drop <table> --index-name STR
```

Пример

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Удаление индекса `idx_aired` с таблицы episodes, построенного в примере создания индекса выше:

```
ydb -p quickstart table index drop episodes --index-name idx_aired
```

Переименование вторичного индекса

Переименование вторичного индекса выполняется командой `table index rename`:

```
ydb [connection options] table index rename <table> --index-name STR --to STR
```

Если индекс с новым именем существует, команда вернет ошибку.

Чтобы атомарно заменить существующий индекс, выполните команду переименования с параметром `--replace`:

```
ydb [connection options] table index rename <table> --index-name STR --to STR --replace
```

Пример

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Переименование индекса `idx_aired` с таблицы episodes, построенного в примере создания индекса выше:

```
ydb -p quickstart table index rename episodes --index-name idx_aired --to idx_aired_renamed
```

Копирование таблиц

С помощью подкоманды `tools copy` вы можете создать копию таблицы или нескольких таблиц БД. При копировании исходная таблица остается на месте, копия содержит все данные исходной таблицы.

Общий вид команды:

```
ydb [global options...] tools copy [options...]
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.

Посмотрите описание команды для копирования таблицы:

```
ydb tools copy --help
```

Параметры подкоманды

Имя параметра	Описание параметра
`--timeout`	Время, в течение которого должна быть выполнена операция на сервере.
`--item <свойство>=<значение>,...`	Свойства операции. Параметр может быть указан несколько раз, если необходимо выполнить копирование нескольких таблиц в одной транзакции. Обязательные свойства: • `destination`, `dst`, `d` — путь таблицы-назначения. Если путь назначения содержит директории, они должны быть созданы заранее. Таблица с именем назначения не должна существовать. • `source`, `src`, `s` — путь таблицы-источника. Необязательные свойства: • `omit-indexes` — исключить индексы при копировании таблицы. Значение `true`: индексы не копируются, `false` (по умолчанию): индексы копируются.

Примеры

Примечание

В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Создайте в базе данных директорию `backup`:

```
ydb -p quickstart scheme mkdir backup
```

Скопируйте таблицу `series` в таблицу `series-v1`, таблицу `seasons` в `seasons-v1`, таблицу `episodes` в `episodes-v1` директории `backup`:

```
ydb -p quickstart tools copy --item destination=backup/series-v1,source=series --item destination=backup/seas
ons-v1,source=seasons --item destination=backup/episodes-v1,source=episodes
```

Посмотрите листинг объектов директории `backup`:

```
ydb -p quickstart scheme ls backup
```

Результат:

```
episodes-v1  seasons-v1  series-v1
```

Переименование строковых таблиц

С помощью подкоманды `tools rename` вы можете переименовать одну или несколько таблиц одновременно, перенести таблицу в другую директорию в пределах той же БД, заменить одну таблицу другой в рамках одной транзакции.

Общий вид команды:

```
ydb [global options...] tools rename [options...]
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.

Посмотрите описание команды для переименования таблицы:

```
ydb tools rename --help
```

Параметры подкоманды

Один запуск команды `tools rename` выполняет одну транзакцию переименования, которая может включать одну или несколько операций переименования разных таблиц.

Имя параметра	Описание параметра
`--item <свойство>=<значение>,...`	Описание операции переименования. Может быть указан несколько раз, если необходимо выполнить несколько операций переименования в одной транзакции. Обязательные свойства: • `source`, `src`, `s` — путь до таблицы источника. • `destination`, `dst`, `d` — путь до таблицы назначения. Если путь назначения содержит директории, они должны быть созданы заранее. Дополнительные свойства: • `replace`, `force` — перезаписывать таблицу назначения. Если значение `True`, то таблица назначения будет перезаписана, а существующие в ней данные удалены. `False` — если таблица назначения существует, то будет выдана ошибка, и транзакция переименования будет целиком откачена. Значение по умолчанию: `False`.
`--timeout <значение>`	Таймаут операции, мс.

При включении нескольких операций переименования в один вызов `tools rename` они исполняются в заданном порядке, но в рамках одной транзакции, что позволяет ротировать таблицу под нагрузкой без потери данных: первой операцией идет переименование рабочей таблицы в резервную, второй -- переименование новой таблицы в рабочую.

Примеры

- Переименование одной таблицы:

  ```
  ydb tools rename --item src=old_name,dst=new_name
  ```

- Переименование нескольких таблиц в одной транзакции:

  ```
  ydb tools rename \
    --item source=new-project/main_table,destination=new-project/episodes \
    --item source=new-project/second_table,destination=new-project/seasons \
    --item source=new-project/third_table,destination=new-project/series
  ```

- Перемещение таблиц в другую директорию:

  ```
  ydb tools rename \
    --item source=new-project/main_table,destination=cinema/main_table \
    --item source=new-project/second_table,destination=cinema/second_table \
    --item source=new-project/third_table,destination=cinema/third_table
  ```

- Замена таблицы

  ```
  ydb tools rename \
    --item replace=True,source=pre-prod-project/main_table,destination=prod-project/main_table
  ```

- Ротация таблицы

  ```
  ydb tools rename \
    --item source=prod-project/main_table,destination=prod-project/main_table.backup \
    --item source=pre-prod-project/main_table,destination=prod-project/main_table
  ```

Установка параметров TTL

С помощью подкоманды `table ttl set` вы можете установить TTL для указанной таблицы.

Общий вид команды:

```
ydb [global options...] table ttl set [options...] <table path>
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.
- `table path` — путь таблицы.

Посмотрите описание команды установки TTL:

```
ydb table ttl set --help
```

Параметры подкоманды

Имя	Описание
`--column`	Имя колонки, которая будет использована для вычисления времени жизни строк. Колонка должна иметь тип числовой или дата и время. В случае числового типа значение будет интерпретироваться как время, прошедшее с начала эпохи Unix. Единицы измерения должны быть заданы в параметре `--unit`.
`--expire-after`	Дополнительное время до удаления, которое должно пройти после истечения времени жизни строки. Указывается в секундах. Значение по умолчанию — `0`.
`--unit`	Единицы измерения значений колонки, которая указана в параметре `--column`. Обязателен, если колонка имеет числовой тип. Возможные значения: • `seconds (s, sec)` — секунды; • `milliseconds (ms, msec)` — миллисекунды; • `microseconds (us, usec)` — микросекунды; • `nanoseconds (ns, nsec)` — наносекунды.
`--run-interval`	Интервал запуска операции удаления строк с истекшим TTL. Указывается в секундах. Наименьший допустимый интервал составляет 15 минут (900 секунд). Значение по умолчанию (`3600`) оптимально для старта работы с таблицей с установленным TTL. При выборе значения `--run-interval` важно учитывать баланс между задержкой удаления просроченных данных и дополнительной нагрузкой на кластер. Слишком маленький интервал приводит к частому запуску фоновых задач удаления и может заметно увеличить нагрузку на CPU и I/O, особенно при большом объёме данных. Слишком большой интервал, наоборот, задерживает физическое удаление строк: просроченные данные дольше занимают место на диске и могут продолжать участвовать в сканированиях. При любом значении `--run-interval` следует ожидать, что на запрос к таблице могут вернуться строки с истекшим `TTL`.

Примеры

Примечание

В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Установите TTL для таблицы `series`

```
ydb -p quickstart table ttl set \
  --column createtime \
  --expire-after 3600 \
  --run-interval 1200 \
  series
```

Сброс параметров TTL

С помощью подкоманды `table ttl reset` вы можете выключить TTL для указанной таблицы.

Общий вид команды:

```
ydb [global options...] table ttl reset <table path>
```

- `global options` — глобальные параметры.
- `table path` — путь таблицы.

Посмотрите описание команды выключения TTL:

```
ydb table ttl reset --help
```

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Выключите TTL для таблицы `series`:

```
ydb -p quickstart table ttl reset \
  series
```

Удаление таблицы

С помощью подкоманды `table drop` вы можете удалить указанную таблицу.

Общий вид команды:

```
ydb [global options...] table drop [options...] <table path>
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.
- `table path` — путь таблицы.

Посмотрите описание команды удаления таблицы:

```
ydb table drop --help
```

Параметры подкоманды

Имя	Описание
`--timeout`	Время, в течение которого должна быть выполнена операция на сервере.

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Удалите таблицу `series`:

```
ydb -p quickstart table drop series
```

Генерация скрипта создания таблицы

Команда `ydb tools infer csv` позволяет на основе имеющегося CSV-файла с данными сгенерировать скрипт для создания таблицы.

Общий вид команды:

```
ydb [global options...] tools infer csv [options...] <input files...>
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.

Получить описание параметров команды можно с помощью опции `--help`:

```
ydb tools infer csv --help
```

Параметры подкоманды

Имя параметра	Описание параметра
`-p, --path`	Путь в базе данных, по которому должна быть создана новая таблица. Значение по умолчанию: `table`.
`--columns`	Список имен колонок таблицы, разделенных запятыми.
`--gen-columns`	Имена колонок таблицы необходимо сгенерировать автоматически (column1, column2, ...).
`--header`	Имена колонок таблицы необходимо считать из первой строчки CSV-файла.
`--rows-to-analyze`	Количество первых строк CSV-файла, подлежащих анализу для автоматического определения типов колонок. `0` - будут прочитаны и проанализированы все строки из файла. Значение по умолчанию: `500 000`.
`--execute`	Выполнить создание таблицы по результатам генерации скрипта.

Примечание

Если ни одна из опций `--columns`, `--gen-names` или `--header` явно не указана, то применяется следующий алгоритм:

Берётся первая строка из файла, и указанные в ней значения проверяются на следующие условия:

- значения соответствуют требованиям к наименованию колонок;
- типы значений в первой строке отличаются от типов значений данных из других строк файла

При выполнении обоих условий значения из первой строки файла используются в качестве имён колонок таблицы. В противном случае, имена колонок генерируются автоматически (как `column1`, `column2`, и т.д.). Подробнее см. пример ниже.

Алгоритм вывода типов колонок

Для каждой колонки команда определяет наименее общий тип, подходящий для всех её значений. Наиболее универсальным типом считается строка (`Text`): если среди числовых значений встречается хотя бы одно строковое (например, `abc`), тип всей колонки будет определён как `Text`.

Все целые числа относятся к типу `Int64`, если они укладываются в его диапазон. При превышении границ используется тип `Double`.

Числа с плавающей запятой также определяются как `Double`.

Текущее ограничение

В качестве первичного ключа таблицы выбирается первая колонка. При необходимости можно изменить состав колонок первичного ключа на требуемый.

Выбор первичного ключа

Примеры

Примечание

В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Имена колонок заданы в первой строке CSV-файла

В этом примере ни одна из опций `--columns`, `--gen-names` или `--header` явно не указана.
Значения `key` и `value` из первой строки файла соответствуют требованиям к именам колонок, а их типы (`Text` и `Text`) не совпадают с типами в остальных строках (`Int64` и `Text`). Таким образом, `key` и `value` будут выбраны в качестве имен колонок.

```
$ cat data_with_header.csv
key,value
123,abc
456,def
```

```
ydb tools infer csv data_with_header.csv
CREATE TABLE table (
    key Int64,
    value Text,
    PRIMARY KEY (key) -- First column is chosen. Probably need to change this.
)
WITH (
    STORE = ROW -- or COLUMN
    -- Other useful table options to consider:
    --, AUTO_PARTITIONING_BY_SIZE = ENABLED
    --, AUTO_PARTITIONING_BY_LOAD = ENABLED
    --, UNIFORM_PARTITIONS = 100 -- Initial number of partitions
    --, AUTO_PARTITIONING_MIN_PARTITIONS_COUNT = 100
    --, AUTO_PARTITIONING_MAX_PARTITIONS_COUNT = 1000
);
```

Примечание

При генерации скрипта автоматически добавляется блок `WITH` с дополнительными опциями для создаваемой таблицы. Все опции, кроме `STORE`, имеют значения по умолчанию и закомментированы.
Вы можете указать требуемые значения нужных дополнительных опций самостоятельно.

Имена колонок в первой строке CSV-файла, используется опция `--header`

В этом примере значения `key` и `value` в первой строке совпадают с типами данных (`Text`) в остальных строках. Поэтому без опции `--header` команда не будет использовать первую строку как имена колонок, а сгенерирует их автоматически. Чтобы использовать первую строку как имена колонок в таком случае, явно укажите опцию `--header`:

```
$ cat data_with_header_text.csv
key,value
aaa,bbb
ccc,ddd

ydb tools infer csv data_with_header_text.csv --header
CREATE TABLE table (
    key Text,
    value Text,
    PRIMARY KEY (key) -- First column is chosen. Probably need to change this.
)
WITH (
    STORE = ROW -- or COLUMN
    -- Other useful table options to consider:
    --, AUTO_PARTITIONING_BY_SIZE = ENABLED
    --, AUTO_PARTITIONING_BY_LOAD = ENABLED
    --, UNIFORM_PARTITIONS = 100 -- Initial number of partitions
    --, AUTO_PARTITIONING_MIN_PARTITIONS_COUNT = 100
    --, AUTO_PARTITIONING_MAX_PARTITIONS_COUNT = 1000
);
```

Явное указание списка имен колонок

```
cat ~/data_no_header.csv
123,abc
456,def

ydb tools infer csv -p newtable ~/data_no_header.csv --columns my_key,my_value
CREATE TABLE newtable (
    my_key Int64,
    my_value Text,
    PRIMARY KEY (my_key)
)
WITH (
    STORE = ROW -- or COLUMN
    -- Other useful table options to consider:
    --, AUTO_PARTITIONING_BY_SIZE = ENABLED
    --, AUTO_PARTITIONING_BY_LOAD = ENABLED
    --, UNIFORM_PARTITIONS = 100 -- Initial number of partitions
    --, AUTO_PARTITIONING_MIN_PARTITIONS_COUNT = 100
    --, AUTO_PARTITIONING_MAX_PARTITIONS_COUNT = 1000
);
```

Автоматическая генерация имён колонок

```
cat ~/data_no_header.csv
123,abc
456,def

ydb tools infer csv -p newtable ~/data_no_header.csv --gen-columns
CREATE TABLE newtable (
    column1 Int64,
    column2 Text,
    PRIMARY KEY (f0) -- First column is chosen. Probably need to change this.
)
WITH (
    STORE = ROW -- or COLUMN
```

```
    -- Other useful table options to consider:
    --, AUTO_PARTITIONING_BY_SIZE = ENABLED
    --, AUTO_PARTITIONING_BY_LOAD = ENABLED
    --, UNIFORM_PARTITIONS = 100 -- Initial number of partitions
    --, AUTO_PARTITIONING_MIN_PARTITIONS_COUNT = 100
    --, AUTO_PARTITIONING_MAX_PARTITIONS_COUNT = 1000
);
```

Выполнение сгенерированного запроса с помощью опции `--execute`

В этом примере запрос `CREATE TABLE` выполняется сразу после генерации.

```
$ cat data_with_header.csv
key,value
123,abc
456,def

ydb -p quickstart tools infer csv data_with_header.csv --execute
Executing request:

CREATE TABLE table (
    key Int64,
    value Text,
    PRIMARY KEY (key) -- First column is chosen. Probably need to change this.
)
WITH (
    STORE = ROW -- or COLUMN
    -- Other useful table options to consider:
    --, AUTO_PARTITIONING_BY_SIZE = ENABLED
    --, AUTO_PARTITIONING_BY_LOAD = ENABLED
    --, UNIFORM_PARTITIONS = 100 -- Initial number of partitions
    --, AUTO_PARTITIONING_MIN_PARTITIONS_COUNT = 100
    --, AUTO_PARTITIONING_MAX_PARTITIONS_COUNT = 1000
);

Query executed successfully.
```

Получение плана исполнения запроса и AST

YDB предоставляет два типа планов запроса: логический план и план исполнения. Логический план лучше подходит для анализа сложных запросов с большим количеством операторов join. План исполнения более детализирован: он дополнительно показывает стадии распределённого плана и коннекторы между ними, что делает его более удобным для анализа простых OLTP-запросов.

Планы исполнения запроса

Получить план исполнения через YDB CLI можно с помощью команды `explain`:

```
ydb -p <profile_name> table query explain \
  -q "SELECT season_id, episode_id, title
  FROM episodes
  WHERE series_id = 1
  AND season_id > 1
  ORDER BY season_id, episode_id
  LIMIT 3"
```

Полученный результат:

```
Query Plan:
ResultSet
└──Limit (Limit: 3)
    └──<Merge>
      └──TopSort (Limit: 3, TopSortBy: )
      └──Filter (Predicate: Exist(item.season_id) And Exist(item.series_id))
      └──TablePointLookup (ReadRange: [series_id (1), season_id (1, +∞), episode_id (-∞, +∞)], ReadColumns:
[episode_id, season_id, title, series_id], Table: episodes)
         Tables: [episodes]
```

Дополнительно к плану исполнения запроса вы можете получить AST (абстрактное синтаксическое дерево).
Раздел AST содержит представление на внутреннем языке miniKQL.

Для получения AST необходимо вызвать `explain` с флагом `--ast`:

```
ydb -p <profile_name> table query explain \
  -q "SELECT season_id, episode_id, title
  FROM episodes
  WHERE series_id = 1
  AND season_id > 1
  ORDER BY season_id, episode_id
  LIMIT 3" --ast
```

Полученный AST:

```
Query AST:
(
(let $1 (KqpTable '"episodes" '"72075186224045943:83859" '"" '1))
(let $2 '('"episode_id" '"season_id" '"title" '"series_id"))
(let $3 (Uint64 '1))
(let $4 (KqpRowsSourceSettings $1 $2 '() '((KqlKeyExc $3 $3) (KqlKeyInc $3))))
(let $5 (Uint64 '"3"))
(let $6 (DqPhyStage '((DqSource (DataSource '"KqpReadRangesSource") $4)) (lambda '($12) (block '(
  (let $13 (Bool 'true))
  (return (FromFlow (TopSort (OrderedMap (OrderedFilter (ToFlow $12) (lambda '($14) (And (Exists (Member $14
'"season_id")) (Exists (Member $14 '"series_id")))))) (lambda '($15) (AsStruct '('"episode_id" (Member $15 '"e
pisode_id")) '('"season_id" (Member $15 '"season_id")) '('"title" (Member $15 '"title"))))) $5 '($13 $13) (la
mbda '($16) '((Member $16 '"season_id") (Member $16 '"episode_id"))))))
))) '('('"_logical_id" '842) '('"_id" '"14388682-e03b28dd-60f91f84-d7cd002f"))))
(let $7 (DqCnMerge (TDqOutput $6 '0) '('('"season_id" '"Asc") '('"episode_id" '"Asc"))))
(let $8 (DqPhyStage '($7) (lambda '($17) (FromFlow (Take (ToFlow $17) $5))) '('('"_logical_id" '855) '('"_id"
'"295c0459-34d4b026-b467e71f-35402430"))))
(let $9 '('"season_id" '"episode_id" '"title"))
(let $10 (DqCnResult (TDqOutput $8 '0) $9))
(let $11 (OptionalType (DataType 'Uint64)))
(return (KqpPhysicalQuery '((KqpPhysicalTx '($6 $8) '($10) '() '('('"type" '"data")))) '((KqpTxResultBinding
(ListType (StructType '('"episode_id" $11) '('"season_id" $11) '('"title" (OptionalType (DataType 'Strin
g))))) '0 '0)) '('('"type" '"data_query"))))
)
```

Логический план запроса

Рассмотрим аналитический запрос Q18 из бенчмарка TPCH:

```
$p = (
  select
    p.p_brand as p_brand,
    p.p_type as p_type,
    p.p_size as p_size,
    ps.ps_suppkey as ps_suppkey
  from
    part as p
  join
    partsupp as ps
on
    p.p_partkey = ps.ps_partkey
```

```
where
    p.p_brand <> 'Brand#45'
    and p.p_type not like 'MEDIUM POLISHED%'
    and (p.p_size = 49 or p.p_size = 14 or p.p_size = 23 or p.p_size = 45 or p.p_size = 19 or p.p_size = 3 or
p.p_size = 36 or p.p_size = 9)
);

$s = (
  select
    s_suppkey
  from
    supplier
  where
    s_comment like "%Customer%Complaints%"
);

$j = (
  select
    p.p_brand as p_brand,
    p.p_type as p_type,
    p.p_size as p_size,
    p.ps_suppkey as ps_suppkey
  from
    $p as p
  left only join
    $s as s
  on
    p.ps_suppkey = s.s_suppkey
);

select
    j.p_brand as p_brand,
    j.p_type as p_type,
    j.p_size as p_size,
    count(distinct j.ps_suppkey) as supplier_cnt
from
    $j as j
group by
    j.p_brand,
    j.p_type,
    j.p_size
order by
    supplier_cnt desc,
    p_brand,
    p_type,
    p_size
;
```

Получить логический план выполнения можно командой:

```
ydb -p <profile_name> sql --explain -f q18.sql
```

В результате получится логический план запроса, состоящий из операторов плана и различных предсказаний оптимизатора:

```
Query Plan:
┌─────────────┬─────────────┬─────────────┬──────────────────────────────────────────────────────────────┐
│ E-Cost      │ E-Rows      │ E-Size      │ Operation                                                    │
├─────────────┼─────────────┼─────────────┼──────────────────────────────────────────────────────────────┤
│             │             │             │ ┌> ResultSet                                                 │
│             │             │             │ └─┬> Sort ([row.supplier_cnt,row.p_brand,row.p_type,r        │
│             │             │             │ ow.p_size])                                                  │
│             │             │             │   └─┬> Aggregate (Phase: Final)                              │
│             │             │             │     └─┬> HashShuffle (KeyColumns: ["part.p_brand","pa        │
│             │             │             │ rt.p_size","part.p_type"], HashFunc: "HashV2")               │
│             │             │             │       └─┬> Aggregate (GroupBy: [part.p_brand,part.p_s        │
│             │             │             │ ize,part.p_type], Aggregation: {Inc(_yql_agg_0)}, Pha         │
│             │             │             │ se: Intermediate)                                            │
│             │             │             │         └─┬> Aggregate (Phase: Final)                        │
│             │             │             │           └─┬> HashShuffle (KeyColumns: ["part.p_bran        │
│             │             │             │ d","part.p_size","part.p_type","partsupp.ps_suppkey"]        │
│             │             │             │ , HashFunc: "HashV2")                                        │
│             │             │             │             └─┬> Aggregate (GroupBy: [part.p_brand,pa        │
│             │             │             │ rt.p_size,part.p_type,partsupp.ps_suppkey], Aggregati        │
│             │             │             │ on: state, Phase: Intermediate)                              │
│ 4.350e+08   │ 40000000    │ 3.421e+09   │               └─┬> LeftOnlyJoin (Grace) (partsupp.ps_        │
│             │             │             │ suppkey = supplier.s_suppkey)                                │
│             │             │             │                 ├─┬> HashShuffle (KeyColumns: ["parts        │
│             │             │             │ upp.ps_suppkey"], HashFunc: "ColumnShardHashV1")             │
│ 3.135e+08   │ 40000000    │ 2.331e+09   │                 │ └─┬> InnerJoin (Grace) (part.p_part        │
│             │             │             │ key = partsupp.ps_partkey)                                   │
│ 0           │ 9000000     │ 2.869e+08   │                 │   ├─┬> Filter (Contains)                     │
│ 0           │ 9000000     │ 2.869e+08   │                 │   │ └─┬> Filter ((p_brand != "Brand        │
│             │             │             │ #33") AND (NOT p_type LIKE "PROMO POLISHED%"), Pushdo        │
│             │             │             │ wn: True)                                                    │
│ 0           │ 20000000    │ 6.375e+08   │                 │   │   └──> TableFullScan (Table: pa        │
│             │             │             │ rt, ReadColumns: ["p_partkey (-∞, +∞)","p_brand","p_s        │
│             │             │             │ ize","p_type"])                                              │
│             │             │             │                 │   └─┬> HashShuffle (KeyColumns: ["p        │
│             │             │             │ s_partkey"], HashFunc: "ColumnShardHashV1")                  │
│ 0           │ 80000000    │ 2.113e+09   │                 │     └──> TableFullScan (Table: part        │
```

```
|           |          |           | supp, ReadColumns: ["ps_partkey (-∞, +∞)","ps_suppkey |
|           |          |           |  (-∞, +∞)"])                                          |
| 0         | 500000   | 13621431  |                 └─┬> Filter (Apply, Pushdown: True)   |
| 0         | 1000000  | 27242862  |                   └──> TableFullScan (Table: supplier |
|           |          |           | , ReadColumns: ["s_suppkey (-∞, +∞)","s_comment"])    |
└───────────┴──────────┴───────────┴──────────────────────────────────────────────────────┘
```

В этом плане показано дерево операторов YDB и выведены три предсказания оптимизатора для каждого оператора:

- `E-Cost:` оценка стоимости текущего оператора и всех его входов;
- `E-Rows`: оценка количества записей на выходе текущего оператора;
- `E-Size`: оценка объёма результата оператора в байтах.

Кроме предсказаний оптимизатора, можно также получить реальные значения при выполнении запроса:

- `A-Cpu`: общее время работы CPU для текущего оператора;
- `A-Rows`: фактическое количество записей на выходе текущего оператора.

Статистика выполнения не всегда доступна для каждого оператора, так как в настоящий момент она собирается по стадиям выполнения. В одной стадии может вычисляться сразу несколько операторов.

Для получения логического плана запроса со статистикой выполнения выполните следующую команду:

```
ydb -p <profile_name> sql --explain-analyze --analyze -f q18.sql
```

Получится следующий логический план запроса:

```
┌─────────┬────────────┬────────────┬────────────┬────────────┬──────────────────────────────────────────────────────────────────────────────
──────────┐
| A-Cpu | A-Rows   | E-Cost     | E-Rows   | E-Size     | Operation
|
├─────────┼────────────┼────────────┼────────────┼────────────┼──────────────────────────────────────────────────────────────────────────────
──────────┤
|       |          |            |          |            | ┌> ResultSet
|
| 934   | 27840    |            |          |            | └─┬> Sort (A-SelfCpu: 15.706, A-Size: 936768, [row.su
pplier_cnt, |
|       |          |            |          |            | row.p_brand,row.p_type,row.p_size])
|
|       | 27840    |            |          |            |   └─┬> Aggregate (Phase: Final)
|
|       |          |            |          |            |     └─┬> HashShuffle (KeyColumns: ["part.p_brand","pa
rt.p_size", |
|       |          |            |          |            | "part.p_type"], HashFunc: "HashV2")
|
| 918   | 3431671  |            |          |            |       └─┬> Aggregate (GroupBy: [part.p_brand,part.p_s
ize,part.p_ |
|       |          |            |          |            | type], Aggregation: {Inc(_yql_agg_0)}, A-SelfCpu: 13
9.139, Phase |
|       |          |            |          |            | : Intermediate, A-Size: 112058009)
|
|       |          |            |          |            |         └─┬> Aggregate (Phase: Final)
|
|       |          |            |          |            |           └─┬> HashShuffle (KeyColumns: ["part.p_bran
d","part.p_ |
|       |          |            |          |            | size","part.p_type","partsupp.ps_suppkey"], HashFunc:
"HashV2")  |
| 779   | 11865156 |            |          |            |             └─┬> Aggregate (GroupBy: [part.p_brand,pa
rt.p_size,p |
|       |          |            |          |            | art.p_type,partsupp.ps_suppkey], Aggregation: state,
A-SelfCpu:  |
|       |          |            |          |            | 194.29, Phase: Intermediate, A-Size: 410786277)
|
|       | 11867631 | 4.350e+08  | 40000000 | 3.421e+09  |               └─┬> LeftOnlyJoin (Grace) (partsupp.ps_
suppkey = s |
|       |          |            |          |            | upplier.s_suppkey)
|
|       |          |            |          |            |                 ├─┬> HashShuffle (KeyColumns: ["parts
upp.ps_supp |
|       |          |            |          |            | key"], HashFunc: "ColumnShardHashV1")
|
| 584   | 11873200 | 3.135e+08  | 40000000 | 2.331e+09  |                 | └─┬> InnerJoin (Grace) (A-SelfCpu:
308.14, par |
|       |          |            |          |            | t.p_partkey = partsupp.ps_partkey, A-Size: 505055811)
|
| 65    | 2968300  | 0          | 9000000  | 2.869e+08  |                 |   ├─┬> Filter (Contains, A-SelfCpu:
65.311, A- |
|       |          |            |          |            | Size: 105580755)
|
|       |          | 0          | 9000000  | 2.869e+08  |                 |   | └─┬> Filter ((p_brand != "Brand
#33") AND ( |
|       |          |            |          |            | NOT p_type LIKE "PROMO POLISHED%"), Pushdown: True)
|
|       | 18560313 | 0          | 20000000 | 6.375e+08  |                 |   |   └──> TableFullScan (Table: pa
rt, ReadCol |
|       |          |            |          |            | umns: ["p_partkey (-∞, +∞)","p_brand","p_size","p_typ
e"], A-Size |
|       |          |            |          |            | : 1060176372)
|
|       |          |            |          |            |                 |   └─┬> HashShuffle (KeyColumns: ["p
```

```
s_partkey"] |
|       |          |           |          |           | , HashFunc: "ColumnShardHashV1")
|
| 211   | 80000000 | 0         | 80000000 | 2.113e+09 |                 |     └──> TableFullScan (Table: part
supp, A-Sel |
|       |          |           |          |           | fCpu: 210.822, ReadColumns: ["ps_partkey (-∞, +∞)","p
s_suppkey ( |
|       |          |           |          |           | -∞, +∞)"], A-Size: 1283276800)
|
| 0     | 479      | 0         | 500000   | 13621431  |                 └─┬> Filter (Apply, A-SelfCpu: 0.27,
Pushdown: T |
|       |          |           |          |           | rue, A-Size: 4937)
|
|       | 479      | 0         | 1000000  | 27242862  |                   └──> TableFullScan (Table: supplie
r, ReadColum |
|       |          |           |          |           | ns: ["s_suppkey (-∞, +∞)","s_comment"], A-Size: 7885)
|
└───────┴──────────┴───────────┴──────────┴───────────┴──────────────────────────────────────────────────────────
──────────┘
```

В этом плане, кроме операторов и предсказаний оптимизатора уже присутствует статистика выполнения - `A-Cpu` и `A-Rows`.

AST

Потоковое чтение строковой таблицы

> **Важно**
>
> Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Чтобы прочитать снапшот таблицы целиком, используйте подкоманду `read`. Данные передаются в виде стрима, что позволяет прочитать таблицу произвольного размера.

Прочитайте данные:

```
ydb table read episodes \
  --ordered \
  --limit 5 \
  --columns series_id,season_id,episode_id,title
```

Где :

- `--ordered` — упорядочить читаемые записи по ключу.
- `--limit` — ограничить количество читаемых записей.
- `--columns` — колонки, значения которых следует читать (по умолчанию читаются все колонки) в формате CSV.

Результат:

```
┌───────────┬───────────┬────────────┬──────────────────────────────┐
| series_id | season_id | episode_id | title                        |
├───────────┼───────────┼────────────┼──────────────────────────────┤
| 1         | 1         | 1          | "Yesterday's Jam"            |
├───────────┼───────────┼────────────┼──────────────────────────────┤
| 1         | 1         | 2          | "Calamity Jen"               |
├───────────┼───────────┼────────────┼──────────────────────────────┤
| 1         | 1         | 3          | "Fifty-Fifty"                |
├───────────┼───────────┼────────────┼──────────────────────────────┤
| 1         | 1         | 4          | "The Red Door"               |
├───────────┼───────────┼────────────┼──────────────────────────────┤
| 1         | 1         | 5          | "The Haunting of Bill Crouse" |
└───────────┴───────────┴────────────┴──────────────────────────────┘
```

Если вам нужно получить только количество прочитанных записей, используйте параметр `--count-only`:

```
ydb table read episodes \
  --columns series_id \
  --count-only
```

Результат:

```
70
```

Выгрузка и загрузка данных

YDB CLI содержит набор команд, предназначенных для выгрузки (экспорта) и загрузки (импорта) данных и описаний объектов схемы данных. Выгрузка данных может применяться как для создания резервных копий в целях последующего восстановления, так и в других целях.

- Файловая структура выгрузки, применяемая как при выгрузке в файловую систему, так и при выгрузке в S3-совместимое объектное хранилище.
- Выгрузка метаданных кластера в файловую систему с помощью `admin cluster dump`
- Загрузка метаданных кластера из файловой системы с помощью `admin cluster restore`
- Выгрузка данных и метаданных базы данных в файловую систему с помощью `admin database dump`
- Загрузка данных и метаданных базы данных из файловой системы с помощью `admin database restore`
- Выгрузка отдельных схемных объектов в файловую систему с помощью `tools dump`
- Загрузка отдельных схемных объектов из файловой системы с помощью `tools restore`
- Соединение и аутентификация при работе с S3-совместимым объектным хранилищем
- Выгрузка в S3-совместимое объектное хранилище `export s3`
- Загрузка из S3-совместимого объектного хранилища `import s3`
- Настройка NFS для выполнения резервного копирования
- Выгрузка в NFS `export nfs`
- Загрузка из NFS `import nfs`

Файловая структура выгрузки

Описанная ниже файловая структура применяется для выгрузки как в файловую систему, так и в S3-совместимое объектное хранилище. При работе с S3 в ключ объекта записывается путь к файлу, а директория выгрузки является префиксом ключа. В случае зашифрованного файла к нему добавляется расширение `.enc`.

Кластер

> **Примечание**
>
> Выгрузка кластера доступна только в файловую систему.

Кластеру соответствует директория в файловой структуре, в которой находятся:

- Директории, содержащие информацию о базах данных в кластере, за исключением:
  - Схемных объектов базы данных
  - Пользователей и групп базы данных, не являющихся администраторами базы данных
- Файл `permissions.pb`, содержащий информацию об ACL корня кластера и его владельце в формате text protobuf
- Файл `create_user.sql`, содержащий информацию о пользователях кластера в формате YQL
- Файл `create_group.sql`, содержащий информацию о группах кластера в формате YQL
- Файл `alter_group.sql`, содержащий информацию о вхождении пользователей в группы кластера в формате YQL

База данных

> **Примечание**
>
> Выгрузка в S3-совместимое объектное хранилище поддерживает только выгрузку схемных объектов базы данных.

Базе данных соответствует директория в файловой структуре, в которой находятся:

- Директории, содержащие информацию о схемных объектах базы данных, например, о таблицах
- Файл `database.pb`, содержащий информацию о настройках базы данных в формате text protobuf
- Файл `permissions.pb`, содержащий информацию об ACL базы данных и её владельце в формате text protobuf
- Файл `create_user.sql`, содержащий информацию о пользователях базы данных в формате YQL
- Файл `create_group.sql`, содержащий информацию о группах базы данных в формате YQL
- Файл `alter_group.sql`, содержащий информацию о вхождении пользователей в группы базы данных в формате YQL

Schema mapping

При выгрузке в S3-совместимое объектное хранилище с указанием общего префикса создаётся файл `SchemaMapping/mapping.json`. В этом файле содержится список путей выгружаемых объектов, отображённых в соответствующие префиксы S3 относительно общего префикса.

В зашифрованных выгрузках это отображение анонимизирует исходные имена объектов базы данных, а также хранит дополнительные данные, используемые для расшифровки.

Пример содержимого файла `SchemaMapping/mapping.json`:

```
{
    "exportedObjects": {
        "dir/table1": {
            "exportPrefix": "dir/table1"
        },
        "table2": {
            "exportPrefix": "table2"
        }
    }
}
```

Директории

Каждой директории в базе данных соответствует директория в файловой структуре. В каждой из них находится файл `permissions.pb`, содержащий информацию об ACL директории и её владельце в формате text protobuf. Иерархия директорий в файловой структуре соответствует иерархии директорий в базе данных. Если в какой-либо директории базы данных отсутствуют объекты (таблицы или поддиректории), то в файловой структуре такая директория содержит файл нулевого размера с именем `empty_dir`.

Таблицы

Каждой таблице в базе данных также соответствует одноименная директория в иерархии директорий файловой структуры, в которой находятся:

- Файл `scheme.pb`, содержащий информацию о структуре таблицы и её параметрах в формате text protobuf
- Файл `permissions.pb`, содержащий информацию об ACL таблицы и её владельце в формате text protobuf
- Один или несколько файлов `data_XX.csv`, содержащих данные таблицы в формате `csv`, где `XX` - порядковый номер файла. Выгрузка начинается с файла `data_00.csv`, каждый следующий файл создается при превышении размера текущего файла в 100MB
- Директории для описания потоков изменений. Имя директории соответствует названию потока изменений. В директории находятся:
  - Файл `changefeed_description.pb`, содержащий информацию о потоке изменений в формате text protobuf
  - Файл `topic_description.pb`, содержащий информацию о нижележащем топике в формате text protobuf

Топики

Каждому топику в базе данных также соответствует одноименная директория в иерархии директорий файловой структуры, в которой находится файл `create_topic.pb`, содержащий информацию о параметрах топика, настройках партиционирования и консьюмерах в формате text protobuf

Представления (view)

Каждому представлению в базе данных соответствует одноименная директория в иерархии директорий файловой структуры, в которой находятся:

- Файл `create_view.sql`, содержащий YQL-запрос создания представления
- Файл `permissions.pb`, содержащий информацию об ACL представления и его владельце в формате text protobuf

Асинхронные репликации

Каждой асинхронной репликации в базе данных соответствует одноименная директория в иерархии директорий файловой структуры, в которой находятся:

- Файл `create_async_replication.sql`, содержащий YQL-запрос создания асинхронной репликации
- Файл `permissions.pb`, содержащий информацию об ACL репликации и её владельце в формате text protobuf

Трансферы

Каждому трансферу в базе данных соответствует одноименная директория в иерархии директорий файловой структуры, в которой находятся:

- Файл `create_transfer.sql`, содержащий YQL-запрос создания трансфера
- Файл `permissions.pb`, содержащий информацию об ACL трансфера и его владельце в формате text protobuf

Внешние источники данных

Каждому внешнему источнику данных в базе данных соответствует одноименная директория в иерархии директорий файловой структуры, в которой находятся:

- Файл `create_external_data_source.sql`, содержащий YQL-запрос создания внешнего источника данных
- Файл `permissions.pb`, содержащий информацию об ACL внешнего источника данных и его владельце в формате text protobuf

Внешние таблицы

Каждой внешней таблице в базе данных соответствует одноименная директория в иерархии директорий файловой структуры, в которой находятся:

- Файл `create_external_table.sql`, содержащий YQL-запрос создания внешней таблицы
- Файл `permissions.pb`, содержащий информацию об ACL внешней таблицы и её владельце в формате text protobuf

Файлы с данными

Формат файлов с данными - `.csv`, одна строка соответствует одной записи в таблице, без строки с заголовками колонок. Для строк применяется представление в urlencoded формате. Например, строка файла для таблицы с колонками uint64 и utf8, содержащая число 1 и строку "Привет" соответственно, выглядит таким образом:

```
1,"%D0%9F%D1%80%D0%B8%D0%B2%D0%B5%D1%82"
```

Контрольные суммы

> **Примечание**
>
> Контрольные суммы файлов создаются только при выгрузке в S3-совместимое объектное хранилище.

YDB генерирует контрольную сумму для каждого файла выгрузки и сохраняет ее в соответствующем файле с суффиксом `.sha256`.

Контрольные суммы файлов можно проверить с помощью консольной утилиты `sha256sum`:

```
$ sha256sum -c scheme.pb.sha256
scheme.pb: OK
```

Примеры

Кластер

При выгрузке кластера с базыми данных `/Root/db1` и `/Root/db2` будет создана следующая структура файлов:

```
backup
├─ Root
|  ├─ db1
|  |  ├─ alter_group.sql
|  |  ├─ create_group.sql
|  |  ├─ create_user.sql
|  |  ├─ permissions.pb
|  |  └─ database.pb
|  └─ db2
|     ├─ alter_group.sql
|     ├─ create_group.sql
|     ├─ create_user.sql
|     ├─ permissions.pb
```

```
|    └─ database.pb
├─ alter_group.sql
├─ create_group.sql
├─ create_user.sql
└─ permissions.pb
```

Базы данных

При выгрузке базы данных с таблицей `episodes` будет создана следующая структура файлов:

```
├─ episodes
|    ├─ data00.csv
|    ├─ scheme.pb
|    └─ permissions.pb
├─ alter_group.sql
├─ create_group.sql
├─ create_user.sql
├─ permissions.sql
└─ database.pb
```

Таблицы

При выгрузке таблиц, созданных в разделе Быстрый старт с YDB Начала работы, будет создана следующая файловая структура:

```
├── episodes
|   ├── data_00.csv
|   ├── permissions.pb
|   └── scheme.pb
├── seasons
|   ├── data_00.csv
|   ├── permissions.pb
|   └── scheme.pb
└── series
    ├── data_00.csv
    ├── permissions.pb
    └── scheme.pb
    └── updates_feed
        └── changefeed_description.pb
        └── topic_description.pb
```

Содержимое файла `series/scheme.pb`:

```
columns {
  name: "series_id"
  type {
    optional_type {
      item {
        type_id: UINT64
      }
    }
  }
}
columns {
  name: "title"
  type {
    optional_type {
      item {
        type_id: UTF8
      }
    }
  }
}
columns {
  name: "series_info"
  type {
    optional_type {
      item {
        type_id: UTF8
      }
    }
  }
}
columns {
  name: "release_date"
  type {
    optional_type {
      item {
        type_id: UINT64
      }
    }
  }
}
primary_key: "series_id"
storage_settings {
  store_external_blobs: DISABLED
}
column_families {
  name: "default"
```

```
  compression: COMPRESSION_NONE
}
```

Содержимое файла `series/update_feed/changefeed_description.pb`:

```
name: "update_feed"
mode: MODE_UPDATES
format: FORMAT_JSON
state: STATE_ENABLED
```

Содержимое файла `series/update_feed/topic_description.pb`:

```
retention_period {
  seconds: 86400
}
consumers {
  name: "my_consumer"
  read_from {
  }
  attributes {
    key: "_service_type"
    value: "data-streams"
  }
}
```

Директории

При выгрузке пустой директории `series` будет создана следующая структура файлов:

```
└── series
    ├── permissions.pb
    └── empty_dir
```

При выгрузке директории `series` с вложенной таблицей `episodes` будет создана следующая структура файлов:

```
└── series
    ├── permissions.pb
    └── episodes
        ├── data_00.csv
        ├── permissions.pb
        └── scheme.pb
```

Выгрузка в файловую систему

Кластер

Команда `admin cluster dump` выгружает в клиентскую файловую систему метаданные кластера, в описанном в статье Файловая структура выгрузки формате:

```
ydb [connection options] admin cluster dump [options]
```

, где `[connection options]` — опции соединения с БД

`[options]` – параметры команды:

- `-o <PATH>` или `--output <PATH>`: Путь к директории в клиентской файловой системе, куда должна быть сделана выгрузка.
  Если указанная директория не существует, то она будет создана, однако весь путь до неё должен существовать.

  Если указанная директория существует, то она должна быть пуста.

  Если параметр не указан, то в текущей директории будет создана директория с именем в формате `backup_YYYYDDMMTHHMMSS`, где YYYYDDMM - дата, а HHMMSS - время начала выгрузки, согласно системным часам.

Конфигурация кластера выгружается отдельно с помощью команды `ydb admin cluster config fetch`.

База данных

Команда `admin database dump` выгружает в клиентскую файловую систему данные и метаданные базы данных, в описанном в статье Файловая структура выгрузки формате:

Важно

На текущий момент данная команда не обрабатывает колоночные таблицы. Для выгрузки колоночных таблиц используйте команду `export s3`.

Важно

На текущий момент данная команда не обрабатывает секреты. Секреты необходимо создавать при восстановлении вручную до того, как будут восстановлены использующие их объекты, такие как, например, трансферы данных.

```
ydb [connection options] admin database dump [options]
```

, где `[connection options]` — опции соединения с БД

`[options]` – параметры команды:

- `-o <PATH>` или `--output <PATH>`: Путь к директории в клиентской файловой системе, куда должна быть сделана выгрузка.
  Если указанная директория не существует, то она будет создана, однако весь путь до неё должен существовать.

  Если указанная директория существует, то она должна быть пуста.

  Если параметр не указан, то в текущей директории будет создана директория с именем в формате `backup_YYYYDDMMTHHMMSS`, где YYYYDDMM - дата, а HHMMSS - время начала выгрузки, согласно системным часам.

Конфигурация базы данных выгружается отдельно с помощью команды `ydb admin database config fetch`.

Объекты схемы данных

Команда `tools dump` выгружает в клиентскую файловую систему данные и информацию об объектах схемы данных, в описанном в статье Файловая структура выгрузки формате:

Важно

На текущий момент данная команда не обрабатывает колоночные таблицы. Для выгрузки колоночных таблиц используйте команду `export s3`.

Важно

На текущий момент данная команда не обрабатывает секреты. Секреты необходимо создавать при восстановлении вручную до того, как будут восстановлены использующие их объекты, такие как, например, трансферы данных.

```
ydb [connection options] tools dump [options]
```

, где `[connection options]` — опции соединения с БД

`[options]` – параметры команды:

- `-o <PATH>` или `--output <PATH>`: Путь к директории в клиентской файловой системе, куда должна быть сделана выгрузка.
  Если указанная директория не существует, то она будет создана, однако весь путь до неё должен существовать.

  Если указанная директория существует, то она должна быть пуста.

Если параметр не указан, то в текущей директории будет создана директория с именем в формате `backup_YYYYDDMMTHHMMSS`, где YYYYDDMM - дата, а HHMMSS - время начала выгрузки, согласно системным часам.

- `-p <PATH>` или `--path <PATH>`: Путь к директории в базе данных, объекты внутри которой должны быть выгружены, либо путь к таблице. По умолчанию – корневая директория базы данных. В выгрузку будут включены все поддиректории, имена которых не начинаются с точки, и таблицы внутри них, имена которых не начинаются с точки. Для выгрузки таких таблиц или содержимого таких директорий можно явно указать их имена в данном параметре.
- `--exclude <STRING>`: Шаблон (PCRE) для исключения путей из выгрузки. Данный параметр может быть указан несколько раз, для разных шаблонов.
- `--scheme-only`: Выгружать только информацию об объектах схемы данных, без выгрузки данных.
- `--consistency-level <VAL>`: Уровень консистентности. Возможные варианты:
  - `database` - Полностью консистентная выгрузка, со снятием одного снапшота перед началом выгрузки. Применяется по умолчанию.
  - `table` - Консистентность в пределах каждой выгружаемой таблицы, со снятием отдельных независимых снапшотов для каждой выгружаемой таблицы. Может работать быстрее и оказывать меньшее влияние на обработку текущей нагрузки на базу данных.
- `--avoid-copy`: Не применять создание снапшота для выгрузки. Применяемый по умолчанию для обеспечения консистентности снапшот может быть неприменим для некоторых случаев (например, для таблиц со внешними блобами). Для корректной выгрузки таблиц с серийными типами требуется не выставлять этот параметр. Иначе актуальное значение генератора последовательности не будет скопировано, и новые значения будут выдаваться со стартового значения, что может привести к конфликтам по первичному ключу.
- `--save-partial-result`: Не удалять результат частично выполненной выгрузки. Без включения данной опции результат выгрузки будет удален, если в процессе её выполнения произойдет ошибка.
- `--preserve-pool-kinds`: Если эта опция активна, в дамп будут сохранены имена типов устройств хранения, заданные для групп колонок таблиц (см. параметр `DATA` в статье Группы колонок). Если при восстановлении данных в базе не окажется пулов хранения с указанными именами, восстановление завершится ошибкой. По умолчанию данная опция деактивирована и при восстановлении используется пул хранения, заданный для базы данных при ее создании (см. Создайте базу данных).
- `--ordered`: Строки в выгруженных таблицах будут отсортированы по первичному ключу.

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Выгрузка кластера

С автоматическим созданием директории `backup_...` в текущей директории:

```
ydb -e <endpoint> admin cluster dump
```

В заданную директорию:

```
ydb -e <endpoint> admin cluster dump -o ~/backup_cluster
```

Выгрузка базы данных

С автоматическим созданием директории `backup_...` в текущей директории:

```
ydb -e <endpoint> -d <database> admin database dump
```

В заданную директорию:

```
ydb -e <endpoint> -d <database> admin database dump -o ~/backup_db
```

Выгрузка объектов схемы данных базы данных

С автоматическим созданием директории `backup_...` в текущей директории:

```
ydb --profile quickstart tools dump
```

В заданную директорию:

```
ydb --profile quickstart tools dump -o ~/backup_quickstart
```

Выгрузка структуры таблиц в заданной директории базы данных и её поддиректориях

```
ydb --profile quickstart tools dump -p dir1 --scheme-only
```

Загрузка из файловой системы

Кластер

Команда `admin cluster restore` восстанавливает метаданные кластера из резервной копии в файловой системе, ранее выгруженной туда командой `admin cluster dump`, или подготовленной вручную в соответствии с правилами, описанными в статье Файловая структура выгрузки:

```
ydb [connection options] admin cluster restore -i <PATH> [options]
```

, где `[connection options]` — опции соединения с БД

Перед восстановлением целевой кластер должен быть запущен и проинициализирован.

При восстановлении метаданных кластера создаются базы данных и их администраторы. Дальнейшее восстановление баз данных описано в разделе База данных.

Операция восстановления требует, чтобы для каждой восстанавливаемой базы данных были доступны ее узлы базы данных. Узлы базы данных можно запустить заранее или во время ожидания доступных узлов операцией восстановления. При проблемах с доступными узлами базы данных операцию восстановления можно запустить повторно.

Конфигурация кластера восстанавливается отдельно с помощью следующих действий:

1. Загрузить сохраненную конфигурацию с помощью команды `ydb admin cluster config replace`.
2. Перезапустить узлы кластера.

Обязательные параметры

`-i <PATH>` или `--input <PATH>`: Путь к директории в клиентской файловой системе, откуда будет выполняться загрузка.

Необязательные параметры

`[options]` – необязательные параметры команды:

`--wait-nodes-duration <DURATION>`: Продолжительность ожидания доступных узлов восстанавливаемых баз данных. Пример: `10s`, `5m`, `1h`, `1.5d`, `30`. Продолжительность может выражаться в неделях, днях, часах, минутах, секундах, микросекундах, наносекундах. Если не указать суффикс, то продолжительность является секундами. Продолжительность может быть дробной. Комбинированная продолжительность, как `1h30m`, не поддерживается. Если продолжительность `0`, то ожидание отсутствует.

База данных

Команда `admin database restore` восстанавливает базу данных из резервной копии на файловой системы, ранее выгруженную туда командой `admin database dump`, или подготовленную вручную в соответствии с правилами, описанными в статье Файловая структура выгрузки:

> **Важно**
>
> На текущий момент данная команда не восстанавливает данные колоночных таблиц. Для загрузки колоночных таблиц используйте команду `import s3` с резервной копией, созданной командой `export s3`.

> **Важно**
>
> На текущий момент данная команда не восстанавливает секреты (эти данные не выгружаются в резервную копию). Секреты необходимо создавать при восстановлении вручную до того, как будут восстановлены использующие их объекты, такие как, например, трансферы данных.

```
ydb [connection options] admin database restore -i <PATH> [options]
```

, где `[connection options]` — опции соединения с БД

Операция восстановления требует, чтобы для каждой восстанавливаемой базы данных были доступны ее узлы базы данных. Узлы базы данных можно запустить заранее или во время ожидания доступных узлов операцией восстановления. При проблемах с доступными узлами базы данных операцию восстановления можно запустить повторно.

Восстановление схемных объектов базы данных происходит также, как и описано в разделе Объекты схемы данных.

Конфигурация базы данных восстанавливается отдельно с помощью следующих действий:

1. Загрузить сохранненую конфигурацию с помощью команды `ydb admin database config replace`.
2. Перезапустить узлы базы данных.

Обязательные параметры

`-i <PATH>` или `--input <PATH>`: Путь к директории в клиентской файловой системе, откуда будет выполняться загрузка.

Необязательные параметры

`[options]` – необязательные параметры команды:

`--wait-nodes-duration <DURATION>`: Продолжительность ожидания доступных узлов восстанавливаемой базы данных. Пример: `10s`, `5m`, `1h`, `1.5d`, `30`. Продолжительность может выражаться в неделях, днях, часах, минутах, секундах, микросекундах, наносекундах. Если не указать суффикс, то продолжительность является секундами. Продолжительность может быть дробной. Комбинированная продолжительность, как `1h30m`, не поддерживается. Если продолжительность `0`, то ожидание отсутствует.

Объекты схемы данных

Команда `tools restore` создает в базе данных объекты схемы данных и загружает в них данные из файловой системы, ранее выгруженные туда командой `tools dump`, или подготовленные вручную в соответствии с правилами, описанными в статье Файловая структура:

Важно

На текущий момент данная команда не восстанавливает данные колоночных таблиц. Для загрузки колоночных таблиц используйте команду `import s3` с резервной копией, созданной командой `export s3`.

Важно

На текущий момент данная команда не восстанавливает секреты (эти данные не выгружаются в резервную копию). Секреты необходимо создавать при восстановлении вручную до того, как будут восстановлены использующие их объекты, такие как, например, трансферы данных.

```
ydb [connection options] tools restore -p <PATH> -i <PATH> [options]
```

, где `[connection options]` — опции соединения с БД

Если таблица или директория уже существует в базе данных, изменения в её схему или ACL внесены не будут. Это может привести к тому, что операция загрузки данных окажется невозможной, если не все колонки из загружаемых файлов присутствуют в таблице или если есть несоответствия по типам данных.

Для загрузки данных в таблицу применяется команда YQL `REPLACE`. Если до момента загрузки в таблице существовали записи, то те из них, ключи которых существуют в загружаемых файлах, будут заменены данными из файлов; записи с теми ключами, которых не существует в загружаемых файлах, останутся без изменений.

Обязательные параметры

- `-p <PATH>` или `--path <PATH>`: Путь к директории в базе данных, внутрь которой будет выполняться загрузка. Для загрузки в корневой каталог укажите `.`. Все отсутствующие директории на пути будут созданы.
- `-i <PATH>` или `--input <PATH>`: Путь к директории в клиентской файловой системе, откуда будет выполняться загрузка.

Необязательные параметры

`[options]` – необязательные параметры команды:

- `--restore-data <VAL>`: Флаг загрузки данных, 1 (да) или 0 (нет), по умолчанию 1. Если флаг установлен в 0, то при загрузке будут только созданы объекты в схеме, а данные в них загружены не будут. Если в файловой системе нет данных (выгружена только схема), то изменение флага не имеет значения.
- `--restore-indexes <VAL>`: Флаг загрузки индексов, 1 (да) или 0 (нет), по умолчанию 1. Если флаг установлен в 0, то при загрузке индексы не будут ни зарегистрированы в схеме данных, ни заполнены данными.
- `--restore-acl <VAL>`: Флаг загрузки ACL, 1 (да) или 0 (нет), по умолчанию 1. Если флаг установлен в 0, то при загрузке объекты будут созданы в схеме с пустым ACL, а их владельцем станет пользователь, запустивший загрузку.
- `--replace-sys-acl <VAL>`: Флаг замены ACL для системных объектов, 1 (да) или 0 (нет), по умолчанию 1.
- `--dry-run`: Режим проверки соответствия схемы данных в базе данных и файловой системе без внесения изменений в базу данных, 1 (да) или 0 (нет), по умолчанию 0. При включении данного режима проверяется, что:
  - Все таблицы в файловой системе присутствуют в базе данных
  - Схема данных объектов в файловой системе и базе данных одинаковая
- `--save-partial-result`: Сохранять результат неполной загрузки. Без включения данной опции ошибка в процессе выполнения загрузки приведет к восстановлению состояния базы данных на момент перед началом загрузки.
- `--import-data`: Использовать ImportData – более эффективный способ загрузки данных, чем применяется по умолчанию. Этот способ передает на сервер данные, нужным образом разделённые на партиции, и в более легковесном формате. Однако ImportData вернёт ошибку при попытке импорта таблицы, имеющей индексы или находящейся в процессе их построения, в существующую таблицу. Поэтому перед импортом таблицы с индексами необходимо убедиться в том, что они отсутствуют в схеме (например, используя команду `ydb scheme ls`). По умолчанию ImportData не используется.
- `--replace`: Удалить существующие объекты из базы данных, которые совпадают с объектами в резервной копии перед восстановлением. Объекты, присутствующие в резервной копии, но отсутствующие в базе данных, восстанавливаются как обычно; удаление пропускается. Если указаны одновременно `--replace` и `--verify-existence`, восстановление останавливается с ошибкой при обнаружении первого такого объекта.
- `--verify-existence`: Используется с опцией `--replace` для вывода ошибки, если объект из резервной копии отсутствует в базе данных, вместо того чтобы молча пропускать его удаление.
- `--retries`: Количество повторных попыток для каждого запроса загрузки данных. Значение по умолчанию: `10`.

Параметры ограничения нагрузки

Приведенные ниже параметры позволяют ограничить нагрузку на базу данных, создаваемую в процессе загрузки данных.

Важно

Некоторые из перечисленных ниже параметров имеют действующие значения по умолчанию . Это значит, что если даже ни один из них не задан в команде вызова `tools restore`, нагрузка все равно будет ограничена.

- `--rps <VAL>`: Ограничение количества запросов на загрузку пакетов данных в базу данных в секунду, по умолчанию 30.
- `--bandwidth <VAL>`: Ограничение объема загрузки в секунду, по умолчанию 0 (не установлено). `<VAL>` определяет объем, задается в виде числа с приставкой, например 2MiB. Если определено значение `--bandwidth` ограничение на `--rps` (см. выше) не применяется.

- `--in-flight <VAL>`: Ограничения количество запросов, одновременно находящихся в состоянии исполнения, по умолчанию 10. Для достижения максимального параллелизма следует использовать значение, равное количеству ядер процессора, отведённых под операцию восстановления.
- `--upload-batch-rows <VAL>`: Ограничение количества записей в пакете загружаемых данных, по умолчанию 0 (не ограничено). `<VAL>` определяет количество записей, задается в виде числа с необязательной десятичной приставкой, например 1K.
- `--upload-batch-bytes <VAL>`: Ограничение объема пакета загружаемых данных, по умолчанию 512KB. `<VAL>` определяет объем, задается в виде числа с приставкой, например 1MiB. Максимальное значение — 16MiB.
- `--upload-batch-rus <VAL>`: Применимо только для Serverless баз данных, ограничивает потребление Request Units (RU) на загрузку одного пакета, по умолчанию 30 RU. Размер пакета подбирается под указанное значение. `<VAL>` определяет количество RU, задается в виде числа с необязательной десятичной приставкой, например 100 или 1K.

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Загрузка кластера

Из текущей директории файловой системы:

```
ydb -e <endpoint> admin cluster restore -i .
```

Из заданной директории файловой системы:

```
ydb -e <endpoint> admin cluster restore -i ~/backup_cluster
```

Загрузка базы данных

Из текущей директории файловой системы:

```
ydb -e <endpoint> -d <database> admin database restore -i .
```

Из заданной директории файловой системы:

```
ydb -e <endpoint> -d <database> admin database restore -i ~/backup_db
```

Загрузка схемных объектов в корень базы данных

Из текущей директории файловой системы:

```
ydb -p quickstart tools restore -p . -i .
```

Из заданной директории файловой системы:

```
ydb -p quickstart tools restore -p . -i ~/backup_quickstart
```

Загрузка схемных объектов в заданную директорию в базе данных

Из текущей директории файловой системы:

```
ydb -p quickstart tools restore -p dir1/dir2 -i .
```

Из заданной директории файловой системы:

```
ydb -p quickstart tools restore -p dir1/dir2 -i ~/backup_quickstart
```

Проверка соответствия схемы данных в базе данных и файловой системе:

```
ydb -p quickstart tools restore -p dir1/dir2 -i ~/backup_quickstart --dry-run
```

Пример вызова команды для оптимизации скорости загрузки

```
ydb -p quickstart tools restore -p . -i . --import-data --bandwidth=10GiB --in-flight=16 --upload-batch-bytes
=16MiB
```

Соединение с S3-совместимыми объектными хранилищами

Команды выгрузки и загрузки из S3-совместимых хранилищ `export s3` и `import s3` используют одинаковые параметры для соединения с S3 и аутентификации. О том, как узнать эти параметры для некоторых облачных поставщиков, описано в разделе Получение параметров соединения с S3 ниже.

Соединение

Соединение с S3 требует указания эндпоинта и бакета:

`--s3-endpoint HOST`: Эндпоинт S3. `HOST` - валидное имя хоста, например `storage.yandexcloud.net`

`--bucket STR`: Бакет S3. `STR` - строка, содержащая имя бакета

Аутентификация

Для успешного соединения, кроме случая загрузки из публично доступного бакета, потребуется аутентифицироваться под учетной записью, для которой разрешена запись (для выгрузки) или чтение (для загрузки) из данного бакета.

Для аутентификации в S3 необходимы два параметра:

- Идентификатор ключа доступа (`--access-key`).
- Секретный ключ доступа (`--secret-key`).

YDB CLI определяет значения этих параметров из следующих источников (в порядке убывания приоритета):

1. Командной строки.
2. Переменных окружения.
3. Файла `~/.aws/credentials`.

Параметры командной строки

- `--access-key` — идентификатор ключа доступа.
- `--secret-key` — секретный ключ доступа.
- `--aws-profile` — имя профиля в файле `~/.aws/credentials`. Значение по умолчанию `default`.

Переменные окружения

Если какой-либо параметр аутентификации не указан в командной строке, YDB CLI пробует его получить из следующих переменных окружения:

- `AWS_ACCESS_KEY_ID` — идентификатор ключа доступа.
- `AWS_SECRET_ACCESS_KEY` — секретный ключ доступа.
- `AWS_PROFILE` — имя профиля в файле `~/.aws/credentials`.

Файл аутентификации AWS

Если какой-либо параметр аутентификации не указан в командной строке и его не удалось получить из переменной окружения, YDB CLI пробует его получить из указанного профиля или профиля по умолчанию в файле `~/.aws/credentials`, применяемого для аутентификации AWS CLI. Данный файл может быть создан командой AWS CLI `aws configure`.

Получение параметров соединения с S3

Yandex.Cloud

Ниже описан сценарий получения ключей доступа к Yandex.Cloud Object Storage с применением Yandex.Cloud CLI.

1. Установите и сконфигурируйте Yandex.Cloud CLI.
2. Получите ID вашего каталога (`folder-id`) в облаке следующей командой, его понадобится указывать в командах ниже:

   ```
   yc config list
   ```

   В выводе идентификатор каталога в облаке находится в строке `folder-id:`:

   ```
   folder-id: b2ge70qdcff4bo9q6t19
   ```

3. Создайте сервисный аккаунт, выполнив следующую команду:

   ```
   yc iam service-account create --name s3account
   ```

Здесь и ниже используется имя аккаунта `s3account`, но вы можете использовать любое другое. При создании аккаунта будет выведен его id, он потребуется далее.

Вы также можете использовать уже существующий сервисный аккаунт. Чтобы получить id существующего аккаунта по имени, используйте команду:

```
yc iam service-account get --name <account-name>
```

4. Назначьте сервисному аккаунту роли в соответствии с необходимым уровнем доступа к S3, выполнив команду:

   Чтение (для загрузки в базу данных YDB)

   ```
   yc resource-manager folder add-access-binding <folder-id> \
     --role storage.viewer --subject serviceAccount:<s3-account-id>
   ```

Запись (для выгрузки из базы данных YDB)

```
yc resource-manager folder add-access-binding <folder-id> \
  --role storage.editor --subject serviceAccount:<s3-account-id>
```

, где `<folder-id>` - это идентификатор каталога в облаке, полученный на шаге 2, а `<s3-account-id>` - идентификатор аккаунта, созданного на шаге 3.

Вы можете также ознакомиться с полным перечнем ролей Yandex.Cloud.

5. Получите статические ключи доступа, выполнив следующую команду:

```
yc iam access-key create --service-account-name s3account
```

Успешно исполненная команда выведет информацию об атрибутах access_key и значение secret:

```
access_key:
  id: aje6t3vsbj8lp9r4vk2u
  service_account_id: ajepg0mjt06siuj65usm
  created_at: "2018-11-22T14:37:51Z"
  key_id: 0n8X6WY6S24N7OjXQ0YQ
secret: JyTRFdqw8t1kh2-OJNz4JX5ZTz9Dj1rI9hxtzMP1
```

В данном выводе:

- `access_key.key_id` - это идентификатор ключа доступа (`--access-key`).
- `secret` - это секретный ключ доступа (`--secret-key`).

Выгрузка в S3-совместимое хранилище

Команда `export s3` запускает на стороне сервера процесс выгрузки в S3-совместимое хранилище данных и информации об объектах схемы, в описанном в статье Файловая структура формате:

```
ydb [connection options] export s3 [options]
```

, где `[connection options]` — опции соединения с БД

Важно

Выгрузка доступна только для объектов следующих типов:

- директория;
- строковая таблица;
- колоночная таблица;
- вторичный индекс;
- векторный индекс;
- полнотекстовый индекс;
- топик (только схема);
- представление (view);
- асинхронная репликация;
- трансфер;
- внешний источник данных;
- внешняя таблица.

Примечание

В качестве альтернативы для экспорта одиночных таблиц в S3-совместимое хранилище можно использовать внешние источники данных. Подробнее см. в статье Экспорт данных в объектное хранилище S3.

Параметры командной строки

`[options]` - параметры команды:

Параметры S3

Команда выгрузки в S3 требует указания параметров соединения с S3. Так как выгрузка производится в асинхронном режиме сервером YDB, указанный эндпоинт должен быть доступен для установки соединения со стороны сервера.

`--destination-prefix PREFIX` : Префикс ключа в бакете S3.

Перечень выгружаемых объектов

`--root-path PATH` : Корневая директория для выгружаемых объектов. Если не указана, используется корневая директория базы данных.

`--include PATH` : Объекты схемы данных для включения в экспорт. Директории обходятся рекурсивно. Пути указываются относительно `root-path` . Данный параметр может быть указан несколько раз для включения нескольких объектов. Если не указан, выполняется выгрузка всех несистемных объектов в `root-path` .

`--exclude STRING` : Шаблон (PCRE) для исключения путей из выгрузки. Пути указываются относительно `root-path` . Данный параметр может быть указан несколько раз для разных шаблонов.

Альтернативный способ

Поддерживается альтернативный способ указания перечня объектов:

`--item STRING` : Описание объекта выгрузки. Параметр `--item` может быть указан несколько раз, если необходимо выполнить выгрузку нескольких объектов. `STRING` задается в формате `<свойство>=<значение>,...` , со следующими обязательными свойствами:

- `source` , `src` , или `s` — путь до выгружаемой директории или таблицы, `.` указывает на корневую директорию базы данных. При указании директории выгружаются все несистемные объекты в ней, а также рекурсивно все несистемные поддиректории.
- `destination` , `dst` , или `d` — путь (префикс ключа) в S3 для размещения выгружаемых объектов.

`--exclude STRING` : Шаблон (PCRE) для исключения путей из выгрузки. Данный параметр может быть указан несколько раз для разных шаблонов.

Важно

Экспорты, сделанные с использованием альтернативного синтаксиса, не будут содержать списка объектов в составе резервной копии, поэтому некоторые возможности могут быть для них недоступны (в частности, шифрованные резервные копии), а импорт возможен только с использованием соответствующего альтернативного синтаксиса импорта.

Дополнительные параметры

- `--description STRING` : Текстовое описание операции, сохраняемое в истории операций.
- `--retries NUM` : Количество повторных попыток выгрузки, которые будет предпринимать сервер. Значение по умолчанию: `10` .

- `--compression STRING`: Сжимать выгружаемые данные. При уровне сжатия по умолчанию для алгоритма Zstandard данные могут быть сжаты в 5-10 раз. Сжатие данных использует ресурс CPU и может повлиять на скорость выполнения других операций с БД. Допустимые значения:
  - `zstd` — сжатие алгоритмом Zstandard с уровнем сжатия по умолчанию (`3`);
  - `zstd-N` — сжатие алгоритмом Zstandard, `N` — уровень сжатия (`1` — `22`).
- `--include-index-data BOOL`: Выгружать данные индексных таблиц. По умолчанию выгружаются только метаданные индексов, а сами индексы строятся при импорте — это экономит место и сокращает время выгрузки, но может увеличить время импорта. При включении данной опции данные индексных таблиц выгружаются в полном объёме, что позволяет при импорте загрузить их вместо построения. Значение по умолчанию: `false`.
- `--encryption-algorithm ALGORITHM`: Шифровать выгружаемые данные используя указанный алгоритм. Поддерживаемые значения: `AES-128-GCM`, `AES-256-GCM`, `ChaCha20-Poly1305`.
- `--encryption-key-file PATH`: Путь к файлу, содержащему ключ шифрования (только для зашифрованных выгрузок). Данный файл является бинарным и должен содержать точное количество байт, соответствующее длине ключа в выбранном алгоритме шифрования (16 байт для `AES-128-GCM`, 32 байта для `AES-256-GCM` и `ChaCha20-Poly1305`). Ключ также может быть передан через переменную окружения `YDB_ENCRYPTION_KEY`, в шестнадцатеричном строковом представлении.
- `--format STRING`: Формат вывода результата. Допустимые значения:
  - `pretty` — человекочитаемый формат (по умолчанию);
  - `proto-json-base64` — Protocol Buffers в формате JSON, бинарные строки закодированы в Base64.

Выполнение выгрузки

Ход серверной операции выгрузки

1. Создаётся серверная асинхронная операция экспорта.
2. В корневой директории базы данных создаётся служебная директория `export-{id}`, где `{id}` — числовой идентификатор операции.
3. В эту директорию создаётся согласованная копия таблиц с помощью механизма `CopyTables`.
4. Данные каждой таблицы записываются из копии в хранилище в формате файловой структуры выгрузки с параллельной записью из разных узлов кластера.
5. После успешного завершения служебная директория `export-{id}` и копии таблиц удаляются.

Результат запуска

При успешном исполнении команда `export s3` выводит сводную информацию о поставленной в очередь операции выгрузки в S3, в заданном опцией `--format` формате. Фактическая выгрузка производится сервером асинхронно. В сводной информации выводится ID операции, который может быть использован в дальнейшем для проверки статуса и действий с операцией:

- В режиме вывода `pretty` (по умолчанию) идентификатор операции показывается в выделенном псевдографикой поле id:

```
┌──────────────────────────────────────────────┬───────┬──────...
| id                                           | ready | stat...
├──────────────────────────────────────────────┼───────┼──────...
| ydb://export/6?id=281474976788395&kind=s3 | true  | SUCC...
├---------------------------------------------┴-------┴------...
| StorageClass: NOT_SET
| Items:
...
```

- В режиме вывода `proto-json-base64` идентификатор находится в атрибуте "id":

```
{"id":"ydb://export/6?id=281474976788395&kind=s3","ready":true, ... }
```

Статус выгрузки

Выгрузка данных выполняется в фоновом режиме. Получить информацию о статусе и прогрессе выгрузки можно вызовом команды `operation get`, параметром которой должен быть передан **заключенный в кавычки** идентификатор операции, например:

```
ydb -p quickstart operation get "ydb://export/6?id=281474976788395&kind=s3"
```

Формат вывода `operation get` также устанавливается опцией `--format`.

Несмотря на то, что идентификатор операции имеет формат URL, не гарантируется, что он будет сохранен в дальнейшем. Его нужно интерпретировать только как строку.

Завершение выгрузки отслеживается по изменению атрибута "progress":

- В режиме вывода `pretty` (по умолчанию) успешно завершенная операция отражается значением "Done" в выделенном псевдографикой поле `progress`:

```
┌──────── ... ──┬─────────┬──────────┬────────────┬...
| id            | ready | status   | progress | ...
├──────────... ──┼─────────┼──────────┼────────────┼...
| ydb://...     | true  | SUCCESS | Done       | ...
├-----  ... --┴-------┴---------┴----------┴ ...
...
```

- В режиме вывода `proto-json-base64` завершенная операция отражается значением `PROGRESS_DONE` атрибута `progress`:

```
{"id":"ydb://...", ...,"progress":"PROGRESS_DONE",... }
```

Завершение операции выгрузки

После выполнения выгрузки воспользуйтесь командой `operation forget` для того, чтобы выгрузка считалась завершённой (была удалена из перечня операций):

```
ydb -p quickstart operation forget "ydb://export/6?id=281474976788395&kind=s3"
```

Список операций выгрузки

Для получения списка операций выгрузки воспользуйтесь командой `operation list export/s3`:

```
ydb -p quickstart operation list export/s3
```

Формат вывода `operation list` также устанавливается опцией `--format`.

Примеры

Для первоначальной синхронизации асинхронной репликации используйте пошаговое руководство по переносу с помощью резервного копирования и восстановления. В этом сценарии потоки изменений должны быть готовы до запуска экспорта.

Примечание

В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Выгрузка базы данных

Выгрузка всех несистемных объектов базы данных в директорию `export1` в бакете `mybucket` с использованием параметров аутентификации S3 из переменных окружения или файла `~/.aws/credentials`:

```
ydb -p quickstart export s3 \
  --s3-endpoint storage.yandexcloud.net --bucket mybucket \
  --destination-prefix export1
```

Выгрузка нескольких директорий

Выгрузка объектов из директорий `dir1` и `dir2` базы данных, в директорию `export1` в бакете `mybucket`, с использованием явно заданных параметров аутентификации в S3:

```
ydb -p quickstart export s3 \
  --s3-endpoint storage.yandexcloud.net --bucket mybucket \
  --access-key <access-key> --secret-key <secret-key> \
  --destination-prefix export1 --include dir1 --include dir2
```

Либо с использованием альтернативного способа:

```
ydb -p quickstart export s3 \
  --s3-endpoint storage.yandexcloud.net --bucket mybucket \
  --access-key <access-key> --secret-key <secret-key> \
  --item src=dir1,dst=export1/dir1 --item src=dir2,dst=export1/dir2
```

Выгрузка с шифрованием

Выгрузка всей базы данных с шифрованием:

- С использованием алгоритма шифрования `AES-128-GCM`
- С генерацией случайного ключа утилитой `openssl` в файл `~/my_secret_key`
- С чтением сгенерированного ключа из файла `~/my_secret_key`
- В префикс пути `export1` в бакете S3 `mybucket`
- С использованием параметров аутентификации S3 из переменных окружения или файла `~/.aws/credentials`

```
openssl rand -out ~/my_secret_key 16
ydb -p quickstart export s3 \
  --s3-endpoint storage.yandexcloud.net --bucket mybucket --destination-prefix export1 \
  --encryption-algorithm AES-128-GCM --encryption-key-file ~/my_secret_key
```

Выгрузка директории `dir1` базы данных с шифрованием:

- С использованием алгоритма шифрования `AES-256-GCM`
- С генерацией случайного ключа утилитой `openssl` в переменную окружения `YDB_ENCRYPTION_KEY`
- С чтением сгенерированного ключа из переменной окружения `YDB_ENCRYPTION_KEY`
- В префикс пути `export1` в бакете S3 `mybucket`
- С использованием параметров аутентификации S3 из переменных окружения или файла `~/.aws/credentials`

```
export YDB_ENCRYPTION_KEY=$(openssl rand -hex 32)
ydb -p quickstart export s3 \
  --root-path dir1 \
```

```
  --s3-endpoint storage.yandexcloud.net --bucket mybucket --destination-prefix export1 \
  --encryption-algorithm AES-256-GCM
```

Получение идентификаторов операций

Для получения перечня идентификаторов операций выгрузки в удобном для обработки в скриптах bash формате вы можете применить утилиту jq:

```
ydb -p quickstart operation list export/s3 --format proto-json-base64 | jq -r ".operations[].id"
```

Вы получите вывод, где в каждой новой строке находится идентификатор операции, например:

```
ydb://export/6?id=281474976789577&kind=s3
ydb://export/6?id=281474976789526&kind=s3
ydb://export/6?id=281474976788779&kind=s3
```

По этим идентификаторам может быть, например, запущен цикл для завершения всех текущих операций:

```
ydb -p quickstart operation list export/s3 --format proto-json-base64 | jq -r ".operations[].id" | while read
line; do ydb -p quickstart operation forget $line;done
```

Загрузка из S3-совместимого хранилища

Команда `import s3` запускает на стороне сервера процесс загрузки из S3-совместимого хранилища данных и информации об объектах схемы, в описанном в статье Файловая структура формате:

```
ydb [connection options] import s3 [options]
```

> **Примечание**
>
> Импорт таблиц из S3-совместимого хранилища данных в других форматах возможен с использованием внешних таблиц, подробнее см. в статье Импорт данных.

, где `[connection options]` — опции соединения с БД

В отличие от команды `tools restore`, команда `import s3` всегда создает объекты целиком, поэтому для её успешного выполнения ни один из загружаемых объектов (ни директорий, ни таблиц) не должен существовать.

При необходимости догрузки данных в существующие таблицы из S3 вы можете скопировать содержимое S3 в файловую систему (например, с помощью S3cmd) и воспользоваться командой `tools restore`.

> **Важно**
>
> Команда `tools restore` загружает данные только в строковые таблицы. Для догрузки в существующие колоночные таблицы используйте внешние источники данных. Подробнее см. в статье Импорт данных.

Параметры командной строки

`[options]` - параметры команды:

Параметры S3

Команда загрузки из S3 требует указания параметров соединения с S3. Так как загрузка производится в асинхронном режиме сервером YDB, указанный эндпоинт должен быть доступен для установки соединения со стороны сервера.

`--source-prefix PREFIX`: Префикс загрузки в бакете S3.

Загружаемые объекты схемы базы данных

`--destination-path PATH`: Целевая директория для загружаемых объектов; значением по умолчанию является корень базы данных.

`--include PATH`: Объекты схемы данных для включения в импорт. Директории обходятся рекурсивно. Для включения нескольких объектов допускается указание параметра несколько раз. Если не указан, выполняется загрузка всех объектов выгрузки.

`--exclude STRING`: Шаблон (PCRE) для исключения путей из импорта. Данный параметр может быть указан несколько раз для разных шаблонов.

Альтернативный способ

В целях обратной совместимости поддерживается альтернативный способ указания перечня объектов:

`--item STRING`: Описание объекта загрузки. Параметр `--item` может быть указан несколько раз, если необходимо выполнить загрузку нескольких объектов. Если параметры `--item` или `--include` не указаны, будут загружены все объекты, присутствующие в указанной выгрузке. `STRING` задаётся в формате `<свойство>=<значение>,...` со следующими обязательными свойствами:

- `source`, `src` или `s` — префикс ключа в S3 с загружаемой директорией или таблицей.
- `destination`, `dst`, или `d` — путь в базе данных для размещения загружаемой директории или таблицы. Конечный элемент пути не должен существовать. Все директории на пути будут созданы, если не существуют.

Некоторые возможности могут быть недоступны при использовании альтернативного синтаксиса (в частности, шифрованные резервные копии или перечисление объектов выгрузки).

Дополнительные параметры

- `--description STRING`: Текстовое описание операции, сохраняемое в истории операций.
- `--retries NUM`: Количество повторных попыток загрузки, которые будет предпринимать сервер. Значение по умолчанию: `10`.
- `--skip-checksum-validation`: Пропустить этап валидации контрольных сумм загружаемых объектов.
- `--index-population-mode STRING`: Режим наполнения индексов при импорте. Допустимые значения:
  - `build` — построить индексы с нуля (значение по умолчанию);
  - `import` — загрузить данные индексных таблиц (если выгрузка была выполнена с опцией `--include-index-data`);
  - `auto` — попытаться загрузить данные индексных таблиц, при отсутствии данных построить индексы.
- `--encryption-key-file PATH`: Путь к файлу, содержащему ключ шифрования (только для зашифрованных выгрузок). Данный файл является бинарным и должен содержать точное количество байт, соответствующее длине ключа в выбранном алгоритме шифрования (16 байт для `AES-128-GCM`, 32 байта для `AES-256-GCM` и `ChaCha20-Poly1305`). Ключ также может быть передан через переменную окружения `YDB_ENCRYPTION_KEY`, в шестнадцатеричном строковом представлении.
- `--format STRING`: Формат вывода результата. Допустимые значения:
  - `pretty` — человекочитаемый формат (по умолчанию);
  - `proto-json-base64` — Protocol Buffers в формате JSON, бинарные строки закодированы в Base64.

- `--list`: Перечислить объекты в существующей выгрузке.

> **Примечание**
>
> Использование режима `--index-population-mode=import` позволяет максимально задействовать ресурсы и быстро восстановить индексы, либо наоборот — ограничить ресурсы и скопировать индексные таблицы с минимальным влиянием на пользовательскую нагрузку. Для ограничения ресурсов импорта используется очередь `queue_restore` брокера ресурсов.

Выполнение загрузки

Ход серверной операции загрузки

1. Создаётся серверная асинхронная операция импорта.
2. Сервер считывает метаданные объектов из файлов (`scheme.pb`, `metadata.json` и т.д.) из хранилища.
3. Для каждого объекта создаётся новая таблица в базе данных. Целевые пути **не должны существовать** — импорт не может перезаписать существующие таблицы.
4. Данные загружаются из файлов параллельно на всех узлах кластера.

Результат запуска

При успешном исполнении команда `import s3` выводит сводную информацию о поставленной в очередь операции загрузки из S3, в заданном опцией `--format` формате. Фактическая загрузка производится сервером асинхронно. В сводной информации выводится ID операции, который может быть использован в дальнейшем для проверки статуса и действий с операцией:

- В режиме вывода `pretty` (по умолчанию) идентификатор операции показывается в выделенном псевдографикой поле id:

```
┌──────────────────────────────────────────┬───────┬─────...
| id                                       | ready | stat...
├──────────────────────────────────────────┼───────┼─────...
| ydb://import/8?id=281474976788395&kind=s3 | true  | SUCC...
├------------------------------------------┴-------┴-----...
| Items:
...
```

- В режиме вывода `proto-json-base64` идентификатор находится в атрибуте "id":

```
{"id":"ydb://import/8?id=281474976788395&kind=s3","ready":true, ... }
```

Статус загрузки

Загрузка данных выполняется в фоновом режиме. Получить информацию о статусе и прогрессе загрузки можно вызовом команды `operation get`, параметром которой должен быть передан **заключенный в кавычки** идентификатор операции, например:

```
ydb -p quickstart operation get "ydb://import/8?id=281474976788395&kind=s3"
```

Формат вывода `operation get` также устанавливается опцией `--format`.

Несмотря на то, что идентификатор операции имеет формат URL, не гарантируется, что он будет сохранен в дальнейшем. Его нужно интерпретировать только как строку.

Завершение загрузки отслеживается по изменению атрибута "progress":

- В режиме вывода `pretty` (по умолчанию) успешно завершенная операция отражается значением "Done" в выделенном псевдографикой поле `progress`:

  ```
  ┌──── ... ──┬───────┬─────────┬────────────┬...
  | id        | ready | status  | progress | ...
  ├────...  ──┼───────┼─────────┼────────────┼...
  | ydb://...   | true  | SUCCESS | Done     | ...
  ├---- ... --┴-------┴---------┴----------┴ ...
  ...
  ```

- В режиме вывода `proto-json-base64` завершенная операция отражается значением `PROGRESS_DONE` атрибута `progress`:

  ```
  {"id":"ydb://...", ...,"progress":"PROGRESS_DONE",... }
  ```

Завершение операции загрузки

После выполнения загрузки воспользуйтесь командой `operation forget` для того, чтобы загрузка была удалена из перечня операций:

```
ydb -p quickstart operation forget "ydb://import/8?id=281474976788395&kind=s3"
```

Список операций загрузки

Для получения списка операций загрузки воспользуйтесь командой `operation list import/s3`:

```
ydb -p quickstart operation list import/s3
```

Формат вывода `operation list` также устанавливается опцией `--format`.

Примеры

Для восстановления первоначальных данных перед запуском асинхронной репликации используйте пошаговое руководство по переносу с помощью резервного копирования и восстановления. Репликация запускается после успешного импорта всех целевых таблиц.

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Загрузка в корень базы данных

Загрузка в корень базы данных содержимого директории `export1` в бакете `mybucket` с использованием параметров аутентификации S3 из переменных окружения или файла `~/.aws/credentials`:

```
ydb -p quickstart import s3 \
  --s3-endpoint storage.yandexcloud.net --bucket mybucket \
  --source-prefix export1
```

Загрузка нескольких директорий

Загрузка объектов из директорий `dir1` и `dir2` выгрузки, которая находится в директории `export1` в бакете `mybucket`, в одноименные директории базы данных с использованием явно заданных параметров аутентификации в S3:

```
ydb -p quickstart import s3 \
  --s3-endpoint storage.yandexcloud.net --bucket mybucket \
  --access-key <access-key> --secret-key <secret-key> \
  --source-prefix export1
  --include dir1 --include dir2
```

Перечисление объектов в существующей зашифрованной выгрузке

Перечисление путей всех объектов в существующей зашифрованной выгрузке, которая находится в директории `export1` в бакете `mybucket`, с использованием секретного ключа из файла `~/my_secret_key`.

```
ydb -p quickstart import s3 \
  --s3-endpoint storage.yandexcloud.net --bucket mybucket \
  --access-key <access-key> --secret-key <secret-key> \
  --source-prefix export1
  --encryption-key-file ~/my_secret_key
  --list
```

Загрузка зашифрованной выгрузки

Загрузка одной таблицы, которая была выгружена по пути `dir/my_table`, в путь `dir1/dir/my_table` из зашифрованной выгрузки, расположенной по префиксу `export1` в бакете `mybucket`, с использованием секретного ключа из файла `~/my_secret_key`.

```
ydb -p quickstart import s3 \
  --s3-endpoint storage.yandexcloud.net --bucket mybucket \
  --access-key <access-key> --secret-key <secret-key> \
  --source-prefix export1 --destination-path dir1 \
  --include dir/my_table \
  --encryption-key-file ~/my_secret_key
```

Получение идентификаторов операций

Для получения перечня идентификаторов операций загрузки в удобном для обработки в скриптах bash формате вы можете применить утилиту jq:

```
ydb -p quickstart operation list import/s3 --format proto-json-base64 | jq -r ".operations[].id"
```

Вы получите вывод, где в каждой новой строке находится идентификатор операции, например:

```
ydb://import/8?id=281474976789577&kind=s3
ydb://import/8?id=281474976789526&kind=s3
ydb://import/8?id=281474976788779&kind=s3
```

По этим идентификаторам может быть, например, запущен цикл для завершения всех текущих операций:

```
ydb -p quickstart operation list import/s3 --format proto-json-base64 | jq -r ".operations[].id" | while read
line; do ydb -p quickstart operation forget $line;done
```

Выгрузка в NFS

Команда `export nfs` запускает на стороне сервера процесс выгрузки в сетевую файловую систему (Network File System, NFS) хостов кластера YDB данных и информации об объектах схемы, в описанном в статье Файловая структура формате:

```
ydb [connection options] export nfs [options]
```

, где `[connection options]` — опции соединения с БД

Важно

Выгрузка доступна только для объектов следующих типов:

- директория;
- строковая таблица;
- вторичный индекс;
- векторный индекс;
- полнотекстовый индекс;
- топик (только схема);
- представление (view);
- асинхронная репликация;
- трансфер;
- внешний источник данных;
- внешняя таблица.

Важно

На текущий момент данная команда не обрабатывает колоночные таблицы. Для выгрузки колоночных таблиц используйте команду `export s3`.

Параметры командной строки

`[options]` - параметры команды:

Параметры NFS

Команда выгрузки в NFS требует указания монтированной директории (или поддиректории) общей для всех объектов, участвующих в выгрузке. Так как выгрузка производится в асинхронном режиме на всех хостах YDB, указанная директория должна быть на каждом хосте YDB и смонтирована в NFS.

`--fs-path PATH`: путь до монтированной директории (или поддиректории).

Перечень выгружаемых объектов

`--root-path PATH`: Корневая директория для выгружаемых объектов. Если не указана, используется корневая директория базы данных.

`--include PATH`: Объекты схемы данных для включения в экспорт. Директории обходятся рекурсивно. Пути указываются относительно `root-path`. Данный параметр может быть указан несколько раз для включения нескольких объектов. Если не указан, выполняется выгрузка всех несистемных объектов в `root-path`.

`--exclude STRING`: Шаблон (PCRE) для исключения путей из выгрузки. Пути указываются относительно `root-path`. Данный параметр может быть указан несколько раз для разных шаблонов.

Альтернативный способ

Поддерживается альтернативный способ указания перечня объектов:

`--item STRING`: Описание объекта выгрузки. Параметр `--item` может быть указан несколько раз, если необходимо выполнить выгрузку нескольких объектов. `STRING` задается в формате `<свойство>=<значение>,...`, со следующими обязательными свойствами:

- `source`, `src`, или `s` — путь до выгружаемой директории или таблицы, `.` указывает на корневую директорию базы данных. При указании директории выгружаются все несистемные объекты в ней, а также рекурсивно все несистемные поддиректории.
- `destination`, `dst`, или `d` — путь в NFS (относительно `--fs-path`).

`--exclude STRING`: Шаблон (PCRE) для исключения путей из выгрузки. Данный параметр может быть указан несколько раз для разных шаблонов.

Важно

Экспорты, сделанные с использованием альтернативного синтаксиса, не будут содержать списка объектов в составе резервной копии, поэтому некоторые возможности могут быть для них недоступны (в частности, шифрованные резервные копии), а импорт возможен только с использованием соответствующего альтернативного синтаксиса импорта.

Дополнительные параметры

- `--description STRING`: Текстовое описание операции, сохраняемое в истории операций.
- `--retries NUM`: Количество повторных попыток выгрузки, которые будет предпринимать сервер. Значение по умолчанию: `10`.

- `--compression STRING`: Сжимать выгружаемые данные. При уровне сжатия по умолчанию для алгоритма Zstandard данные могут быть сжаты в 5-10 раз. Сжатие данных использует ресурс CPU и может повлиять на скорость выполнения других операций с БД. Допустимые значения:
  - `zstd` — сжатие алгоритмом Zstandard с уровнем сжатия по умолчанию (`3`);
  - `zstd-N` — сжатие алгоритмом Zstandard, `N` — уровень сжатия (`1` — `22`).
- `--include-index-data BOOL`: Выгружать данные индексных таблиц. По умолчанию выгружаются только метаданные индексов, а сами индексы строятся при импорте — это экономит место и сокращает время выгрузки, но может увеличить время импорта. При включении данной опции данные индексных таблиц выгружаются в полном объёме, что позволяет при импорте загрузить их вместо построения. Значение по умолчанию: `false`.
- `--encryption-algorithm ALGORITHM`: Шифровать выгружаемые данные используя указанный алгоритм. Поддерживаемые значения: `AES-128-GCM`, `AES-256-GCM`, `ChaCha20-Poly1305`.
- `--encryption-key-file PATH`: Путь к файлу, содержащему ключ шифрования (только для зашифрованных выгрузок). Данный файл является бинарным и должен содержать точное количество байт, соответствующее длине ключа в выбранном алгоритме шифрования (16 байт для `AES-128-GCM`, 32 байта для `AES-256-GCM` и `ChaCha20-Poly1305`). Ключ также может быть передан через переменную окружения `YDB_ENCRYPTION_KEY`, в шестнадцатеричном строковом представлении.
- `--format STRING`: Формат вывода результата. Допустимые значения:
  - `pretty` — человекочитаемый формат (по умолчанию);
  - `proto-json-base64` — Protocol Buffers в формате JSON, бинарные строки закодированы в Base64.

Выполнение выгрузки

Ход серверной операции выгрузки

1. Создаётся серверная асинхронная операция экспорта.
2. В корневой директории базы данных создаётся служебная директория `export-{id}`, где `{id}` — числовой идентификатор операции.
3. В эту директорию создаётся согласованная копия таблиц с помощью механизма `CopyTables`.
4. Данные каждой таблицы записываются из копии в хранилище в формате файловой структуры выгрузки с параллельной записью из разных узлов кластера.
5. После успешного завершения служебная директория `export-{id}` и копии таблиц удаляются.

Результат запуска

При успешном исполнении команда `export nfs` выводит сводную информацию о поставленной в очередь операции выгрузки в NFS, в заданном опцией `--format` формате. Фактическая выгрузка производится сервером асинхронно. В сводной информации выводится ID операции, который может быть использован в дальнейшем для проверки статуса и действий с операцией:

- В режиме вывода `pretty` (по умолчанию) идентификатор операции показывается в выделенном псевдографикой поле id:

```
┌──────────────────────────────────────────────┬───────┬──────...
| id                                           | ready | stat...
├──────────────────────────────────────────────┼───────┼──────...
| ydb://export/6?id=281474976788395&kind=fs | true  | SUCC...
├---------------------------------------------┴-------┴------...
| Include index data: false
| Items:
...
```

- В режиме вывода `proto-json-base64` идентификатор находится в атрибуте "id":

```
{"id":"ydb://export/6?id=281474976788395&kind=fs","ready":true, ... }
```

Статус выгрузки

Выгрузка данных выполняется в фоновом режиме. Получить информацию о статусе и прогрессе выгрузки можно вызовом команды `operation get`, параметром которой должен быть передан **заключенный в кавычки** идентификатор операции, например:

```
ydb -p quickstart operation get "ydb://export/6?id=281474976788395&kind=fs"
```

Формат вывода `operation get` также устанавливается опцией `--format`.

Несмотря на то, что идентификатор операции имеет формат URL, не гарантируется, что он будет сохранен в дальнейшем. Его нужно интерпретировать только как строку.

Завершение выгрузки отслеживается по изменению атрибута "progress":

- В режиме вывода `pretty` (по умолчанию) успешно завершенная операция отражается значением "Done" в выделенном псевдографикой поле `progress`:

```
┌──────  ...  ──┬─────────┬──────────┬────────────┬...
| id            | ready | status   | progress | ...
├──────...  ──┼─────────┼──────────┼────────────┼...
| ydb://...     | true  | SUCCESS | Done       | ...
├-----  ...  --┴-------┴---------┴----------┴ ...
...
```

- В режиме вывода `proto-json-base64` завершенная операция отражается значением `PROGRESS_DONE` атрибута `progress`:

```
{"id":"ydb://...", ...,"progress":"PROGRESS_DONE",... }
```

Завершение операции выгрузки

После выполнения выгрузки воспользуйтесь командой `operation forget` для того, чтобы выгрузка считалась завершённой (была удалена из перечня операций):

```
ydb -p quickstart operation forget "ydb://export/6?id=281474976788395&kind=fs"
```

Список операций выгрузки

Для получения списка операций выгрузки воспользуйтесь командой `operation list export/nfs`:

```
ydb -p quickstart operation list export/nfs
```

Формат вывода `operation list` также устанавливается опцией `--format`.

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Выгрузка базы данных

Выгрузка всех несистемных объектов базы данных в директорию `/mnt/nfs/backups/export1` на файловой системе:

```
ydb -p quickstart export nfs \
  --fs-path /mnt/nfs/backups/export1
```

Выгрузка нескольких директорий

Выгрузка объектов из директорий `dir1` и `dir2` базы данных в директорию `/mnt/nfs/backups/export1` на файловой системе:

```
ydb -p quickstart export nfs \
  --fs-path /mnt/nfs/backups/export1 \
  --include dir1 --include dir2
```

Либо с использованием альтернативного способа:

```
ydb -p quickstart export nfs \
  --fs-path /mnt/nfs/backups \
  --item src=dir1,dst=export1/dir1 --item src=dir2,dst=export1/dir2
```

Выгрузка с шифрованием

Выгрузка всей базы данных с шифрованием:

- С использованием алгоритма шифрования `AES-128-GCM`
- С генерацией случайного ключа утилитой `openssl` в файл `~/my_secret_key`
- С чтением сгенерированного ключа из файла `~/my_secret_key`
- В директорию `/mnt/nfs/backups/export1` на файловой системе

```
openssl rand -out ~/my_secret_key 16
ydb -p quickstart export nfs \
  --fs-path /mnt/nfs/backups/export1 \
  --encryption-algorithm AES-128-GCM --encryption-key-file ~/my_secret_key
```

Выгрузка директории `dir1` базы данных с шифрованием:

- С использованием алгоритма шифрования `AES-256-GCM`
- С генерацией случайного ключа утилитой `openssl` в переменную окружения `YDB_ENCRYPTION_KEY`
- С чтением сгенерированного ключа из переменной окружения `YDB_ENCRYPTION_KEY`
- В директорию `/mnt/nfs/backups/export1` на файловой системе

```
export YDB_ENCRYPTION_KEY=$(openssl rand -hex 32)
ydb -p quickstart export nfs \
  --root-path dir1 \
  --fs-path /mnt/nfs/backups/export1 \
  --encryption-algorithm AES-256-GCM
```

Получение идентификаторов операций

Для получения перечня идентификаторов операций выгрузки в удобном для обработки в скриптах bash формате вы можете применить утилиту jq:

```
ydb -p quickstart operation list export/nfs --format proto-json-base64 | jq -r ".operations[].id"
```

Вы получите вывод, где в каждой новой строке находится идентификатор операции, например:

```
ydb://export/6?id=281474976789577&kind=fs
ydb://export/6?id=281474976789526&kind=fs
ydb://export/6?id=281474976788779&kind=fs
```

По этим идентификаторам может быть, например, запущен цикл для завершения всех текущих операций:

```
ydb -p quickstart operation list export/nfs --format proto-json-base64 | jq -r ".operations[].id" | while rea
d line; do ydb -p quickstart operation forget $line;done
```

Загрузка из NFS

Команда `import nfs` запускает на стороне сервера процесс загрузки из сетевой файловой системы (Network File System, NFS) хостов кластера YDB данных и информации об объектах схемы, в описанном в статье Файловая структура формате:

```
ydb [connection options] import nfs [options]
```

, где `[connection options]` — опции соединения с БД

> **Важно**
>
> На текущий момент данная команда не восстанавливает данные колоночных таблиц. Для загрузки колоночных таблиц используйте команду `import s3` с резервной копией, созданной командой `export s3`.

В отличие от команды `tools restore`, команда `import nfs` всегда создает объекты целиком, поэтому для её успешного выполнения ни один из загружаемых объектов (ни директорий, ни таблиц) не должен существовать.

При необходимости догрузки данных в существующие таблицы воспользуйтесь командой `tools restore` непосредственно на смонтированной NFS-директории.

Параметры командной строки

`[options]` - параметры команды:

Параметры NFS

Команда загрузки из NFS требует указания монтированной директории (или поддиректории) общей для всех объектов, участвующих в загрузке. Так как загрузка производится в асинхронном режиме на всех хостах YDB, указанная директория должна быть на каждом хосте YDB и смонтирована в NFS.

`--fs-path PATH`: путь до монтированной директории (или поддиректории).

Загружаемые объекты схемы базы данных

`--destination-path PATH`: Целевая директория для загружаемых объектов; значением по умолчанию является корень базы данных.

`--include PATH`: Объекты схемы данных для включения в импорт. Директории обходятся рекурсивно. Для включения нескольких объектов допускается указание параметра несколько раз. Если не указан, выполняется загрузка всех объектов выгрузки.

`--exclude STRING`: Шаблон (PCRE) для исключения путей из импорта. Данный параметр может быть указан несколько раз для разных шаблонов.

Альтернативный способ

В целях обратной совместимости поддерживается альтернативный способ указания перечня объектов:

`--item STRING`: Описание объекта загрузки. Параметр `--item` может быть указан несколько раз, если необходимо выполнить загрузку нескольких объектов. Если параметры `--item` или `--include` не указаны, будут загружены все объекты, присутствующие в указанной выгрузке. `STRING` задаётся в формате `<свойство>=<значение>,...` со следующими обязательными свойствами:

- `source`, `src` или `s` — путь в NFS (относительно `fs-path`) с загружаемой директорией или таблицей.
- `destination`, `dst`, или `d` — путь в базе данных для размещения загружаемой директории или таблицы. Конечный элемент пути не должен существовать. Все директории на пути будут созданы, если не существуют.

Некоторые возможности могут быть недоступны при использовании альтернативного синтаксиса (в частности, шифрованные резервные копии или перечисление объектов выгрузки).

Дополнительные параметры

- `--description STRING`: Текстовое описание операции, сохраняемое в истории операций.
- `--retries NUM`: Количество повторных попыток загрузки, которые будет предпринимать сервер. Значение по умолчанию: `10`.
- `--skip-checksum-validation`: Пропустить этап валидации контрольных сумм загружаемых объектов.
- `--index-population-mode STRING`: Режим наполнения индексов при импорте. Допустимые значения:
  - `build` — построить индексы с нуля (значение по умолчанию);
  - `import` — загрузить данные индексных таблиц (если выгрузка была выполнена с опцией `--include-index-data`);
  - `auto` — попытаться загрузить данные индексных таблиц, при отсутствии данных построить индексы.
- `--encryption-key-file PATH`: Путь к файлу, содержащему ключ шифрования (только для зашифрованных выгрузок). Данный файл является бинарным и должен содержать точное количество байт, соответствующее длине ключа в выбранном алгоритме шифрования (16 байт для `AES-128-GCM`, 32 байта для `AES-256-GCM` и `ChaCha20-Poly1305`). Ключ также может быть передан через переменную окружения `YDB_ENCRYPTION_KEY`, в шестнадцатеричном строковом представлении.
- `--format STRING`: Формат вывода результата. Допустимые значения:
  - `pretty` — человекочитаемый формат (по умолчанию);
  - `proto-json-base64` — Protocol Buffers в формате JSON, бинарные строки закодированы в Base64.

Примечание

Использование режима `--index-population-mode=import` позволяет максимально задействовать ресурсы и быстро восстановить индексы, либо наоборот — ограничить ресурсы и скопировать индексные таблицы с минимальным влиянием на пользовательскую нагрузку. Для ограничения ресурсов импорта используется очередь `queue_restore` брокера ресурсов.

Выполнение загрузки

Ход серверной операции загрузки

1. Создаётся серверная асинхронная операция импорта.
2. Сервер считывает метаданные объектов из файлов (`scheme.pb` , `metadata.json` и т.д.) из хранилища.
3. Для каждого объекта создаётся новая таблица в базе данных. Целевые пути **не должны существовать** — импорт не может перезаписать существующие таблицы.
4. Данные загружаются из файлов параллельно на всех узлах кластера.

Результат запуска

При успешном исполнении команда `import nfs` выводит сводную информацию о поставленной в очередь операции загрузки из NFS, в заданном опцией `--format` формате. Фактическая загрузка производится сервером асинхронно. В сводной информации выводится ID операции, который может быть использован в дальнейшем для проверки статуса и действий с операцией:

- В режиме вывода `pretty` (по умолчанию) идентификатор операции показывается в выделенном псевдографикой поле id:

```
┌──────────────────────────────────────────┬───────┬─────...
| id                                       | ready | stat...
├──────────────────────────────────────────┼───────┼─────...
| ydb://import/8?id=281474976788395&kind=fs | true  | SUCC...
├------------------------------------------┴-------┴-----...
| Items:
...
```

- В режиме вывода `proto-json-base64` идентификатор находится в атрибуте "id":

```
{"id":"ydb://import/8?id=281474976788395&kind=fs","ready":true, ... }
```

Статус загрузки

Загрузка данных выполняется в фоновом режиме. Получить информацию о статусе и прогрессе загрузки можно вызовом команды `operation get` , параметром которой должен быть передан **заключенный в кавычки** идентификатор операции, например:

```
ydb -p quickstart operation get "ydb://import/8?id=281474976788395&kind=fs"
```

Формат вывода `operation get` также устанавливается опцией `--format` .

Несмотря на то, что идентификатор операции имеет формат URL, не гарантируется, что он будет сохранен в дальнейшем. Его нужно интерпретировать только как строку.

Завершение загрузки отслеживается по изменению атрибута "progress":

- В режиме вывода `pretty` (по умолчанию) успешно завершенная операция отражается значением "Done" в выделенном псевдографикой поле `progress` :

```
┌──────  ...  ──┬────────┬──────────┬────────────┬─...
| id           | ready | status   | progress | ...
├──────...  ──┼────────┼──────────┼────────────┼─...
| ydb://...    | true  | SUCCESS | Done      | ...
├-----  ...  --┴-------┴---------┴----------┴- ...
...
```

- В режиме вывода `proto-json-base64` завершенная операция отражается значением `PROGRESS_DONE` атрибута `progress` :

```
{"id":"ydb://...", ...,"progress":"PROGRESS_DONE",... }
```

Завершение операции загрузки

После выполнения загрузки воспользуйтесь командой `operation forget` для того, чтобы загрузка была удалена из перечня операций:

```
ydb -p quickstart operation forget "ydb://import/8?id=281474976788395&kind=fs"
```

Список операций загрузки

Для получения списка операций загрузки воспользуйтесь командой `operation list import/nfs` :

```
ydb -p quickstart operation list import/nfs
```

Формат вывода `operation list` также устанавливается опцией `--format`.

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Загрузка в корень базы данных

Загрузка в корень базы данных содержимого директории `/mnt/nfs/backups/export1` на файловой системе:

```
ydb -p quickstart import nfs \
  --fs-path /mnt/nfs/backups/export1
```

Загрузка нескольких директорий

Загрузка объектов из директорий `dir1` и `dir2` выгрузки, расположенной в `/mnt/nfs/backups/export1` на файловой системе, в одноименные директории базы данных:

```
ydb -p quickstart import nfs \
  --fs-path /mnt/nfs/backups/export1 \
  --include dir1 --include dir2
```

Загрузка зашифрованной выгрузки

Загрузка одной таблицы, которая была выгружена по пути `dir/my_table`, в путь `dir1/dir/my_table` из зашифрованной выгрузки, расположенной в `/mnt/nfs/backups/export1` на файловой системе, с использованием секретного ключа из файла `~/my_secret_key`.

```
ydb -p quickstart import nfs \
  --fs-path /mnt/nfs/backups/export1 --destination-path dir1 \
  --include dir/my_table \
  --encryption-key-file ~/my_secret_key
```

Получение идентификаторов операций

Для получения перечня идентификаторов операций загрузки в удобном для обработки в скриптах bash формате вы можете применить утилиту jq:

```
ydb -p quickstart operation list import/nfs --format proto-json-base64 | jq -r ".operations[].id"
```

Вы получите вывод, где в каждой новой строке находится идентификатор операции, например:

```
ydb://import/8?id=281474976789577&kind=fs
ydb://import/8?id=281474976789526&kind=fs
ydb://import/8?id=281474976788779&kind=fs
```

По этим идентификаторам может быть, например, запущен цикл для завершения всех текущих операций:

```
ydb -p quickstart operation list import/nfs --format proto-json-base64 | jq -r ".operations[].id" | while rea
d line; do ydb -p quickstart operation forget $line;done
```

Импорт данных из файла в существующую таблицу

С помощью подкоманды `import file` вы можете импортировать данные из файлов форматов CSV, JSON, Parquet, TSV в существующую таблицу.

При импорте файл читается пакетами, размер которых задан параметром `--batch-bytes`. Каждый пакет записывается в БД отдельным запросом. Запросы выполняются асинхронно. Когда число выполняемых запросов достигает указанного в `--max-in-flight`, чтение из файла приостанавливается. Вы можете импортировать данные из нескольких файлов одной командой. В этом случае данные из них будут читаться одновременно.

Импорт выполняется методом `BulkUpsert`, который обеспечивает высокоэффективную пакетную вставку большого количества строк без гарантий атомарности. Запись данных разбивается на несколько независимых транзакций, каждая из которых затрагивает единственную партицию, с параллельным исполнением. В случае успеха гарантируется вставка всех данных.

Если в таблице уже содержатся данные, они будут заменены импортируемыми при совпадении первичного ключа.

Импортируемый файл должен быть в кодировке UTF-8. Обработка переноса строки внутри поля данных не поддерживается.

> **Примечание**
>
> Если таблица еще не создана, можно использовать команду ydb tools infer csv для получения текста `CREATE TABLE` на основе имеющегося CSV-файла.
>
> При попытке выполнить импорт в несуществующую таблицу будет предложено выполнить ydb tools infer csv с нужными опциями.

Общий вид команды:

```
ydb [connection options] import file csv|json|parquet|tsv [options] <input files...>
```

, где `[connection options]` — опции соединения с БД

, `<input files>` — пути к файлам на локальной файловой системе, данные которых нужно импортировать.

Параметры подкоманды

Обязательные параметры

- `-p, --path STRING` — путь к таблице в базе данных.

Дополнительные параметры

- `--timeout VAL` — время, в течение которого должна быть выполнена операция на сервере. Значение по умолчанию: `300s`.
- `--skip-rows NUM` — количество строк от начала файла, которое будет пропущено при импорте. Значение по умолчанию: `0`.
- `--header` — укажите этот параметр, если первая строка (исключая пропущенные при использовании параметра `--skip-rows`) содержит имена столбцов данных, которые будут сопоставлены соответствующим столбцам таблицы. Если строка заголовка не указана, данные будут сопоставлены в порядке следования столбцов в схеме таблицы.
- `--delimiter STRING` — символ-разделитель столбцов данных. В этом параметре нельзя указать символ табуляции в качестве разделителя. Чтобы импортировать файл с таким разделителем, используйте подкоманду `import file tsv`. Значение по умолчанию: `,`.
- `--null-value STRING` — значение, которое будет импортировано как `NULL`. По умолчанию ни одно значение не интерпретируется как `NULL`. Чтобы интерпретировать пустое поле как `NULL`, укажите `--null-value ""`.
- `--batch-bytes VAL` — разбивать загружаемый файл на пакеты указанного размера. Если строка не умещается в пакет целиком, она будет отброшена и передана в следующем пакете. При любом значении размера пакет будет состоять минимум из одной строки. Значение по умолчанию: `1 Миб`.
- `--max-in-flight VAL` — количество одновременно загружаемых пакетов данных. Для ускорения импорта больших файлов можно увеличить значение этого параметра. Значение по умолчанию: `100`.
- `--threads VAL` — максимальное число потоков, выполняющих загрузку данных. Значение по умолчанию: число логических процессоров.
- `--columns` — список имен столбцов данных в файле, разделенный `,` для формата `csv` или символом табуляции для формата `tsv`. Если указан параметр `--header`, переданные через него имена столбцов будут заменены именами из списка. Если число столбцов в списке не совпадает с числом столбцов в данных, будет выдана ошибка.
- `--newline-delimited` — флаг гарантии отсутствия переносов строк внутри записей. Если данный флаг установлен, и загрузка производится из файла, то разные потоки загрузки будут работать с разными частями исходного файла. Это позволяет обеспечить максимальную производительность загрузки отсортированных датасетов в партицированные таблицы, за счет распределения нагрузки по всем партициям.

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Перед выполнением примеров создайте таблицу `series`.

Импортировать файл

Файл содержит данные без дополнительной информации. В качестве разделителя используется символ `,`:

```
1,Айтишники,The IT Crowd is a British sitcom.,13182
2,Silicon Valley,Silicon Valley is an American comedy television series.,16166
```

Примечание

Столбец `release_date` таблицы `series` имеет тип Date, поэтому дата релиза в файле для импорта представлена в виде числа. Чтобы импортировать значения в формате timestamp, используйте для них столбцы таблицы строкового типа, либо импортируйте их во временную таблицу и выполните преобразование к нужному типу.

Чтобы импортировать такой файл, выполните команду:

```
ydb import file csv -p series series.csv
```

В результате будут импортированы следующие данные:

```
┌──────────────┬───────────┬──────────────────────────────────────────────────────┬──────────────────┐
| release_date | series_id | series_info                                          | title            |
├──────────────┼───────────┼──────────────────────────────────────────────────────┼──────────────────┤
| "2006-02-03" | 1         | "The IT Crowd is a British sitcom."                  | "Айтишники"      |
├──────────────┼───────────┼──────────────────────────────────────────────────────┼──────────────────┤
| "2014-04-06" | 2         | "Silicon Valley is an American comedy television series." | "Silicon Valley" |
└──────────────┴───────────┴──────────────────────────────────────────────────────┴──────────────────┘
```

Импортировать несколько файлов

Следующие файлы содержат данные в формате CSV без дополнительной информации:

- series1.csv:

  ```
  1,Айтишники,The IT Crowd is a British sitcom.,13182
  ```

- series2.csv:

  ```
  2,Silicon Valley,Silicon Valley is an American comedy television series.,16166
  ```

Чтобы импортировать такие файлы, выполните команду:

```
ydb import file csv -p series series1.csv series2.csv
```

Импортировать файл с разделителем `|`

Файл содержит данные без дополнительной информации. В качестве разделителя используется символ `|`:

```
1|IT Crowd|The IT Crowd is a British sitcom.|13182
2|Silicon Valley|Silicon Valley is an American comedy television series.|16166
```

Чтобы импортировать такой файл, укажите параметр `--delimiter` со значением `|`:

```
ydb import file csv -p series --delimiter "|" series.csv
```

Пропустить строки и считать заголовки столбцов

Файл содержит дополнительную информацию в первой и второй строке, а также заголовки столбцов в третьей. Порядок данных в строках файла не соответствует порядку столбцов в таблице:

```
#The file contains data about the series.
#
series_id,title,release_date,series_info
1,Айтишники,13182,The IT Crowd is a British sitcom.
2,Silicon Valley,16166,Silicon Valley is an American comedy television series.
```

Чтобы пропустить комментарии в первой и второй строке, укажите параметр `--skip-rows 2`. Чтобы обработать строку третью строку как заголовки и сопоставить данные файла нужным столбцам таблицы, укажите параметр `--header`:

```
ydb import file csv -p series --skip-rows 2 --header series.csv
```

Заменить значения на `NULL`

Файл содержит часто используемое для `NULL` обозначение — `\N`, а также пустую строку.

```
1,Айтишники,The IT Crowd is a British sitcom.,13182
2,Silicon Valley,"",\N
3,Lost,,\N
```

Укажите параметр `--null-value "\N"`, чтобы интерпретировать значение `\N` как `NULL`:

```
ydb import file csv -p series --null-value "\N" series.csv
```

В результате будут импортированы следующие данные:

```
┌──────────────┬───────────┬──────────────────────────────────────┬──────────────────┐
| release_date | series_id | series_info                          | title            |
├──────────────┼───────────┼──────────────────────────────────────┼──────────────────┤
| "2006-02-03" | 1         | "The IT Crowd is a British sitcom." | "Айтишники"      |
├──────────────┼───────────┼──────────────────────────────────────┼──────────────────┤
| null         | 2         | ""                                   | "Silicon Valley" |
├──────────────┼───────────┼──────────────────────────────────────┼──────────────────┤
| null         | 3         | ""                                   | "Lost"           |
└──────────────┴───────────┴──────────────────────────────────────┴──────────────────┘
```

Команды для работы с топиком

С помощью команд YDB CLI вы можете выполнять следующие операции:

- Создание топика.
- Изменение топика.
- Удаление топика.
- Добавление читателя топика.
- Удаление читателя топика.
- Сохранение позиции чтения.
- Чтение из топика.
- Запись в топик.

Создание топика

С помощью подкоманды `topic create` вы можете создать новый топик.

Общий вид команды:

```
ydb [global options...] topic create [options...] <topic-path>
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.
- `topic-path` — путь топика.

Посмотрите описание команды создания топика:

```
ydb topic create --help
```

Параметры подкоманды

Имя	Описание
`--partitions-count`	Количество партиций топика. Значение по умолчанию — `1`.
`--retention-period`	Время хранения данных в топике. Положительное число с указанием единицы измерения. Поддерживаются следующие единицы: • `s` – секунды; • `m` – минуты; • `h` – часы; • `d` – дни. Значение по умолчанию — `24h`.
`--partition-write-speed-kbps`	Максимальная скорость записи в партицию, задается в КБ/с. Значение по умолчанию — `1024`.
`--retention-storage-mb`	Максимальный объем хранения для топика, задается в МБ. При достижении ограничения будут удаляться самые старые данные. При включенном автоматическом партиционировании потребляемое место может превышать установленное значение. Значение по умолчанию — `0` (ограничение не задано).
`--auto-partitioning-strategy`	Режим автопартиционирования. Возможные значения: • `up` — количество партиций топика при изменении нагрузки может увеличиваться, но не уменьшаться; • `paused` — автопартиционирование на топике приостановлено; • `disabled` — автопартиционирование отключено. Значение по умолчанию — `disabled`.
`--supported-codecs`	Поддерживаемые методы сжатия данных. Задаются через запятую. Значение по умолчанию — `raw`. Возможные значения: • `RAW` — без сжатия; • `ZSTD` — сжатие zstd; • `GZIP` — сжатие gzip; • `LZOP` — сжатие lzop.
`--metering-mode`	Режим тарификации топика для serverless базы данных. Возможные значения: • `request-units` — по фактическому использованию. • `reserved-capacity` — по выделенным ресурсам.

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Создание топика с 2 партициями, методами сжатия `RAW` и `GZIP`, временем хранения сообщений 2 часа и путем `my-topic`:

```
ydb -p quickstart topic create \
  --partitions-count 2 \
  --supported-codecs raw,gzip \
  --retention-period 2h \
  my-topic
```

Посмотрите параметры созданного топика:

```
ydb -p quickstart scheme describe my-topic
```

Результат:

```
RetentionPeriod: 2h
PartitionsCount: 2
SupportedCodecs: RAW, GZIP
```

Изменение топика

С помощью подкоманды `topic alter` вы можете изменить созданный ранее топик.

Общий вид команды:

```
ydb [global options...] topic alter [options...] <topic-path>
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.
- `topic-path` — путь топика.

Посмотрите описание команды изменения топика:

```
ydb topic alter --help
```

Параметры подкоманды

При исполнении команды будут изменены значения тех параметров, которые заданы в командной строке. Значения остальных параметров останутся без изменений.

Имя	Описание
`--partitions-count`	Количество партиций топика. Возможно только увеличение количества партиций.
`--retention-period`	Время хранения данных в топике. Положительное число с указанием единицы измерения. Поддерживаются следующие единицы: • `s` – секунды; • `m` – минуты; • `h` – часы; • `d` – дни.
`--partition-write-speed-kbps`	Максимальная скорость записи в партицию, задается в КБ/с. Значение по умолчанию — `1024`.
`--retention-storage-mb`	Максимальный объем хранения для топика, задается в МБ. При достижении ограничения будут удаляться самые старые данные. При включенном автоматическом партиционировании потребляемое место может превышать установленное значение. Значение по умолчанию — `0` (ограничение не задано).
`--auto-partitioning-strategy`	Режим автопартиционирования. Возможные значения: • `up` — количество партиций топика при изменении нагрузки может увеличиваться, но не уменьшаться; • `paused` — автопартиционирование на топике приостановлено.
`--supported-codecs`	Поддерживаемые методы сжатия данных. Возможные значения: • `RAW` — без сжатия; • `ZSTD` — сжатие zstd; • `GZIP` — сжатие gzip; • `LZOP` — сжатие lzop.
`--metering-mode`	Режим тарификации топика для serverless базы данных. Возможные значения: • `request-units` — по фактическому использованию. • `reserved-capacity` — по выделенным ресурсам.

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Добавьте партицию и метод сжатия `lzop` созданному ранее топику:

```
ydb -p quickstart topic alter \
  --partitions-count 3 \
  --supported-codecs raw,gzip,lzop \
  my-topic
```

Убедитесь, что параметры топика изменились:

```
ydb -p quickstart scheme describe my-topic
```

Результат:

```
RetentionPeriod: 2h
PartitionsCount: 3
SupportedCodecs: RAW, GZIP, LZOP
```

Удаление топика

С помощью подкоманды `topic drop` вы можете удалить созданный ранее топик.

> **Примечание**
>
> При удалении топика также будут удалены все добавленные для него читатели.

Общий вид команды:

```
ydb [global options...] topic drop <topic-path>
```

- `global options` — глобальные параметры.
- `topic-path` — путь топика.

Посмотрите описание команды удаления топика:

```
ydb topic drop --help
```

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Удалите созданный ранее топик:

```
ydb -p quickstart topic drop my-topic
```

Добавление читателя топика

С помощью команды `topic consumer add` вы можете добавить читателя для созданного ранее топика.

Общий вид команды:

```
ydb [global options...] topic consumer add [options...] <topic-path>
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.
- `topic-path` — путь топика.

Посмотрите описание команды добавления читателя:

```
ydb topic consumer add --help
```

Параметры подкоманды

Имя	Описание
`--consumer VAL`	Имя читателя, которого нужно добавить.
`--starting-message-timestamp VAL`	Время в формате UNIX timestamp (секунды с 1970.01.01) или в формате ISO-8601 (например, `2020-07-10T15:00:00Z`). Чтение начнется с первого сообщения, полученного после указанного времени. Если время не задано, то чтение начнется с самого старого сообщения в топике.
`--supported-codecs`	Поддерживаемые методы сжатия данных. Значение по умолчанию — `raw`. Возможные значения: • `RAW` — без сжатия; • `ZSTD` — сжатие zstd; • `GZIP` — сжатие gzip; • `LZOP` — сжатие lzop.
`--important`	Указывает, является ли читатель важным. Значение по умолчанию — `false`. Для важных читателей: • не применяется ограничение по периоду доступности (`--availability-period`); • данные в топике не удаляются, пока они не обработаны всеми важными читателями; • это влияет на процесс очистки данных в топике. Используйте этот параметр для критически важных читателей, которые должны гарантированно прочитать и обработать все сообщения.
`--availability-period VAL`	Время доступности данных в топике для необработанных сообщений. Опция позволяет продлить время хранения сообщений в топике с времени хранения `retention-period` вплоть до указанного времени доступности, если читатель не подтверждает их обработку. Формат: положительное число с указанием единицы измерения времени (без пробелов). Поддерживаются следующие единицы измерения: • `s` — секунды (например, `30s`, `120s`); • `m` — минуты (например, `5m`, `1440m`); • `h` — часы (например, `1h`, `72h`); • `d` — дни (например, `1d`, `7d`). Примеры: `72h`, `1440m`, `2d`, `3600s`. Для важных читателей (с параметром `--important`) этот параметр не применяется.

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Создайте читателя с именем `my-consumer` для созданного ранее топика `my-topic`, чтение начнется с первого сообщения, полученного после 15 августа 2022 13:00:00 GMT

```
ydb -p quickstart topic consumer add \
  --consumer my-consumer \
  --starting-message-timestamp 1660568400 \
  my-topic
```

Убедитесь, что читатель создан:

```
ydb -p quickstart scheme describe my-topic
```

Результат:

```
RetentionPeriod: 2h
PartitionsCount: 2
SupportedCodecs: RAW, GZIP

Consumers:
┌──────────────┬─────────────────┬──────────────────────────────┬─────────────┐
| ConsumerName | SupportedCodecs | ReadFrom                     | Important |
├──────────────┼─────────────────┼──────────────────────────────┼─────────────┤
| my-consumer  | RAW, GZIP       | Mon, 15 Aug 2022 16:00:00 MSK | 0         |
└──────────────┴─────────────────┴──────────────────────────────┴─────────────┘
```

Создайте читателя с именем `backup-consumer` для топика `my-topic` с периодом доступности данных 3 дня

```
ydb -p quickstart topic consumer add \
  --consumer backup-consumer \
  --availability-period 3d \
  my-topic
```

Если читатель успевает обрабатывать данные, то сообщения в топике будут храниться в течение 2-х часов, в соответствии со значением параметра `retention-period` топика.
Однако при временной остановке чтения, время хранения данных в топике, для которых читатель `backup-consumer` ещё не подтвердил обработку, будет увеличено вплоть до 3-х дней, в соответствии с параметром `availability-period`.

Создайте важного читателя с именем `critical-consumer` для топика `my-topic` с дополнительной поддержкой кодека `ZSTD`

```
ydb -p quickstart topic consumer add \
  --consumer critical-consumer \
  --important \
  --supported-codecs raw,gzip,zstd \
  my-topic
```

Обратите внимание, что для важного читателя параметр `--availability-period` не применяется, даже если он указан.

Удаление читателя топика

С помощью команды `topic consumer drop` вы можете удалить добавленного ранее читателя.

Общий вид команды:

```
ydb [global options...] topic consumer drop [options...] <topic-path>
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.
- `topic-path` — путь топика.

Посмотрите описание команды удаления читателя:

```
ydb topic consumer drop --help
```

Параметры подкоманды

Имя	Описание
`--consumer VAL`	Имя читателя, которого нужно удалить.

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Удалите созданного ранее читателя с именем `my-consumer` для топика `my-topic`:

```
ydb -p quickstart topic consumer drop \
  --consumer my-consumer \
  my-topic
```

Сохранение позиции чтения

Каждый читатель топика обладает позицией чтения.

С помощью команды `topic consumer offset commit` можно сохранить позицию чтения добавленного ранее читателя.

Общий вид команды:

```
ydb [global options...] topic consumer offset commit [options...] <topic-path>
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.
- `topic-path` — путь топика.

Посмотреть описание команды:

```
ydb topic consumer offset commit --help
```

Параметры подкоманды

Имя	Описание
`--consumer <значение>`	Имя читателя.
`--partition <значение>`	Номер партиции.
`--offset <значение>`	Устанавливаемое значение смещения.

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Установить для читателя с именем `my-consumer` смещение 123456789 в топике `my-topic` и партиции `1`:

```
ydb -p db1 topic consumer offset commit \
  --consumer my-consumer \
  --partition 1 \
  --offset 123456789 \
  my-topic
```

Чтение из топика

Командой `topic read` выполняется чтение сообщений из топика с выводом их в файл или в терминал командной строки:

```
ydb [connection options] topic read <topic-path> [--consumer STR] \
  [--format STR] [--wait] [--limit INT] \
  [--transform STR] [--file STR] [--commit BOOL] \
  [дополнительные параметры...]
```

, где `[connection options]` — опции соединения с БД

Поддерживается три режима работы команды:

1. **Одно сообщение**. Из топика считывается не более одного сообщения.
2. **Пакетный режим**. Сообщения из топика считываются до того момента, пока в топике не закончатся сообщения для обработки, или количество сообщений не превысит устанавливаемое обязательно ограничение.
3. **Потоковый режим**. Сообщения из топика считываются по мере их появления, с ожиданием появления новых, до момента пока исполнение команды не будет прервано нажатием `Ctrl+C`, или количество сообщений не превысит опционально задаваемое ограничение.

Параметры

Обязательные параметры

`<topic-path>`: Путь топика

Основные опциональные параметры

`-c VAL`, `--consumer VAL`: Имя читателя топика.

- Если имя читателя не задано, то необходимо указать значение параметра `--partition-ids`. Только в этом случае возможно чтение из топика без указания имени читателя.
- Чтение сообщений будет начато с текущей рабочей позиции (offset)
  для данного читателя (если не указан параметр `--timestamp`). Если не указать имя читателя, то чтение сообщений начнется с первого сообщения в партиции

`--format STR`: Формат вывода

- Задает правило оформления сообщений на выходе. Не все форматы могут работать в потоковом режиме.
- Перечень поддерживаемых форматов:

Имя	Описание	Поддерживает потоковый режим?
`single-message` (по умолчанию)	Вывод содержимого не более одного сообщения без оформления.	-
`pretty`	Вывод в псевдографическую таблицу с колонками, содержащими метаинформацию о сообщениях. В колонке `body` выводится само сообщение.	Нет
`newline-delimited`	Вывод сообщений с добавлением после каждого сообщения разделителя - символа перевода строки `0x0A`	Да
`concatenated`	Вывод сообщений одного за другим без добавления какого-либо разделителя	Да

`--wait` (`-w`): Ожидание появления сообщений

- Включает ожидание появления первого сообщения в топике. Если он не задан, и в топике нет сообщений для обработки, то исполнение команды будет завершено сразу после старта. Если он задан, то запущенная команда чтения будет ожидать появления первого сообщения для обработки.
- Включает потоковый режим отбора для форматов, которые его поддерживают, иначе используется пакетный режим.

`--limit INT`: Максимальное количество считываемых из топика сообщений

- Значения по умолчанию и допустимые значения зависят от выбранного формата вывода:

Поддерживает ли формат потоковый режим отбора	Значение лимита по умолчанию	Допустимые значения
Нет	10	1-500
Да	0 (без ограничений)	0-500

`--transform VAL`: Метод преобразования сообщений

- Значение по умолчанию — `none`
- Возможные значения:
  `base64` — преобразовать в кодировку Base64
  `none` — не выполнять преобразований, передать на выход содержимое сообщения побайтово

`--file VAL` (`-f VAL`): Записывать читаемые сообщения в указанный файл. Если параметр не задан, то сообщения выводятся в `stdout`.

`--commit BOOL`: Подтверждение чтения. Значение по умолчанию - `false`

- Возможные значения: `true`, `false`.
- Если установлено значение `true`, то текущая рабочая позиция (offset) читателя в топике будет сохраняться по мере чтения сообщений из топика.
- Если установлено значение `false`, то сообщения будут вычитываться, но прогресс чтения не будет сохраняться и при перезапуске сообщения будут вычитаны заново. Эта функциональность полезна при дебаге: для вычитывания

сообщений без влияния на продуктовую систему (без смещение offset).

Другие опциональные параметры

Имя	Описание
`--idle-timeout VAL`	Таймаут принятия решения о том что топик пуст, то есть новые сообщения для обработки отсутствуют. Замеряется время с момента установки соединения при запуске команды, или получения последнего сообщения. Если в течение заданного таймаута с сервера не приходит новых сообщений, топик считается пустым. Значение по умолчанию — `1s` (1 секунда).
`--timestamp VAL`	Чтение с заданного в формате UNIX timestamp времени. Если параметр не указан, то чтение выполняется с текущей рабочей позиции читателя в топике. Если параметр указан, то чтение начнется с первого сообщения, полученного после указанного времени.
`--metadata-fields VAL`	Список атрибутов сообщения, значения которых нужно выводить в колонках с метаинформацией в формате `pretty`. Если параметр не указан, то выводятся колонки со всеми атрибутами. Возможные значения: • `write_time` — время записи сообщения на сервер в формате UNIX timestamp; • `meta` — метаданные сообщения; • `create_time` — время создания сообщения источником в формате UNIX timestamp; • `seq_no` — порядковый номер сообщения; • `offset` — порядковый номер сообщения внутри партиции; • `message_group_id` — идентификатор группы сообщений; • `body` — тело сообщения.
`--partition-ids VAL`	Идентификаторы (порядковые номера) партиций, из которых будет производиться чтение. Если параметр не указан, то чтение производится из всех партиций.

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Все примеры используют имя топика `topic1` и имя читателя `c1`.

- Чтение одного сообщения с выводом в терминал. Если в топике нет новых сообщения для данного читателя, исполнение команды будет завершено без какого-либо вывода:

```
ydb -p quickstart topic read topic1 -c c1
```

- Ожидание появления и чтение одного сообщения с записью его в файл `message.bin`. До тех пор пока в топике нет новых сообщений для данного читателя, команда будет оставаться запущенной, но её можно прервать нажав `Ctrl+C`:

```
ydb -p quickstart topic read topic1 -c c1 -w -f message.bin
```

- Просмотр информации об ожидающих обработки читателем сообщениях без их подтверждения. Будет выведено до 10 первых сообщений:

```
ydb -p quickstart topic read topic1 -c c1 --format pretty --commit false
```

- Вывод в терминал сообщений по мере их появления в формате разделения символами перевода строки, с конвертацией в кодировку Base64. Команда будет исполняться до прерывания нажатием `Ctrl+C`:

```
ydb -p quickstart topic read topic1 -c c1 -w --format newline-delimited --transform base64
```

- Отслеживание появления в топике сообщений, содержащих текст `ERROR`, с выводом их в терминал по мере появления:

```
ydb -p quickstart topic read topic1 -c c1 --format newline-delimited -w | grep ERROR
```

- Получение следующего непустого пакета из не более 150 сообщений, преобразованных в base64, с разделением символом перевода строки, и записью в файл `batch.txt`:

```
ydb -p quickstart topic read topic1 -c c1 \
  --format newline-delimited -w --limit 150 \
  --transform base64 -f batch.txt
```

- Примеры интеграции команд YDB CLI

Запись в топик

Командой `topic write` выполняется запись сообщений в топик из файла или `stdin`:

```
ydb [connection options] topic write <topic-path> \
  [--file STR] [--format STR] [--transform STR] \
  [дополнительные параметры...]
```

, где `[connection options]` — опции соединения с БД

Параметры

Основные параметры

`<topic-path>` : Путь топика, единственный обязательный параметр

`--file VAL` (`-f VAL`): Читать поток входящих сообщений для записи в топик из указанного файла. Если не указан -- чтение производится из `stdin`.

`--format STR` : Формат потока входящих сообщений. Поддерживаемые форматы:

Имя	Описание
`single-message` (по умолчанию)	Все содержимого потока на входе рассматривается как одно сообщение для записи в топик
`newline-delimited`	Поток на входе содержит множество сообщений, разделенных символом перевода строки `0x0A`

`--transform VAL` : Метод преобразования сообщений

- Значение по умолчанию — `none`
- Возможные значения:

  `base64` — выполнить декодирование каждого сообщения из Base64 на входном потоке, записать результат декодирования в топик. Если декодирование не удалось, исполнение команды будет прервано с ошибкой.
  `none` — не выполнять преобразований, записать в топик содержимое сообщения из входного потока побайтово

Дополнительные параметры

Имя	Описание
`--delimiter STR`	Байт-разделитель. Поток на входе разделяется на сообщения указанным байтом. Может быть задан только в случае, если не указан `--format`. Задается в виде экранированной строки.
`--message-group-id STR`	Строковой идентификатор группы сообщений. Если не указан, то все сгенерированные из входного потока сообщения получат одинаковое значение идентификатора, представляющее собой шестнадцатеричное строковое представление случайного трехбайтового целого числа.
`--codec STR`	Кодек, используемый для сжатия сообщений на клиенте перед отправкой на сервер. Возможные варианты: `RAW` (по умолчанию) - без сжатия, `GZIP`, `ZSTD`. Сжатие увеличивает затраты CPU на клиенте при записи и чтении сообщений, но обычно позволяет уменьшить объем передаваемых по сети и хранимых данных. При последующем чтении сообщений подписчиками они автоматически расжимаются использованным при записи кодеком, не требуя указания каких-либо специальных опций. Указанный кодек должен быть перечислен среди допустимых в параметрах топика.

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Все примеры используют имя топика `topic1`.

- Запись ввода с терминала в одно сообщение. После запуска команды можно ввести произвольный многострочный текст, завершив ввод нажатием `Ctrl+D`.

  ```
  ydb -p quickstart topic write topic1
  ```

- Запись содержимого бинарного файла `message.bin` в одно сообщение с использованием сжатия кодеком GZIP

  ```
  ydb -p quickstart topic write topic1 -f message.bin --codec GZIP
  ```

- Запись содержимого строкового файла `example.txt` с разделением на сообщения по строкам

  ```
  ydb -p quickstart topic write topic1 -f example.txt --format newline-delimited
  ```

- Запись скачивамого по протоколу HTTP ресурса с разделением на сообщения символами табуляции

  ```
  curl http://example.com/resource | ydb -p quickstart topic write topic1 --delimiter "\t"
  ```

- Примеры интеграции команд YDB CLI

Конвейерная обработка сообщений

Возможности работы команд `topic read` и `topic write` со стандартными устройствами ввода/вывода, а также поддержка потокового режима на чтении, позволяет выстраивать полноценные сценарии интеграции с передачей сообщений между топиками и их преобразованиями. В данном разделе собраны несколько примеров подобных сценариев.

- Перекладывание одного сообщения из `topic1` в базе данных `quickstart` в `topic2` в базе данных `db2`, с ожиданием его появления в топике-источнике

```
ydb -p quickstart topic read topic1 -c c1 -w | ydb -p db2 topic write topic2
```

- Фоновая передача всех появляющихся однострочных сообщений в топике `topic1` в базе данных `quickstart`, в топик `topic2` в базе данных `db2`. Данный сценарий можно использовать в случае, если гарантируется отсутствие байтов `0x0A` (перевод строки) в исходных сообщениях.

```
ydb -p quickstart topic read topic1 -c c1 --format newline-delimited -w | \
ydb -p db2 topic write topic2 --format newline-delimited
```

- Фоновая передача точной бинарной копии всех появляющихся сообщений в топике `topic1` в базе данных `quickstart`, в топик `topic2` в базе данных `db2`, с использованием кодировки сообщений base64 на потоке передачи.

```
ydb -p quickstart topic read topic1 -c c1 --format newline-delimited -w --transform base64 | \
ydb -p quickstart topic write topic2 --format newline-delimited --transform base64
```

- Передача ограниченного пакета однострочных сообщений с фильтрацией по подстроке `ERROR`

```
ydb -p quickstart topic read topic1 -c c1 --format newline-delimited | \
grep ERROR | \
ydb -p db2 topic write topic2 --format newline-delimited
```

- Запись результата исполнения YQL-запроса в виде сообщений в топик `topic1`

```
ydb -p quickstart yql -s "select * from series" --format json-unicode | \
ydb -p quickstart topic write topic1 --format newline-delimited
```

Исполнение YQL-запроса с передачей сообщений из топика в качестве параметров

- Исполнение YQL-запроса с передачей параметром каждого сообщения, считанного из топика `topic1`

```
ydb -p quickstart topic read topic1 -c c1 --format newline-delimited -w | \
ydb -p quickstart sql -s 'declare $s as String;select Len($s) as Bytes' \
--input-framing newline-delimited --input-param-name s --input-format raw
```

- Исполнение YQL-запроса с адаптивным пакетированием параметров из сообщений, считанных из топика `topic1`

```
ydb -p quickstart topic read topic1 -c c1 --format newline-delimited -w | \
ydb -p quickstart sql \
-s 'declare $s as List<String>;select ListLength($s) as Count, $s as Items' \
--input-framing newline-delimited --input-param-name s --input-format raw \
--input-batch adaptive
```

Обзор

Для выполнения запросов в YDB CLI рекомендуется использовать следующие команды:

1. ydb sql — единая команда для выполнения любого SQL запроса, поддерживаемого YDB.
2. ydb — переключает консоль в интерактивный режим выполнения запросов.

Выполнение параметризованных запросов описано на отдельной странице.

Выполнение запросов

С помощью подкоманды `ydb sql` вы можете выполнить SQL-запрос. Запрос может быть любого типа (DDL, DML и т.д.), а так же состоять из нескольких подзапросов. Подкоманда `ydb sql` устанавливает стрим и получает данные через него. Выполнение запроса в стриме позволяет снять ограничение на размер читаемых данных. Эта команда также позволяет записывать данные в YDB, что более эффективно при выполнении повторяющихся запросов с передачей данных через параметры.

Общий вид команды:

```
ydb [global options...] sql [options...]
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.

Посмотрите описание команды выполнения запроса:

```
ydb sql --help
```

Параметры подкоманды

Имя	Описание
`-h`, `--help`	Выводит общую справку по использованию команды.
`-hh`	Выводит полную справку по использованию команды. Вывод содержит некоторые специфичный команды, которых нет в выводе `--help`.
`-s`, `--script`	Текст скрипта(запроса) для выполнения.
`-f`, `--file`	Путь к файлу, содержащему текст запроса для выполнения. Путь `-` означает, что текст запроса будет прочитан из `stdin`, при этом передача параметров через `stdin` будет невозможна.
`--stats`	Режим сбора статистики. Возможные значения: • `none` (по умолчанию) — не собирать; • `basic`: Собирать обобщенную статистику по обновлениям(update) и удалениям(delete) в таблицах. • `full`: К тому, что содержит режим `basic` добавляется статистика и план выполнения запроса. • `profile`: Собирать детальную статистику выполнения, содержащую статистику по каждому отдельному таску и каналу выполнения запроса.
`--explain`	Выполнить explain-запрос, будет выведен логический план запроса. Сам запрос не будет выполнен, поэтому не затронет данные в базе.
`--explain-ast`	То же, что и `--explain`, но вдобавок к логическому плану выводит AST (abstract syntax tree). Раздел с AST содержит представление на внутреннем языке miniKQL.
`--explain-analyze`	Выполнить запрос в режиме `EXPLAIN ANALYZE`. Показывает план выполнения запроса. Возвращаемые в рамках запроса данные игнорируются. **Важное замечание: Запрос фактически выполняется, поэтому может внести изменения в базу**.
`--format`	Формат вывода. Возможные значения: • `pretty` (по умолчанию) — человекочитаемый формат; • `json-unicode` — вывод в формате JSON, бинарные строки закодированы в юникод, каждая строка JSON выводится в отдельной строке; • `json-unicode-array` — вывод в формате JSON, бинарные строки закодированы в Юникод, результат выводится в виде массива строк JSON, каждая строка JSON выводится в отдельной строке; • `json-base64` — вывод в формате JSON, бинарные строки закодированы в Base64, каждая строка JSON выводится в отдельной строке; • `json-base64-array` — вывод в формате JSON, бинарные строки закодированы в Base64, результат выводится в виде массива строк JSON, каждая строка JSON выводится в отдельной строке; • `parquet`: вывод в формате Apache Parquet. • `csv` — вывод в формате CSV; • `tsv` — вывод в формате TSV.

Работа с параметризованными запросами

Подробное описание работы с параметрами с примерами смотрите в статье Выполнение параметризованных запросов.

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Совместное выполнение DDL + DML в одном запросе не поддерживается.

```
# Создание таблицы
ydb -p quickstart sql -s '
```

```
  CREATE TABLE series (
    series_id Uint64,
    title Utf8,
    series_info Utf8,
    release_date Date,
    PRIMARY KEY (series_id)
  );
'

# Заполнение данными и получение выборки
ydb -p quickstart sql -s '
    UPSERT INTO series (series_id, title, series_info, release_date)
    VALUES (1, "Title1", "Info1", Cast("2023-04-20" as Date));
    SELECT * FROM series;
'

# Добавление индекса
ydb -p quickstart sql -s '
    ALTER TABLE series
    ADD INDEX title_idx GLOBAL ON (title);
'
```

Вывод команды:

```
┌──────────────┬───────────┬──────────────┬──────────┐
| release_date | series_id | series_info | title    |
├──────────────┼───────────┼──────────────┼──────────┤
| "2023-04-20" | 1         | "Info1"     | "Title1" |
└──────────────┴───────────┴──────────────┴──────────┘
```

Для выполнения запроса из файла (например, script1.yql) с выводом в формате JSON

```
ydb -p quickstart sql -f script1.yql --format json-unicode
```

Вывод команды:

```
{"release_date":"2023-04-20","series_id":1,"series_info":"Info1","title":"Title1"}
```

Примеры передачи параметров в скрипты приведены в статье о передаче параметров в команды исполнения запросов.

Выполнение параметризованных запросов

Обзор

YDB CLI поддерживает исполнение параметризованных запросов. Для работы с параметрами в тексте запроса должны присутствовать их определения командой YQL `DECLARE`.

Основной инструмент для выполнения параметризованных запросов в YDB CLI — это команда ydb sql.

Зачем использовать параметризованные запросы?

Использование параметризованных запросов предоставляет несколько важных преимуществ:

- **Улучшенная производительность:** Параметризованные запросы значительно повышают производительность при массовом выполнении однотипных операций, которые различаются только входными значениями. Это происходит благодаря кешированию плана запроса. После первой компиляции запроса его план сохраняется в LRU-кеше узла YDB. Все последующие вызовы того же запроса с новыми параметрами избегают стадии компиляции и начинают выполняться сразу, что экономит время и ресурсы.
- **Защита от SQL-инъекций:** Другим важным преимуществом использования параметризованных запросов является защита от SQL-инъекций. Эта функция безопасности гарантирует правильную обработку входных данных, что снижает риск выполнения вредоносного кода.

Единичное исполнение запроса

Эта команда поддерживает передачу параметров через опции командной строки, файл, а также через `stdin`. При передаче параметров через `stdin` или файл поддерживается многократное поточное исполнение запроса с разными значениями параметров и возможностью пакетирования. Для этого в команде ydb sql предназначены следующие параметры:

Имя	Описание
`-p, --param`	Значение одного параметра запроса в формате `$name=value` или `name=value`, где `name` — имя параметра, а `value` — его значение (корректный JSON value).
`--input-file`	Имя файла в формате JSON в кодировке UTF-8, в котором заданы значения параметров, сопоставляемые с параметрами запроса по именам ключей. Может быть использован максимум один файл с параметрами.
`--input-format`	Формат представления значений параметров. Действует на все способы их передачи (через параметр команды, файл или `stdin`). Возможные значения: • `json` (по умолчанию): Формат JSON. • `csv` — формат CSV. По умолчанию имена параметров должны находиться в header'е CSV файла. При единичном исполнении запроса допустима только одна строка в файле, не считая header'а. • `tsv` — формат TSV. • `raw`: Входной поток из `stdin` или `--input-file` содержит только значение параметра в виде бинарных данных. Имя параметра должно быть указано опцией `--input-param-name`.
`--input-binary-strings`	Формат кодировния значения параметров с типом «бинарная строка» (`DECLARE $par AS String`). Говорит о том, как должны интерпретироваться бинарные строки из входного потока. Возможные значения: • `unicode`: Каждый байт в бинарной строке, не являющийся печатаемым ASCII-символом (codes 32-126), должен быть закодирован в кодировке UTF-8. • `base64`: Бинарные строки представлены в кодировке Base64. Такая возможность позволяет передавать бинарные данные, декодирование которых из Base64 выполнит YDB CLI.

Если значения указаны для всех параметров, которые не допускают `NULL` (то есть с типом NOT NULL), в составе оператора `DECLARE`, запрос будет выполнен на сервере. Если отсутствует значение хотя бы для одного такого параметра, выполнение команды завершится с ошибкой и сообщением "Не задано значение для параметра".

Более специфичные опции для использования входных параметров

Следующие опции не представлены в выводе `--help`. Их описание можно увидеть только в выводе `-hh`.

Имя	Описание
`--input-framing`	Задает фрейминг для входного потока (`stdin` или `--input-file`). Определяет как входной поток будет разделяться на отдельные наборы параметров. Возможные значения: • `no-framing` (по умолчанию): От входного потока ожидается один набор параметров, запрос исполняется однократно. • `newline-delimited`: Символ перевода строки отмечает во входном потоке окончание одного набора параметров, отделяя его от следующего. Набор параметров считается собранным каждый раз при получении во входном потоке символа перевода строки.
`--input-param-name`	Имя параметра, значение которого передано во входной поток. Указывается без символа `$`. Обязательно при использовании формата `raw` в `--input-format`. При использовании с JSON-форматом входной поток интерпретируется не как JSON-документ, а как JSON value, с передачей значения в параметр с указанным именем.
`--input-columns`	Строка с именами колонок, заменяющими header CSV/TSV документа, читаемого из входного потока. При указании опции считается, что header отсутствует. Опция допустима только с форматами CSV и TSV.
`--input-skip-rows`	Число строк с начала данных, читаемых со stdin'а, которые нужно пропустить, не включая строку header'а, если она имеется. Опция допустима только с форматами stdin'а CSV и TSV.

`--input-batch`	Режим пакетирования значений наборов параметров, получаемых из входного потока (`stdin` или `--input-file`). Возможные значения: • `iterative` (по умолчанию): Пакетирование выключено. Запрос выполняется для каждого набора параметров (ровно один раз, если в опции `--input-framing` используется `no-framing`) • `full` : Полный пакет. Запрос выполнится один раз после завершения чтения входного потока, все полученные наборы параметров заворачиваются в `List<...>` , имя параметра задается опцией `--input-param-name` • `adaptive` : Адаптивное пакетирование. Запрос выполняется каждый раз, когда срабатывает ограничение на количество наборов параметров в одном запросе (`--input-batch-max-rows`) или на задержку обработки (`--input-batch-max-delay`). Все полученные к этому моменту наборы параметров заворачиваются в `List<...>` , имя параметра задается опцией `--input-param-name` .
`--input-batch-max-rows`	Максимальное количество наборов параметров в пакете для адаптивного режима пакетирования. Следующий пакет будет отправлен на исполнение вместе с запросом, если количество наборов данных в нем достигло указанного значения. Установка в `0` снимает ограничение. Значение по умолчанию — `1000` . Параметры передаются в запрос без стриминга и общий объем одного GRPC-запроса, в который включаются значения параметров, имеет верхнюю границу около 5 МБ.
`--input-batch-max-delay`	Максимальная задержка отправки на обработку полученного набора параметров для адаптивного режима пакетирования. Задается в виде числа с размерностью времени - `s` (секунды), `ms` (миллисекунды), `m` (минуты) и т.д. Значение по умолчанию — `1s` (1 секунда). YDB CLI будет отсчитывать время с момента получения первого набора параметров для пакета и отправит накопившийся пакет на исполнение, как только время превысит указанное значение. Параметр позволяет получить эффективное пакетирование в случае непредсказуемого темпа появления новых наборов параметров на `stdin` .

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart` , подробнее смотрите в Создание профиля для соединения с тестовой БД.

Передача значения одного параметра

В командной строке, через опцию `--param` :

```
ydb -p quickstart sql -s 'DECLARE $a AS Int64; SELECT $a' --param '$a=10'
```

Через файл в формате JSON (который используется по умолчанию):

```
echo '{"a":10}' > p1.json
ydb -p quickstart sql -s 'DECLARE $a AS Int64; SELECT $a' --input-file p1.json
```

Через `stdin` , передавая JSON-строку как набор из одного параметра:

```
echo '{"a":10}' | ydb -p quickstart sql -s 'DECLARE $a AS Int64; SELECT $a'
```

Через `stdin` , передавая только значение параметра и задавая имя параметра через опцию `--input-param-name` :

```
echo '10' | ydb -p quickstart sql -s 'DECLARE $a AS Int64; SELECT $a' --input-param-name a
```

Передача значений параметров разных типов из нескольких источников

```
# Create a JSON file with fields 'a', 'b', and 'x', where 'x' will be ignored in the query
echo '{ "a":10, "b":"Some text", "x":"Ignore me" }' > p1.json

# Run the query using ydb-cli, passing in 'a' and 'b' from the input file, and 'c' as a direct parameter
ydb -p quickstart sql \
  -s 'DECLARE $a AS Int64;
      DECLARE $b AS Utf8;
      DECLARE $c AS Int64;

      SELECT $a, $b, $c' \
  --input-file p1.json \
  --param '$c=30'
```

Вывод команды:

```
┌─────────┬─────────────┬─────────┐
| column0 | column1     | column2 |
├─────────┼─────────────┼─────────┤
| 10      | "Some text" | 30      |
└─────────┴─────────────┴─────────┘
```

Передача бинарных строк в кодировке Base64

```
ydb -p quickstart sql \
  -s 'DECLARE $a AS String;
      SELECT $a' \
  --input-format json \
  --input-binary-strings base64 \
  --param '$a="SGVsbG8sIHdvcmxkCg=="'
```

Вывод команды:

```
┌──────────────────────┐
| column0              |
├──────────────────────┤
| "Hello, world\n" |
└──────────────────────┘
```

Прямая передача бинарного контента

```
curl -Ls http://ydb.tech/docs/en | ydb -p quickstart sql \
  -s 'DECLARE $a AS String;
      SELECT LEN($a)' \
  --input-format raw \
  --input-param-name a
```

Вывод команды (точное количество байт может отличаться):

```
┌──────────┐
| column0 |
├──────────┤
| 66426   |
└──────────┘
```

Передача файла в формате CSV

```
echo '10,Some text' | ydb -p quickstart sql \
  -s 'DECLARE $a AS Int32;
      DECLARE $b AS String;
      SELECT $a, $b' \
  --input-format csv \
  --input-columns 'a,b'
```

Вывод команды:

```
┌──────────┬─────────────┐
| column0 | column1     |
├──────────┼─────────────┤
| 10      | "Some text" |
└──────────┴─────────────┘
```

Итеративная потоковая обработка

YDB CLI поддерживает возможность многократного исполнения запроса с разными наборами значений параметров, при их передаче через `stdin` **или** входной файл (не одновременно). При этом соединение с базой данных устанавливается однократно, а план исполнения запроса кешируется, что существенно повышает производительность такого подхода по сравнению с отдельными вызовами CLI.

Для того чтобы воспользоваться этой возможностью, необходимо на вход команды друг за другом передавать разные наборы значений одних и тех параметров, сообщив YDB CLI правило, по которому будет возможно отделить эти наборы друг от друга.

Запрос выполняется столько раз, сколько наборов значений параметров было получено. Каждый полученный через входной поток (`stdin` или `--input-file`) набор объединяется со значениями параметров, определенными через опции `--param` . Исполнение команды будет завершено после завершения входного потока. Каждый запрос исполняется в своей транзакции.

Правило отделения наборов параметров друг от друга (фрейминг) дополняет описание формата представления параметров на входном потоке, задаваемое параметром `--input--framing` :

Имя	Описание
`--input-framing`	Задает фрейминг для входного потока (файл или `stdin`). Возможные значения: • `no-framing` (по умолчанию) — на входе ожидается один набор параметров, запрос исполняется однократно после завершения чтения потока. • `newline-delimited` — символ перевода строки отмечает на входе окончание одного набора параметров, отделяя его от следующего. Запрос исполняется каждый раз при получении символа перевода строки.

Важно

При использовании символа перевода строки в качестве разделителя наборов параметров необходимо гарантировать его отсутствие внутри наборов параметров. Заключение текста в кавычки не делает допустимым перевод строки внутри такого текста. Многострочные JSON-документы не допускаются.

Пример

Потоковая обработка нескольких наборов параметров

JSON

Допустим, нам необходимо выполнить запрос трижды, со следующими наборами значений параметров `a` и `b`:

1. `a` = 10, `b` = 20
2. `a` = 15, `b` = 25
3. `a` = 35, `b` = 48

Создадим файл, который будет содержать строки с JSON-представлением этих наборов:

```
echo -e '{"a":10,"b":20}\n{"a":15,"b":25}\n{"a":35,"b":48}' | tee par1.txt
```

Вывод команды:

```
{"a":10,"b":20}
{"a":15,"b":25}
{"a":35,"b":48}
```

Исполним запрос, передав на `stdin` содержимое данного файла, с форматированием вывода в JSON:

```
cat par1.txt | \
ydb -p quickstart sql \
  -s 'DECLARE $a AS Int64;
      DECLARE $b AS Int64;
      SELECT $a+$b' \
  --input-framing newline-delimited \
  --format json-unicode
```

Вывод команды:

```
{"column0":30}
{"column0":40}
{"column0":83}
```

Или просто передав имя файла в опцию `--input-file`:

```
ydb -p quickstart sql \
  -s 'DECLARE $a AS Int64;
      DECLARE $b AS Int64;
      SELECT $a + $b' \
  --input-file par1.txt \
  --input-framing newline-delimited \
  --format json-unicode
```

Вывод команды:

```
{"column0":30}
{"column0":40}
{"column0":83}
```

Полученный в таком формате результат можно использовать для передачи на вход команде исполнения следующего запроса.

CSV

Допустим, нам необходимо выполнить запрос трижды, со следующими наборами значений параметров `a` и `b`:

1. `a` = 10, `b` = 20
2. `a` = 15, `b` = 25
3. `a` = 35, `b` = 48

Создадим файл с наборами значений параметров в формате CSV:

```
echo -e 'a,b\n10,20\n15,25\n35,48' | tee par1.txt
```

Вывод команды:

```
a,b
10,20
15,25
35,48
```

Исполним запрос, передав на `stdin` содержимое данного файла, с форматированием вывода в CSV:

```
cat par1.txt | \
ydb -p quickstart sql \
  -s 'DECLARE $a AS Int64;
      DECLARE $b AS Int64;
      SELECT $a + $b' \
```

```
  --input-format csv \
  --input-framing newline-delimited \
  --format csv
```

Вывод команды:

```
30
40
83
```

Или просто передав имя файла в опцию `--input-file`:

```
ydb -p quickstart sql \
  -s 'DECLARE $a AS Int64;
      DECLARE $b AS Int64;
      SELECT $a + $b' \
  --input-file par1.txt \
  --input-format csv \
  --input-framing newline-delimited \
  --format csv
```

Вывод команды:

```
30
40
83
```

Полученный в таком формате результат можно использовать для передачи на вход команде исполнения следующего запроса, задав header данным в CSV формате опцией `--input-columns`.

TSV

Допустим, нам необходимо выполнить запрос трижды, со следующими наборами значений параметров `a` и `b`:

1. `a` = 10, `b` = row1
2. `a` = 15, `b` = row 2
3. `a` = 35, `b` = "row"\n3

Создадим файл с наборами значений параметров в формате TSV:

```
echo -e 'a\tb\n10\t20\n15\t25\n35\t48' | tee par1.txt
```

Вывод команды:

```
a  b
10 20
15 25
35 48
```

Исполним запрос, передав на `stdin` содержимое данного файла, с форматированием вывода в TSV:

```
cat par1.txt | \
ydb -p quickstart sql \
  -s 'DECLARE $a AS Int64;
      DECLARE $b AS Utf8;
      SELECT $a, $b' \
  --input-framing newline-delimited \
  --input-format tsv \
  --format tsv
```

Вывод команды:

```
30
40
83
```

Или просто передав имя файла в опцию `--input-file`:

```
ydb -p quickstart sql \
  -s 'DECLARE $a AS Int64;
      DECLARE $b AS Int64;
      SELECT $a + $b' \
  --input-file par1.txt \
  --input-format tsv \
  --input-framing newline-delimited \
  --format tsv
```

Вывод команды:

```
30
40
```

```
83
```

Полученный в таком формате результат можно использовать для передачи на вход команде исполнения следующего запроса, задав header данным в TSV формате опцией `--input-columns`.

Потоковая обработка с объединением значений параметров из разных источников

Допустим, нам необходимо выполнить запрос трижды, со следующими наборами значений параметров `a` и `b`:

1. `a` = 10, `b` = 100
2. `a` = 15, `b` = 100
3. `a` = 35, `b` = 100

```
echo -e '10\n15\n35' | \
ydb -p quickstart sql \
  -s 'DECLARE $a AS Int64;
      DECLARE $b AS Int64;
      SELECT $a + $b AS sum1' \
  --param '$b=100' \
  --input-framing newline-delimited \
  --input-param-name a \
  --format json-unicode
```

Вывод команды:

```
{"sum1":110}
{"sum1":115}
{"sum1":135}
```

Пакетная потоковая обработка

YDB CLI поддерживает автоматическую конвертацию наборов параметров в `List<...>`, позволяя одним запросом к серверу обработать множество наборов параметров в одной транзакции, что может дополнительно существенно повышать производительность по сравнению с выполнением запросов "по одному".

Поддерживаются два режима пакетирования:

- Полный (`full`);
- Адаптивный (`adaptive`).

Полный режим пакетирования

Полный (`full`) режим является упрощенным вариантом пакетирования, когда запрос выполняется один раз, с заворачиванием в `List<...>` всех полученных с входного потока наборов параметров. Если размер запроса окажется слишком большим, будет выдана ошибка.

Данный вариант пакетирования необходим в случае, когда необходимо гарантировать атомарность за счет применения всех параметров в одной транзакции.

Адаптивный режим пакетирования

При работе в адаптивном (`adaptive`) режиме обработка входного потока разбивается на множество транзакций, с автоматическим подбором размера пакета для каждой из них.

Данный режим позволяет эффективно обрабатывать широкий спектр входной нагрузки, с непредсказуемым или бесконечным количеством данных, а также непредсказуемым или сильно меняющимся темпом их появления на входе. В частности, такой профиль характерен при подаче на `stdin` выхода другой команды через оператор `|`.

Адаптивный режим решает две основные проблемы обработки динамического потока:

1. Ограничение максимального размера пакета.
2. Ограничение максимальной задержки обработки данных.

Синтаксис

Для того чтобы воспользоваться возможностью пакетирования, необходимо описать параметр типа `List<...>` или `List<Struct<...>>` в секции DECLARE запроса, и выбрать режим следующим параметром:

Имя	Описание
`--input-batch`	Режим пакетирования значений наборов параметров, получаемых из входного потока (`stdin` или `--input-file`). Возможные значения: • `iterative` (по умолчанию): Пакетирование выключено. Запрос выполняется для каждого набора параметров (ровно один раз, если в опции `--input-framing` используется `no-framing`) • `full`: Полный пакет. Запрос выполнится один раз после завершения чтения входного потока, все полученные наборы параметров заворачиваются в `List<...>`, имя параметра задается опцией `--input-param-name` • `adaptive`: Адаптивное пакетирование. Запрос выполняется каждый раз, когда срабатывает ограничение на количество наборов параметров в одном запросе (`--input-batch-max-rows`) или на задержку обработки (`--input-batch-max-delay`). Все полученные к этому моменту наборы параметров заворачиваются в `List<...>`, имя параметра задается опцией `--input-param-name`.

В адаптивном режиме пакетирования доступны дополнительные параметры:

Имя	Описание

`--input-batch-max-rows`	Максимальное количество наборов параметров в пакете для адаптивного режима пакетирования. Следующий пакет будет отправлен на исполнение вместе с запросом, если количество наборов данных в нем достигло указанного значения. Установка в `0` снимает ограничение. Значение по умолчанию — `1000`. Параметры передаются в запрос без стриминга и общий объем одного GRPC-запроса, в который включаются значения параметров, имеет верхнюю границу около 5 МБ.
`--input-batch-max-delay`	Максимальная задержка отправки на обработку полученного набора параметров для адаптивного режима пакетирования. Задается в виде числа с размерностью времени - `s` (секунды), `ms` (миллисекунды), `m` (минуты) и т.д. Значение по умолчанию — `1s` (1 секунда). YDB CLI будет отсчитывать время с момента получения первого набора параметров для пакета и отправит накопившийся пакет на исполнение, как только время превысит указанное значение. Параметр позволяет получить эффективное пакетирование в случае непредсказуемого темпа появления новых наборов параметров на `stdin`.

Примеры - полная пакетная обработка

```
echo -e '{"a":10,"b":20}\n{"a":15,"b":25}\n{"a":35,"b":48}' | \
ydb -p quickstart sql \
  -s 'DECLARE $x AS List<Struct<a:Int64,b:Int64>>;
      SELECT ListLength($x), $x' \
  --input-framing newline-delimited \
  --input-param-name x \
  --input-batch full
```

Вывод команды:

```
┌─────────┬───────────────────────────────────────────────────┐
| column0 | column1                                           |
├─────────┼───────────────────────────────────────────────────┤
| 3       | [{"a":10,"b":20},{"a":15,"b":25},{"a":35,"b":48}] |
└─────────┴───────────────────────────────────────────────────┘
```

Примеры - адаптивная пакетная обработка

Ограничение максимальной задержки обработки

Для демонстрации работы адаптивного пакетирования со срабатыванием ограничения по задержке обработки в первой строке команды ниже производится генерация 1000 строк с задержкой в 0.2 секунды в `stdout`, которые передаются на `stdin` команде исполнения запроса. Команда исполнения запроса, в свою очередь, отображает пакеты параметров в каждом следующем вызове запроса.

```
for i in $(seq 1 1000); do echo "Line$i"; sleep 0.2; done | \
ydb -p quickstart sql \
  -s 'DECLARE $x AS List<Utf8>;
      SELECT ListLength($x), $x' \
  --input-framing newline-delimited \
  --input-format raw \
  --input-param-name x \
  --input-batch adaptive
```

Вывод команды (точные значения могут отличаться):

```
┌─────────┬─────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┐
| column0 | column1                                                                                                                     |
├─────────┼─────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┤
| 14      | ["Line1","Line2","Line3","Line4","Line5","Line6","Line7","Line8","Line9","Line10","Line11","Line12","Line13","Line14"] |
└─────────┴─────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┘
┌─────────┬───────────────────────────────────────────────────────┐
| column0 | column1                                               |
├─────────┼───────────────────────────────────────────────────────┤
| 6       | ["Line15","Line16","Line17","Line18","Line19","Line20"] |
└─────────┴───────────────────────────────────────────────────────┘
┌─────────┬───────────────────────────────────────────────────────┐
| column0 | column1                                               |
├─────────┼───────────────────────────────────────────────────────┤
| 6       | ["Line21","Line22","Line23","Line24","Line25","Line26"] |
└─────────┴───────────────────────────────────────────────────────┘
^C
```

В первый пакет попадают все строки, накопившиеся на входе за время открытия соединения с БД, и поэтому он больше чем последующие.

Выполнение команды можно прервать по Ctrl+C, или дождаться пока пройдут 200 секунд, в течение которых генерируется вход.

Ограничение по количеству записей

Для демонстрации работы адаптивного пакетирования со срабатыванием триггера по количеству наборов параметров в первой строке команды ниже производится генерация 200 строк. Команда будет отображать пакеты параметров в каждом

следующем вызове запроса, учитывая заданное ограничение `--input-batch-max-rows` равное 20 (по умолчанию 1000).

В данном примере также показана возможность объединения параметров из разных источников, и формирование JSON на выходе.

```
for i in $(seq 1 200); do echo "Line$i"; done | \
ydb -p quickstart sql \
  -s 'DECLARE $x AS List<Utf8>;
      DECLARE $p2 AS Int64;
      SELECT ListLength($x) AS count, $p2 AS p2, $x AS items' \
  --input-framing newline-delimited \
  --input-format raw \
  --input-param-name x \
  --input-batch adaptive \
  --input-batch-max-rows 20 \
  --param '$p2=10' \
  --format json-unicode
```

Вывод команды:

```
{"count":20,"p2":10,"items":["Line1","Line2","Line3","Line4","Line5","Line6","Line7","Line8","Line9","Line1
0","Line11","Line12","Line13","Line14","Line15","Line16","Line17","Line18","Line19","Line20"]}
{"count":20,"p2":10,"items":["Line21","Line22","Line23","Line24","Line25","Line26","Line27","Line28","Line2
9","Line30","Line31","Line32","Line33","Line34","Line35","Line36","Line37","Line38","Line39","Line40"]}
...
{"count":20,"p2":10,"items":["Line161","Line162","Line163","Line164","Line165","Line166","Line167","Line16
8","Line169","Line170","Line171","Line172","Line173","Line174","Line175","Line176","Line177","Line178","Line1
79","Line180"]}
{"count":20,"p2":10,"items":["Line181","Line182","Line183","Line184","Line185","Line186","Line187","Line18
8","Line189","Line190","Line191","Line192","Line193","Line194","Line195","Line196","Line197","Line198","Line1
99","Line200"]}
```

Удаление множества записей из строковой таблицы YDB по первичным ключам

Если вы попытаетесь удалить большое количество строк из крупной таблицы с помощью простого запроса `DELETE FROM large_table WHERE id > 10;`, вы можете столкнуться с ошибкой из-за превышения ограничения на количество записей в транзакции. Данный пример показывает, как можно удалять неограниченное количество записей из таблиц YDB без нарушения этого ограничения.
Создадим тестовую строковую таблицу:

```
ydb -p quickstart sql -s 'CREATE TABLE test_delete_1(id UInt64 NOT NULL, PRIMARY KEY (id))'
```

Занесем в неё 100,000 записей:

```
for i in $(seq 1 100000); do echo "$i";done | \
ydb -p quickstart import file csv -p test_delete_1
```

Удалим все записи со значениями `id` больше 10:

```
ydb -p quickstart sql \
  -s 'SELECT t.id FROM test_delete_1 AS t WHERE t.id > 10' \
  --format json-unicode | \
ydb -p quickstart sql \
  -s 'DECLARE $lines AS List<Struct<id:UInt64>>;
      DELETE FROM test_delete_1 WHERE id IN (SELECT tl.id FROM AS_TABLE($lines) AS tl)' \
  --input-framing newline-delimited \
  --input-param-name lines \
  --input-batch adaptive \
  --input-batch-max-rows 10000
```

Обработка сообщений, считываемых из топика

Примеры обработки сообщений, считываемых из топика, приведены в статье Исполнение YQL-запроса с передачей сообщений из топика в качестве параметров.

См. также

- Параметризованные запросы и повторная компиляция
- Параметризованные запросы в YDB SDK

Интерактивный режим выполнения запросов

Общее описание

После выполнения команды `ydb` без подкоманд запускается интерактивный режим выполнения запросов. После этого можно вводить запросы напрямую в консоль или терминал. При вводе символа перевода строки запрос считается законченным, и он начинает исполняться. Текст запроса может представлять из себя как YQL запрос, так и специальную команду.

Общий вид команды:

```
ydb [global options...]
```

- `global options` — глобальные параметры.

> **Примечание**
>
> Обратите внимание, что для работы команды должны быть заданы параметры подключения. Это может быть профиль по умолчанию, явно указанный профиль и/или набор параметров подключения.

Пример использования:



Интерактивный режим выполнения запросов в YDB CLI предоставляет следующие возможности:

- Подсветка синтаксиса
- Горячие клавиши
- История запросов
- Автодополнение
- Специальные команды
- Режим AI-ассистента

Подсветка синтаксиса



Интерактивный режим поддерживает цветовую подсветку синтаксиса YQL, которая помогает лучше воспринимать структуру запросов. Разными цветами выделяются следующие группы элементов:

- Ключевые слова YQL (SELECT, FROM, WHERE, INSERT, UPDATE и другие)
- Имена таблиц и столбцов
- Строковые литералы (текст в кавычках)
- Числовые литералы
- Операторы (=, <, >, +, - и другие)
- Специальные символы (скобки, запятые, точки)
- Комментарии

Горячие клавиши

Вы можете использовать эти горячие клавиши при работе в интерактивном режиме:

Горячая клавиша	Описание
`Up arrow`	Показывает предыдущий запрос из истории.
`Down arrow`	Показывает следующий запрос из истории.
`TAB`	Дополняет текущее слово на основе синтаксиса YQL.

`CTRL + R`	Поиск запроса в истории по введенной подстроке.
`CTRL + T`	Переключает между режимами YQL и AI.
`CTRL + D`	Выход из интерактивного режима.

История запросов

Клавиши со стрелками вверх и вниз позволяют перемещаться по истории запросов:



История сохраняется локально и доступна между запусками CLI.

Также поддерживается функция поиска по запросам (`CTRL + R`):



Автодополнение

Автодополнение помогает эффективнее писать запросы. Во время ввода предлагаются возможные варианты завершения текущего слова на основе синтаксиса YQL.

Также выполняется поиск имен объектов схемы в базе данных, где это возможно.

Существует два типа подсказок: автодополнение по нажатию клавиши `TAB` и интерактивные подсказки.

Автодополнение по нажатию клавиши TAB

В интерактивном режиме при нажатии клавиши `TAB` отображается список вариантов завершения текущего слова в соответствии с синтаксисом YQL.



Продолжайте набирать текст, уменьшая количество подходящих вариантов.

Если доступен только один вариант, нажатие `TAB` автоматически дополнит текущее слово до него.

Если у всех доступных вариантов общий префикс, нажатие `TAB` автоматически его подставит.

Интерактивные подсказки

В процессе ввода в интерактивном режиме под курсором появляется список подсказок, показывающий первые 4 варианта завершения текущего слова согласно грамматике YQL.



Эта функция предоставляет быстрые подсказки без перегрузки всеми возможными вариантами, помогая придерживаться правильного синтаксиса при написании запросов.

Специальные команды

Специальные команды специфичны для CLI и не являются частью синтаксиса YQL. Они предназначены для выполнения различных функций, которые нельзя выполнить через YQL запрос.

Команда	Описание
`SET param = value`	Устанавливает значение внутренней переменной `param` в `value`.
`EXPLAIN query-text`	Выводит план запроса `query-text`. Эквивалентна команде ydb sql --explain.
`EXPLAIN AST query-text`	Выводит план запроса `query-text` вместе с AST. Эквивалентна команде ydb sql --explain-ast.

Список внутренних переменных

Внутренние переменные устанавливают поведение команд и задаются с помощью специальной команды `SET`.

Переменная	Описание
`stats`	Режим сбора статистики для последующих запросов. Возможные значения: • `none` (по умолчанию) — не собирать; • `basic` — собирать статистику; • `full` — собирать статистику и план запроса.
`resource_pool`	Определяет ресурсный пул для выполнения последующих запросов. В качестве значения ожидает наименование пула.

Примеры

Выполнение запроса в режиме сбора статистики `full`:

```
$ ydb
ydb> SET stats = full
ydb> select * from table1 limit 1
┌────┬─────┬─────────┐
| id | key | value |
├────┼─────┼─────────┤
| 10 | 0   | ""    |
└────┴─────┴─────────┘

Statistics:
query_phases {
  duration_us: 14987
  table_access {
    name: "/ru-central1/a1v7bqj3vtf10qjleyow/laebarufb61tguph3g22/table1"
    reads {
      rows: 9937
      bytes: 248426
    }
  }
  cpu_time_us: 2925
  affected_shards: 1
}
process_cpu_time_us: 3816
total_duration_us: 79530
total_cpu_time_us: 6741


Full statistics:
Query 0:
ResultSet
└──Limit (Limit: 1)
   TotalCpuTimeUs: 175
   TotalTasks: 1
   TotalInputBytes: 6
   TotalInputRows: 1
   TotalOutputBytes: 16
   TotalDurationMs: 0
   TotalOutputRows: 1
   └──<UnionAll>
      └──Limit (Limit: 1)
      └──TableFullScan (ReadColumns: ["id","key","value"], ReadRanges: ["key (-∞, +∞)"], Table: impex_table)
         Tables: ["table1"]
         TotalCpuTimeUs: 154
         TotalTasks: 1
         TotalInputBytes: 0
         TotalInputRows: 0
         TotalOutputBytes: 16
         TotalDurationMs: 0
         TotalOutputRows: 1
```

Выполнение запроса в ресурсном пуле `default`:

```
$ ydb
ydb> SET resource_pool = default
Resource pool set to "default".

ydb> select 1;
┌─────────┐
| column0 |
├─────────┤
| 1       |
└─────────┘
```

Режим AI-ассистента

Интерактивный режим включает режим AI-ассистента — диалогового агента на основе большой языковой модели, который помогает исследовать структуру базы данных, писать и выполнять запросы YQL, искать информацию в документации и вызывать команды YDB CLI. Переключаться между режимами YQL и AI можно горячей клавишей `Ctrl + T` или командой `/switch`.

Подробнее см. Режим AI-ассистента.

Режим AI-ассистента

Интерактивный режим выполнения запросов YDB CLI включает режим AI-ассистента — диалогового агента на основе большой языковой модели (LLM). Ассистент понимает запросы на естественном языке и помогает работать с базой данных: исследовать её структуру, писать и выполнять запросы YQL, объяснять планы запросов, искать информацию в документации и вызывать команды YDB CLI.

> **Важно**
>
> Ассистент обращается к внешнему API большой языковой модели, которое вы указываете самостоятельно. Содержимое диалога, включая структуру схемы и результаты запросов, которые ассистент решит передать модели, отправляется этому API. Не используйте режим с конфиденциальными данными, если это запрещено вашими политиками. YDB CLI не предоставляет языковую модель и ключ доступа к ней — их необходимо получить у выбранного вами провайдера.

Переключение в AI-режим

Интерактивный режим работает в одном из двух подрежимов:

- **YQL** — ввод и выполнение запросов YQL (используется по умолчанию при первом запуске);
- **AI** — диалог с AI-ассистентом.

Запустите интерактивный режим, выполнив команду `ydb` без подкоманды с необходимыми параметрами подключения:

```
ydb [global options...]
```

Активный подрежим отображается в приглашении командной строки (`YQL>` или `AI>`). Переключаться между подрежимами можно горячей клавишей `Ctrl + T` или командой `/switch`. YDB CLI запоминает последний использованный подрежим и открывает его при следующем запуске.

Настройка AI-профиля

Для работы ассистента нужен AI-профиль — набор параметров подключения к API языковой модели. При первом переходе в AI-режим, если профилей ещё нет, YDB CLI предложит создать профиль и проведёт через настройку шаг за шагом:

Параметр	Описание
API endpoint	URL API языковой модели. Должен начинаться с `http://` или `https://` (например, `https://api.openai.com/v1/`). YDB CLI проверяет соединение с указанным адресом.
API type	Тип API: • **OpenAI-compatible** — для моделей с OpenAI-совместимым API; • **Anthropic** — для API Anthropic.
API token	Токен доступа к API. Сохраняется в файле конфигурации. Поле можно оставить пустым и передавать токен через переменную окружения `YDB_CLI_AI_TOKEN`.
Model name	Имя модели. Если API позволяет, YDB CLI запросит список доступных моделей; имя также можно ввести вручную. Пустое значение означает, что имя модели не передаётся в запросах.
Profile display name	Отображаемое имя профиля, под которым он сохраняется и показывается при выборе.

После настройки профиль сохраняется в файле конфигурации и становится активным. Сменить активный профиль или создать новый можно командой `/model`.

Работа с ассистентом

В AI-режиме введите сообщение на естественном языке и нажмите `Enter` для отправки (`Ctrl + Enter` — перенос строки). Ассистент анализирует запрос и при необходимости вызывает инструменты для работы с базой данных:

Возможность	Изменяет данные	Подтверждение
Просмотр списка объектов схемы по указанному пути	Нет	Не требуется
Описание объекта схемы (таблицы, топика и т. д.)	Нет	Не требуется
Получение плана запроса YQL (`EXPLAIN`)	Нет	Не требуется
Поиск информации в документации YDB	Нет	Не требуется
Получение справки по командам YDB CLI	Нет	Не требуется
Выполнение запроса YQL	Да	Требуется
Выполнение команды оболочки	Да	Требуется

Ассистент сохраняет контекст диалога: последующие сообщения в той же сессии учитывают предыдущие. Очистить контекст можно командой `/config`.

Подтверждение действий

Перед выполнением действия, которое может изменить данные, — выполнения запроса YQL или команды оболочки — ассистент запрашивает подтверждение и предлагает варианты:

- **Approve execution** — выполнить запрос или команду в текущем виде;
- **Edit query** / **Edit command** — отредактировать текст запроса или команды перед выполнением;
- **Abort operation** — отменить выполнение (и продолжить работу в том же контексте).

Команды

В AI-режиме доступны следующие команды:

Команда	Описание
`/help`	Показать справку по AI-режиму и список горячих клавиш.
`/model`	Выбрать другой AI-профиль или создать новый.
`/config`	Изменить настройки сессии: очистить контекст диалога, сменить или отредактировать активный профиль, удалить профиль.
`/switch`	Переключиться в режим YQL (аналог `Ctrl + T`).

Файл конфигурации

AI-профили и настройки AI-режима хранятся в YAML-файле. По умолчанию это `~/.config/ydb/ai_profiles.yaml`; путь можно переопределить переменной окружения `YDB_CLI_AI_PROFILE_FILE`. Файл содержит список профилей, идентификатор активного профиля и текущий подрежим интерактивного режима. Обычно его не требуется редактировать вручную — управляйте профилями через команды `/model` и `/config`.

> **Примечание**
>
> Токен, введённый при настройке профиля напрямую, сохраняется в файле конфигурации. YDB CLI ограничивает права доступа к этому файлу так, что читать и изменять его может только пользователь-владелец, запустивший CLI (в Unix-системах — режим доступа `0600`). Если вы предпочитаете не хранить токен на диске, оставьте поле токена пустым и передавайте его через переменную окружения `YDB_CLI_AI_TOKEN`.

Переменные окружения

Переменная	Описание
`YDB_CLI_AI_TOKEN`	Токен доступа к API языковой модели. Используется, если токен не задан в активном профиле.
`YDB_CLI_AI_PROFILE_FILE`	Путь к файлу конфигурации AI-профилей. По умолчанию `~/.config/ydb/ai_profiles.yaml`. Позволяет использовать отдельный набор профилей или нестандартное расположение файла, например в автоматизированных сценариях.
`YDB_CLI_AI_DISABLE_HISTORY`	Если переменная задана, история сообщений AI-режима не сохраняется на диск между запусками. Может использоваться, если переписку с ассистентом не следует хранить на диске.

Выполнение скан запросов

Важно

Использование `ScanQuery` не рекомендуется для новых сценариев. Для выполнения запросов используйте стандартные механизмы.

Подробнее см. в разделе Выполнение запросов.

Исключение — выполнение длительных (более 5 минут) запросов к строчным таблицам. В этом случае мы по-прежнему рекомендуем использовать `ScanQuery`, так как стандартные механизмы выполнения запросов пока не полностью поддерживают этот сценарий.

Запуск запроса через Scan Queries посредством YDB CLI осуществляется добавлением флага `-t scan` в команду `ydb table query execute`.

Выполните запрос к данным:

```
ydb table query execute -t scan \
 --query "SELECT season_id, episode_id, title \
 FROM episodes \
 WHERE series_id = 1 AND season_id > 1 \
 ORDER BY season_id, episode_id \
 LIMIT 3"
```

Где:

- `--query` — текст запроса.

Результат:

```
┌───────────┬────────────┬──────────────────────────────────┐
| season_id | episode_id | title |
├───────────┼────────────┼──────────────────────────────────┤
| 2 | 1 | "The Work Outing" |
├───────────┼────────────┼──────────────────────────────────┤
| 2 | 2 | "Return of the Golden Child" |
├───────────┼────────────┼──────────────────────────────────┤
| 2 | 3 | "Moss and the German" |
└───────────┴────────────┴──────────────────────────────────┘
```

Получение списка фоновых операций

С помощью подкоманды `ydb operation list` вы можете получить список фоновых операций указанного типа.

Общий вид команды:

```
ydb [global options...] operation list [options...] <kind>
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.
- `kind` — тип операции. Возможные значения:
  - `analyze` — операции сбора статистики ANALYZE;
  - `buildindex` — операции построения индекса;
  - `compaction` — операции компакшна таблиц;
  - `export/s3` — операции экспорта в S3;
  - `export/nfs` — операции экспорта в NFS;
  - `import/s3` — операции импорта из S3;
  - `import/nfs` — операции импорта с NFS;
  - `scriptexec` — операции выполнения скриптов;
  - `fullbackup` — операции создания полной резервной копии;
  - `incbackup` — операции создания инкрементальной резервной копии;
  - `restore` — операции восстановления из коллекции резервных копий;
  - `setnotnull` — операции выставления ограничения `NOT NULL`.

Посмотреть описание команды получения списка фоновых операций:

```
ydb operation list --help
```

Параметры подкоманды

Имя	Значение по умолчанию	Описание
`-s`, `--page-size`	`10` (устанавливается на стороне сервера, может меняться в зависимости от типа операции)	Количество операций на одной странице. Если список операций содержит больше строк, чем задано в параметре `--page-size`, то вывод будет разделен на несколько страниц. Для получения следующей страницы укажите параметр `--page-token`.
`-t`, `--page-token`	`0`	Токен страницы.
`--format`	`pretty`	Возможные значения: • `pretty` — человекочитаемый формат вывода; • `proto-json-base64` — вывод Protobuf в формате JSON, бинарные строки закодированы в Base64.

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Получить список фоновых операций построение индекса для таблицы `series`:

```
ydb -p quickstart operation list \
  buildindex
```

Результат:

```
┌──────────────────────────────────────┬───────┬─────────┬───────┬──────────┬──────────────────────┬──────────
──┐
| id                                   | ready | status  | state | progress | table                | index
|
├──────────────────────────────────────┼───────┼─────────┼───────┼──────────┼──────────────────────┼──────────
──┤
| ydb://buildindex/7?id=281489389055514 | true  | SUCCESS | Done  | 100.00%  | /my-database/series | release_
idx |
├--------------------------------------┼-------┼---------┼-------┼----------┼----------------------┼-------
-----┤
| Create time: 2023-04-20T11:04:29Z
|
| End time: 2023-04-20T11:04:29Z
|
├──────────────────────────────────────┼───────┼─────────┼───────┼──────────┼──────────────────────┼──────────
──┤
| ydb://buildindex/7?id=281489389055513 | true  | SUCCESS | Done  | 100.00%  | /my-database/series | title_id
x   |
├--------------------------------------┼-------┼---------┼-------┼----------┼----------------------┼-------
-----┤
| Create time: 2023-04-20T10:58:29Z
```

```
 │
 │ End time: 2023-04-20T10:59:29Z
 │
 └──────────────────────────────────────────────────────────────────────────────────────────────────────────────
 ────┘

 Next page token: 0
```

По умолчанию в выводе `buildindex` показываются только 10 операций, отсортированных в порядке убывания `id`. Для каждой новой операции выбирается большее значение идентификатора, но это не гарантируется. Если `id` оказался меньше существующих, тогда новая операция попадёт на другую страницу.

Получение статуса фоновой операции

С помощью подкоманды `ydb operation get` вы можете получить статус указанной фоновой операции.

Общий вид команды:

```
ydb [global options...] operation get [options...] <id>
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.
- `id` — идентификатор фоновой операции. Идентификатор содержит символы, которые могут быть интерпретированы вашей командной оболочкой. При необходимости используйте экранирование, например `'<id>'` для bash.

Посмотрите описание команды получения статуса фоновой операции:

```
ydb operation get --help
```

Параметры подкоманды

Имя	Описание
`--format`	Формат вывода. Значение по умолчанию — `pretty`. Возможные значения: • `pretty` — человекочитаемый формат; • `proto-json-base64` — вывод Protobuf в формате JSON, бинарные строки закодированы в Base64.

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Получите статус фоновой операции с идентификатором `ydb://buildindex/7?id=281489389055514`:

```
ydb -p quickstart operation get \
  'ydb://buildindex/7?id=281489389055514'
```

Результат:

```
┌──────────────────────────────────────────┬───────┬─────────┬───────┬──────────┬──────────────────────┬──────────
──┐
| id                                       | ready | status  | state | progress | table                | index
|
├──────────────────────────────────────────┼───────┼─────────┼───────┼──────────┼──────────────────────┼──────────
──┤
| ydb://buildindex/7?id=281489389055514 | true  | SUCCESS | Done  | 100.00%  | /my-database/series | idx_rele
ase |
└──────────────────────────────────────────┴───────┴─────────┴───────┴──────────┴──────────────────────┴──────────
──┘
```

Отмена фоновой операции

С помощью подкоманды `ydb operation cancel` вы можете инициировать отмену указанной фоновой операции. Можно отменить только незавершенную операцию.

Общий вид команды:

```
ydb [global options...] operation cancel <id>
```

- `global options` — глобальные параметры.
- `id` — идентификатор фоновой операции. Идентификатор содержит символы, которые могут быть интерпретированы вашей командной оболочкой. При необходимости используйте экранирование, например `'<id>'` для bash.

Посмотрите описание команды получения статуса фоновой операции:

```
ydb operation cancel --help
```

Примеры

Примечание

В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Удаление фоновой операции из списка

С помощью подкоманды `ydb operation forget` вы можете удалить информацию об указанной фоновой операции из списка. Операция должна быть завершена.

Общий вид команды:

```
ydb [global options...] operation forget <id>
```

- `global options` — глобальные параметры.
- `id` — идентификатор фоновой операции. Идентификатор содержит символы, которые могут быть интерпретированы вашей командной оболочкой. При необходимости используйте экранирование, например `'<id>'` для bash.

Посмотрите описание команды удаления информации об указанной фоновой операции:

```
ydb operation forget --help
```

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Удалите из списка фоновую операцию с идентификатором `ydb://buildindex/7?id=281489389055514`:

```
ydb -p quickstart operation forget \
  'ydb://buildindex/7?id=281489389055514'
```

Управление профилями

Профиль — именованный набор параметров соединения с БД, сохраненный в конфигурационном файле на локальной файловой системе. Использование профилей позволяет переиспользовать данные о размещении БД и параметрах аутентификации, сделав вызовы CLI существенно короче:

- Вызов команды `scheme ls` без профиля:

```
ydb \
-e grpsc://some.host.in.some.domain:2136 \
-d /some_long_identifier1/some_long_identifier2/database_name \
--yc-token-file ~/secrets/token_database1 \
scheme ls
```

- Вызов той же команды `scheme ls` с использованием профиля:

```
ydb -p quickstart scheme ls
```

Команды управления профилями

- Создание профиля
- Использование профиля
- Получение списка профилей и параметров профиля
- Удаление профиля
- Активация профиля и применение активированного профиля

Где хранятся профили

Профили хранятся локально в файле `~/.ydb/config/config.yaml`.

Создание и изменение профиля

Значения параметров соединения для создаваемого или изменяемого профиля могут быть заданы в командной строке, или запрошены в интерактивном режиме из консоли.

Командная строка

Для создания или изменения профиля из командной строки применяются команды `profile create`, `profile update` и `profile replace`.

Используются только те значения, которые непосредственно указаны в командной строке, без обращений к переменным окружения или активированному профилю.

Profile create

Команда `profile create` создает новый профиль с заданными значениями параметров:

```
ydb config profile create <profile_name> <connection_options>
```

В данной команде:

- `<profile_name>` -- обязательное имя профиля
- `<connection options>` -- параметры соединения для записи в профиле. Необходимо указание как минимум одного параметра соединения, иначе команда будет выполняться в интерактивном режиме.

Если профиль с указанным именем существует, то выполнение команды завершится с ошибкой.

Profile replace

Команда `profile replace` создает или заменяет профиль с заданными значениями параметров:

```
ydb config profile replace <profile_name> [connection_options]
```

В данной команде:

- `<profile_name>` -- обязательное имя профиля
- `<connection options>` -- необязательные параметры соединения для записи в профиле

Если профиль с указанным именем существует, то он будет заменен на новый, содержащий переданные параметры. Если не указан ни один параметр соединения, после исполнения команды профиль будет пустым.

Profile update

Команда `profile update` изменяет параметры существующего профиля:

```
ydb config profile update <profile_name> [connection_options] [reset-options]
```

В данной команде:

- `<profile_name>` -- обязательное имя профиля
- `<connection options>` -- необязательные параметры соединения для записи в профиле
- `<reset options>` -- необязательные опции удаления параметров из существующего профиля. Возможные значения:

  `--no-endpoint` -- удалить эндпоинт из профиля
  `--no-database` -- удалить путь базы данных из профиля
  `--no-auth` -- удалить аутентификационную информацию из профиля
  `--no-iam-endpoint` -- удалить URL сервера IAM

В профиле будут обновлены те параметры, которые упомянуты в командной строке. Значения не перечисленных в командной строке параметров останутся без изменений.

Примеры

Создание профиля для соединения с тестовой БД

Вы можете использовать профиль `quickstart` для соединения с БД в составе одноузлового кластера YDB:

```
ydb config profile create quickstart --endpoint grpc://localhost:2136 --database <path_database>
```

- `path_database` — путь базы данных. Укажите одно из значений:
  - `/Root/test` — если вы развернули кластер с помощью исполняемого файла;
  - `/local` — если вы использовали Docker-образ.

Создание профиля по ранее использованным параметрам соединения

Любая команда выполнения операции в базе данных YDB с явно заданными параметрами соединения может быть преобразована в команду создания профиля перемещением параметров соединения из глобальных опций в опции команды `config profile create`.

Например, если вы успешно выполнили команду `scheme ls` со следующими реквизитами:

```
ydb \
  -e grpcs://example.com:2135 -d /Root/somedatabase --sa-key-file ~/sa_key.json \
  scheme ls
```

То создать профиль для соединения с использованной базой данных можно следующей командой:

```
ydb \
  config profile create db1 \
  -e grpcs://example.com:2135 -d /Root/somedatabase --sa-key-file ~/sa_key.json
```

Теперь можно записать исходную команду гораздо короче:

```
ydb -p db1 scheme ls
```

Профиль для соединения с локальной базой данных

Создание/замена профиля `local` для соединения с локальной БД YDB, развернутой сценариями быстрого развертывания:

```
ydb config profile replace local --endpoint grpc://localhost:2136 --database /Root/test
```

Определение способа аутентификации по логину и паролю в профиле `local`:

```
ydb config profile update local --user user1 --password-file ~/pwd.txt
```

Интерактивный режим

Профили создаются и изменяются в интерактивном режиме следующими командами:

```
ydb init
```

или

```
ydb config profile create [profile_name] [connection_options]
```

В данной команде:

- `[profile_name]` -- необязательное имя создаваемого или изменяемого профиля
- `[connection_options]` -- необязательные параметры соединения для записи в профиле

Команда `init` всегда работает в интерактивном режиме, а `config profile create` запускается в интерактивном режиме в случае, если не указано имя профиля, или не указан ни один из параметров соединения в командной строке.

Начало интерактивного сценария отличается в командах `init` и `profile create`:

Init

1. Выводится перечень существующих профилей (если они есть), и предлагается выбор - создать новый (Create a new) или изменить конфигурацию одного из существующих:

   ```
   Please choose profile to configure:
   [1] Create a new profile
   [2] test
   [3] local
   ```

2. Если существующих профилей нет, или выбран вариант `1` на предыдущем шаге, то запрашивается имя профиля для создания:

   ```
   Please enter name for a new profile:
   ```

3. Если ввести в этот момент имя существующего профиля, то YDB CLI переходит к шагам изменения его параметров, как если бы сразу была выбрана опция с именем этого профиля.

Profile Create

Если в командной строке не указано имя профиля, то оно запрашивается:

```
Please enter configuration profile name to create or re-configure:
```

Далее будет последовательно предложено выполнить действия с каждым параметром соединения, доступным для сохранения в профиле:

- Не сохранять в профиле - Don't save
- Задать или выбрать значение - Set a new value или Use <вариант>
- Оставить предыдущее значение - Use current value (опция доступна при изменении существующего профиля)

Пример

Создание нового профиля `mydb1`:

1. Выполните команду:

   ```
   ydb config profile create mydb1
   ```

2. Введите эндпоинт или не сохраняйте этот параметр для профиля:

```
Pick desired action to configure endpoint:
 [1] Set a new endpoint value
 [2] Don't save endpoint for profile "mydb1"
Please enter your numeric choice:
```

3. Введите имя базы данных или не сохраняйте этот параметр для профиля:

```
Pick desired action to configure database:
 [1] Set a new database value
 [2] Don't save database for profile "mydb1"
Please enter your numeric choice:
```

4. Выберите режим аутентификации или не сохраняйте этот параметр для профиля:

```
Pick desired action to configure authentication method:
  [1] Use static credentials (user & password)
  [2] Use IAM token (iam-token) cloud.yandex.ru/docs/iam/concepts/authorization/iam-token
  [3] Use OAuth token of a Yandex Passport user (yc-token). Doesn't work with federative accounts. clou
d.yandex.ru/docs/iam/concepts/authorization/oauth-token
  [4] Use metadata service on a virtual machine (use-metadata-credentials) cloud.yandex.ru/docs/compute/
operations/vm-connect/auth-inside-vm
  [5] Use service account key file (sa-key-file) cloud.yandex.ru/docs/iam/operations/iam-token/create-fo
r-sa
  [6] Set new access token (ydb-token)
  [7] Don't save authentication data for profile "mydb1"
Please enter your numeric choice:
```

   Все доступные методы аутентификации описаны в статье Аутентификация. Набор методов и текст подсказок может отличаться от приведенного в данном примере.

   Если выбранный вами метод подразумевает указание дополнительного параметра, вам будет предложено его ввести. Например, если вы выбрали `4` (Use service account key file):

```
Please enter Path to service account key file (sa-key-file):
```

5. На последнем шаге вам будет предложено активировать созданный профиль для использования по умолчанию. Ответьте 'n' (Нет), пока вы не прочитали статью про Активацию профиля и применение активированного профиля:

```
Activate profile "mydb1" to use by default? (current active profile is not set) y/n: n
```

Использование профиля

Соединение по выбранному профилю

Профиль может быть применен при запуске команды YDB CLI указанием опции `--profile` или `-p`:

```
ydb -p <profile_name> <команда и опции команды>
```

Например:

```
ydb -p mydb1 scheme ls -l
```

В таком случае все параметры соединения с БД будут взяты из профиля. При этом, если в профиле не указано параметров аутентификации, то YDB CLI попробует их определить по переменным окружения, как описано в статье Соединение с БД и аутентификация — Переменные окружения.

Соединение по выбранному профилю и параметрам командной строки

Опция `--profile` (`-p`) может быть не единственной среди параметров соединения в командной строке, например:

```
ydb -p mydb1 -d /local2 scheme ls -l
```

```
ydb -p mydb1 --user alex scheme ls -l
```

В таком случае указанные в командной строке параметры соединения имеют приоритет перед сохраненными в профиле. Такой формат позволяет переиспользовать профили для соединения с разными БД или под разными учетными записями. Также, указание в командной строке параметра аутентификации (как `--user alex` в примере выше) отключает проверку переменных окружения независимо от их наличия в профиле.

Соединение по активированному профилю

Если в командной строке не указана опция `--profile` (`-p`), то YDB CLI попробует взять из текущего активированного профиля все параметры соединения, которые он не смог определить другими способами (из опций командной строки или переменных окружения, как описано в статье Соединение с БД и аутентификация).

Неявное применение активированного профиля может приводить к ошибкам, поэтому перед использованием данного режима рекомендуется изучить статью Активированный профиль.

Получение информации о профиле

Получение списка профилей

Получение списка профилей:

```
ydb config profile list
```

Если существует текущий активированный профиль, он будет помечен как `(active)` в выведенном списке, например:

```
prod
test (active)
local
```

Получение подробной информации о профиле

Получение параметров, сохраненных в заданном профиле:

```
ydb config profile get <profile_name>
```

Например:

```
$ ydb config profile get local1
  endpoint: grpcs://ydb.serverless.yandexcloud.net:2135
  database: /rul1/b1g8skp/etn02099
  sa-key-file: /Users/username/secrets/sa_key_test.json
```

Получение профилей с содержимым

Полная информация по всем профилям и сохраненным в них параметрам:

```
ydb config profile list --with-content
```

Вывод данной команды объединяет вывод команды получения списка (с пометкой активного профиля) и параметров каждого профиля в следующих строках после его имени.

Удаление профиля

Удаление профиля производится следующей командой:

```
ydb config profile delete <profile_name> [-f]
```

Параметры

`<profile_name>` : Имя профиля.

`--force` (`-f`) : Флаг усиленного режима. Если не указан, то команда будет запрашивать подтверждение удаления в интерактивном режиме, и вернет ошибку в случае если профиль отсутствует. Если указан, то запрос на подтверждение не выдается, и команда завершается успешно даже если профиля с указанным именем не существовало.

Примеры

Удаление профиля mydb1 без интерактивного подтверждения

```
ydb config profile delete mydb1 -f
```

Команда исполнится успешно также и в случае, если профиль с именем `mydb1` не существует.

Удаление профиля mydb1 с интерактивным подтверждением

```
ydb config profile delete mydb1
```

Если профиль не существует, выполнение команды завершится с ошибкой:

```
No existing profile "mydb1". Run "ydb config profile list" without arguments to see existing profiles
```

Если профиль существует, то выдается интерактивный запрос на подтверждение:

```
Profile "mydb1" will be permanently removed. Continue? (y/n): y
Profile "mydb1" was removed.
```

Удаление всех профилей

Хотя специальной команды удаления всех профилей нет в YDB CLI, в Linux-совместимых операционных системах вы можете воспользоваться следующим рецептом для удаления всех существующих профилей:

```
ydb config profile list | while read l; do ydb config profile delete $l; done;
```

Активированный профиль

Исполнение команд YDB CLI над базой данных требует установки соединения с ней. Если YDB CLI не удалось определить какой-либо параметр соединения по параметрам командной строки и переменным окружения, он берется из активированного профиля.

Активация профиля является простым способом начать изучение YDB CLI, так как один раз заданные параметры соединения будут применяться автоматически к любой команде, без указания каких-либо параметров соединения в командной строке.

Однако, данная простота может приводить к нежелательному поведению в дальнейшей работе, как только возникнет необходимость работы с несколькими базами данных:

- Активированный профиль применяется неявно, а значит, может примениться по ошибке, когда забыт какой-либо параметр соединения в командной строке.
- Активированный профиль применяется неявно, а значит, может примениться по ошибке, когда допущена описка в имени переменной окружения.
- Активированный профиль нельзя применять в скриптах, так как он сохраняется в файле, и его смена в одном окне терминала повлияет на все другие окна, возможно приведя к неожиданной смене БД посередине исполняемого в скрипте цикла.

С момента как вам в первый раз потребуется соединиться с какой-либо новой БД, отличной от начальной, рекомендуется деактивировать профиль и всегда выбирать его явно опцией `--profile` (`-p`).

Активация профиля

Активация профиля выполняется командой

```
ydb config profile activate [profile_name]
```

, где `[profile_name]` - необязательное имя профиля.

Если имя профиля указано, то производится его активация. Если профиля с указанным именем не существует, то будет выдана ошибка с предложением посмотреть список доступных профилей:

```
No existing profile "<profile_name>". Run "ydb config profile list" without arguments to see existing profile
s
```

Если имя профиля не указано, то в интерактивном режиме будет предложено выбрать между следующими вариантами:

```
Please choose profile to activate:
 [1] Don't do anything, just exit
 [2] Deactivate current active profile (if any)
 [3] <profile_name_1> (active)
 [4] <profile_name_2>
 ...
Please enter your numeric choice:
```

- `1` завершает исполнение, активированным останется текущий активированный профиль. Он отмечен как `(active)` в списке существующих профилей, начинающемся с пункта 3.
- `2` деактивирует текущий активированный профиль. Если никакой профиль ранее не был активирован, то ничего не изменится.
- `3` и далее — активация выбранного профиля. Текущий активированный профиль отмечен как `(active)`

При успешной активации профиля исполнение завершается сообщением

```
Profile "<profile_name>" was activated.
```

Пример

Активация профиля с именем `mydb1`:

```
ydb config profile activate mydb1
Profile "mydb1" was activated.
```

Активация профиля при его инициализации

На последнем шаге интерактивного выполнения команды создания или изменения профиля `ydb config profile create` предлагается активировать созданный (или измененный) профиль:

```
Activate profile "<profile_name>" to use by default? (current active profile is not set) y/n:
```

Ответьте `n` (Нет) чтобы не активировать создаваемый профиль, или `y` (Да) чтобы он был активирован.

Деактивация профиля

Деактивация текущего активированного профиля выполняется командой

```
ydb config profile deactivate
```

Если активированный профиль существовал, то выводится сообщение о его успешной деактивации:

```
Profile "profile_name" was deactivated.
```

Если активированного профиля не существовало, то выводится сообщение о том что в результате исполнения команды никаких изменений не произведено:

```
There is no profile active. Nothing is done.
```

В любом случае, команда возвращает result code `0`.

Список эндпоинтов

Информационная команда `discovery list` позволяет получить список эндпоинтов кластера YDB, с которыми можно открывать соединения для работы с заданной базой данных:

```
ydb [connection options] discovery list
```

, где `[connection options]` — опции соединения с БД

В выводимых в ответ строках содержится следующая информация:

1. Эндпоинт, включая протокол и порт
2. В квадратных скобках зона доступности
3. С символом `#` перечень сервисов YDB, доступных на данном эндпоинте

Запрос к кластеру YDB на endpoint discovery выполняется в YDB SDK при инициализации драйвера, и команду CLI `discovery list` можно использовать для локализации возможных проблем соединения.

Пример

```
$ ydb -p quickstart discovery list
grpcs://vm-etn01q5-ysor.etn01q5k.ydb.mdb.yandexcloud.net:2135 [sas] #table_service #scripting #discovery #rat
e_limiter #locking #kesus
grpcs://vm-etn01q5-arum.etn01ftr.ydb.mdb.yandexcloud.net:2135 [vla] #table_service #scripting #discovery #rat
e_limiter #locking #kesus
grpcs://vm-etn01q5beftr.ydb.mdb.yandexcloud.net:2135 [myt] #table_service #scripting #discovery #rate_limiter
#locking #kesus
```

Проверка аутентификации

Информационная команда `discovery whoami` позволяет проверить, от имени какой учетной записи сервер фактически принимает запросы:

```
ydb [connection options] discovery whoami [-g|--groups] [-l|--access-list] [-a|--all]
```

, где `[connection options]` — опции соединения с БД

В ответ выводится имя учетной записи (User SID). Дополнительную информацию можно запросить с помощью следующих опций:

- `-g`, `--groups` — показать группы, в которые входит учетная запись;
- `-l`, `--access-list` — показать уровни доступа, выданные учетной записи (administration, monitoring, viewer, database, register node, bootstrap);
- `-a`, `--all` — показать всю дополнительную информацию (эквивалентно одновременному указанию `-g` и `-l`).

Уровни доступа, отображаемые при использовании опций `-l` или `-a`, соответствуют иерархической конфигурации управления доступом. Подробная информация о списках уровней доступа и их иерархии приведена в разделе Списки уровней доступа документации по авторизации.

Выводятся только те уровни доступа, которые действительно выданы пользователю:

- **Database** (наличие в `database_allowed_sids`) — даёт право работать с YDB UI только как «пользователь базы данных»: можно открыть UI и видеть данные в рамках конкретной базы, но нельзя просматривать общекластерные данные или выполнять операции уровня кластера.
- **Viewer** (наличие в `viewer_allowed_sids`) — даёт право просматривать YDB UI без возможности вносить изменения.
- **Monitoring** (наличие в `monitoring_allowed_sids`) — даёт право выполнять в YDB UI действия, изменяющие состояние системы.
- **Administration** (наличие в `administration_allowed_sids`) — даёт право выполнять административные действия с базами данных или кластером.
- **Register node** (наличие в `register_dynamic_node_allowed_sids`) — даёт право регистрировать динамические узлы в кластере.
- **Bootstrap** (наличие в `bootstrap_allowed_sids`) — даёт право выполнять операции начальной инициализации кластера.

Если на сервере YDB не включена аутентификация (что может применяться, например, при самостоятельном локальном развертывании), выполнение команды завершится ошибкой.

Поддержка опции `-g` зависит от конфигурации сервера. Если она отключена, в ответ всегда будет возвращаться `User has no groups`, независимо от фактического членства вашей учетной записи в каких-либо группах.

Примеры

Базовое использование

```
$ ydb -p quickstart discovery whoami
User SID: aje5kkjdgs0puc18976co@as
```

С выводом групп

```
$ ydb -p quickstart discovery whoami -g
User SID: aje5kkjdgs0puc18976co@as

User has no groups
```

С выводом списка уровней доступа

```
$ ydb -p quickstart discovery whoami -l
User SID: user1@builtin

Access levels:
Database
Viewer
```

С выводом всей информации

```
$ ydb -p quickstart discovery whoami -a
User SID: admin@builtin

Group SIDs:
all-users@well-known
ADMINS

Access levels:
Database
Viewer
Monitoring
Administration
```

Получение токена аутентификации

С помощью подкоманды `auth get-token` вы можете получить токен аутентификации на основе параметров аутентификации, указанных в профиле, переменных окружения или параметрах командной строки.

Общий вид команды:

```
ydb [global options...] auth get-token [options...]
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.

Посмотрите описание команды получения токена:

```
ydb auth get-token --help
```

Параметры подкоманды

Параметр	Описание
`-f, --force`	Вывести токен без запроса подтверждения.
`--timeout`	Время ожидания ответа клиента в миллисекундах. После истечения этого времени нет смысла ждать результат.

Примеры

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Получение токена с подтверждением

По умолчанию команда запрашивает подтверждение перед выводом токена, так как токен будет выведен в консоль:

```
ydb -p quickstart auth get-token
```

Результат:

```
Caution: Your auth token will be printed to console. Use "--force" ("-f") option to print without prompting.
Do you want to proceed? (y/N): y
t1.eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...
```

Получение токена без подтверждения

Для автоматизации или использования в скриптах используйте опцию `--force` для вывода токена без запроса подтверждения:

```
ydb -p quickstart auth get-token --force
```

Результат:

```
t1.eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...
```

Использование в скриптах

Команда может быть использована для получения токена в скриптах:

```
TOKEN=$(ydb -p quickstart auth get-token --force)
echo "Token: $TOKEN"
```

Просмотр параметров соединения

`config info` — это сервисная команда для отладки различных проблем с соединением и аутентификацей. Команда выводит итоговые параметры соединения, вроде эндпоинта и пути базы данных, полученные рассмотрением параметров из всевозможных источников. С этими параметрами CLI подключилась бы при выполнении команды, подразумевающей соединение с базой данных. При указании глобальной опции `-v`, помимо итоговых параметров соединения, будут выведены значения всех параметров соединения из всех источников, которые CLI смог обнаружить, вместе с названиями этих источников, в порядке приоритета.

Общий вид команды:

```
ydb [global options...] config info
```

- `global options` — глобальные параметры.

Список параметров соединения

- endpoint — URL кластера базы данных.
- database — Путь к базе данных.
- **Параметры аутентификации:**
  - token — Access Token.
  - yc-token — Refresh Token.
  - sa-key-file — Service Account Key.
  - use-metadata-credentials — Metadata.
  - user
  - password
- ca-file — Корневой сертификат.
- iam-endpoint — URL IAM сервиса.

Примеры

Вывод итоговых параметров соединения

```
$ ydb -e grpcs://another.endpoint:2135 --ca-file some_certs.crt -p db123 config info
endpoint: another.endpoint:2135
yc-token: SOME_A12****************21_TOKEN
iam-endpoint: iam.api.cloud.yandex.net
ca-file: some_certs.crt
```

Вывод всех параметров соединения вместе с источниками

```
$ ydb -e grpcs://another.endpoint:2135 --ca-file some_certs.crt -p db123 -v config info
Using Yandex.Cloud Passport token from YC_TOKEN env variable

endpoint: another.endpoint:2135
yc-token: SOME_A12****************21_TOKEN
iam-endpoint: iam.api.cloud.yandex.net
ca-file: some_certs.crt
current auth method: yc-token

"ca-file" sources:
  1. Value: some_certs.crt. Got from: explicit --ca-file option

"database" sources:
  1. Value: /some/path. Got from: active profile "test_config_info"

"endpoint" sources:
  1. Value: another.endpoint:2135. Got from: explicit --endpoint option
  2. Value: db123.endpoint:2135. Got from: profile "db123" from explicit --profile option
  3. Value: some.endpoint:2135. Got from: active profile "test_config_info"

"iam-endpoint" sources:
  1. Value: iam.api.cloud.yandex.net. Got from: default value

"sa-key-file" sources:
  1. Value: /Users/username/some-sa-key-file. Got from: active profile "test_config_info"

"yc-token" sources:
  1. Value: SOME_A12****************21_TOKEN. Got from: YC_TOKEN enviroment variable
```

Вывод версии YDB CLI

С помощью подкоманды `version` вы можете узнать версию установленного YDB CLI, а также управлять автоматической проверкой доступности новой версии.

Автоматическая проверка доступности новой версии происходит при выполнении любой команды YDB CLI кроме `ydb version --enable-checks` и `ydb version --disable-checks`, но не чаще одного раза в сутки. Результат и время последней проверки сохраняются в конфигурационном файле YDB CLI.

Общий вид команды:

```
ydb [global options...] version [options...]
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.

Посмотрите описание команды:

```
ydb version --help
```

Параметры подкоманды

Параметр	Описание
`--semantic`	Вывести только номер версии.
`--check`	Проверить доступность новой версии.
`--disable-checks`	Отключить проверку доступности новой версии.
`--enable-checks`	Включить проверку доступности новой версии.

Примеры

Отключить проверку доступности новой версии

При выполнении команд YDB CLI происходит автоматическая проверка доступности новой версии. Если хост, на котором выполняется команда, не имеет доступа в интернет, это приводит к нежелательной задержке и выводу предупреждения при выполнении команды. Чтобы отключить автоматическую проверку обновления, выполните команду:

```
ydb version --disable-checks
```

Результат:

```
Latest version checks disabled
```

Вывести только номер версии

Для более удобной обработки в скриптах вы можете ограничить вывод номером версии YDB CLI:

```
ydb version --semantic
```

Результат:

```
1.9.1
```

Проверка состояния базы данных

YDB имеет встроенную систему самодиагностики, с помощью которой можно получить краткий отчёт о состоянии базы данных и информацию о выявленных проблемах.

Общий вид команды:

```
ydb [global options...] monitoring healthcheck [options...]
```

- `global options` — глобальные параметры,
- `options` — параметры подкоманды.

Параметры подкоманды

Имя	Описание
`--timeout`	Время, в течение которого должна быть выполнена операция на сервере, мс.
`--format`	Формат вывода. Возможные значения: • `pretty` — обобщенный статус базы данных. Возможные варианты значений приведены в таблице. • `json` — подробный ответ в формате JSON, содержащий иерархический список обнаруженных проблем. Перечень возможных проблем приведен в документации Healthcheck API. Значение по умолчанию — `pretty`.
`--no-merge`	Не объединять записи результата проверки состояния.
`--no-cache`	Не использовать кэшированный результат.

Примеры

Краткий результат проверки

```
ydb --profile quickstart monitoring healthcheck --format pretty
```

Проблем с базой не обнаружено:

```
Healthcheck status: GOOD
```

Обнаружена деградация базы данных:

```
Healthcheck status: DEGRADED
```

Подробный результат проверки

```
ydb --profile quickstart monitoring healthcheck --format json
```

Проблем с базой не обнаружено:

```
{
 "self_check_result": "GOOD",
 "location": {
  "id": 51059,
  "host": "my-host.net",
  "port": 19001
 }
}
```

Обнаружена деградация базы данных:

```
{
 "self_check_result": "DEGRADED",
 "issue_log": [
  {
   "id": "YELLOW-b3c0-70fb",
   "status": "YELLOW",
   "message": "Database has multiple issues",
   "location": {
    "database": {
     "name": "/my-cluster/my-database"
    }
   },
   "reason": [
    "YELLOW-b3c0-1ba8",
    "YELLOW-b3c0-1c83"
   ],
   "type": "DATABASE",
   "level": 1
  },
  {
   "id": "YELLOW-b3c0-1ba8",
   "status": "YELLOW",
   "message": "Compute is overloaded",
```

```
    "location": {
     "database": {
      "name": "/my-cluster/my-database"
     }
    },
    "reason": [
     "YELLOW-b3c0-343a-51059-User"
    ],
    "type": "COMPUTE",
    "level": 2
   },
   {
    "id": "YELLOW-b3c0-343a-51059-User",
    "status": "YELLOW",
    "message": "Pool usage is over than 99%",
    "location": {
     "compute": {
      "node": {
       "id": 51059,
       "host": "my-host.net",
       "port": 31043
      },
      "pool": {
       "name": "User"
      }
     },
     "database": {
      "name": "/my-cluster/my-database"
     }
    },
    "type": "COMPUTE_POOL",
    "level": 4
   },
   {
    "id": "YELLOW-b3c0-1c83",
    "status": "YELLOW",
    "message": "Storage usage over 75%",
    "location": {
     "database": {
      "name": "/my-cluster/my-database"
     }
    },
    "type": "STORAGE",
    "level": 2
   }
  ],
  "location": {
   "id": 117,
   "host": "my-host.net",
   "port": 19001
  }
}
```

Нагрузочное тестирование

С помощью команды `workload` вы можете запустить различные виды нагрузки для вашей БД.

Общий вид команды:

```
ydb [global options...] workload [subcommands...]
```

- `global options` — глобальные параметры.
- `subcommands` — подкоманды.

Посмотрите описание команды для запуска нагрузки:

```
ydb workload --help
```

Доступные подкоманды

В данный момент поддерживаются следующие виды нагрузочных тестов:

- Stock - симулятор склада интернет-магазина.
- Key-value - Key-Value нагрузка.
- ClickBench - аналитический бенчмарк ClickBench.
- TPC-C - TPC-C benchmark.
- TPC-H - TPC-H бенчмарк.
- TPC-DS - TPC-DS бенчмарк.
- Topic - Topic нагрузка.
- Transfer - Transfer нагрузка.
- Query - Пользовательская нагрузка.
- Fulltext - Fulltext нагрузка.
- Vector - Нагрузка векторного поиска.

Общие параметры для всех видов нагрузки

Имя параметра	Короткое имя	Описание параметра
`--seconds <значение>`	`-s <значение>`	Продолжительность теста, в секундах. Значение по умолчанию: 10.
`--threads <значение>`	`-t <значение>`	Количество параллельных потоков, создающих нагрузку. Значение по умолчанию: 10.
`--rate <значение>`	-	Суммарная частота запросов от всех потоков, в транзакциях в секунду. Значение по умолчанию: 0 (не ограничена).
`--quiet`	-	Выводит только итоговый результат теста.
`--print-timestamp`	-	Печатает время вместе со статистикой каждого временного окна.
`--client-timeout`	-	Транспортный таймаут в миллисекундах.
`--operation-timeout`	-	Таймаут на операцию в миллисекундах.
`--cancel-after`	-	Таймаут отмены операции в миллисекундах.
`--window`	-	Длительность окна сбора статистики, в секундах. Значение по умолчанию: 1.

Stock нагрузка

Важно

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Симулирует работу склада интернет-магазина: создание заказов из нескольких товаров, получение списка заказов по клиенту.

Виды нагрузки

Данный нагрузочный тест содержит 5 видов нагрузки:

- user-hist - читает заданное количество заказов покупателя с id = 10 000. Создается нагрузка на чтение одних и тех же строк из разных потоков.
- rand-user-hist - читает заданное количество заказов у случайно выбранного покупателя. Создается нагрузка на чтение из разных потоков.
- add-rand-order - создает случайно сгенерированный заказ. Например, клиент создал заказ из 2 товаров, но еще не оплатил его, поэтому остатки товаров не снижаются. В БД записывается информация о заказе и товарах. Создается нагрузка на запись и чтение (insert перед вставкой проверяет есть ли уже запись).
- put-rand-order - создает и обрабатывает случайно сгенерированный заказ. Например, покупатель создал и оплатил заказ из 2 товаров. В БД записывается информация о заказе, товарах, проверяется их наличие и уменьшаются остатки. Создается смешанная нагрузка.
- put-same-order - создает заказы с одним и тем же набором товаров. Например, все покупатели покупают один и тот же набор товаров (только что вышедший телефон и заряжающее устройство). Создается нагрузка в виде конкурентного обновления одних и тех же строк в строковой таблице.

Инициализация нагрузочного теста

Для начала работы необходимо создать строковую таблицы и заполнить их данными:

```
ydb workload stock init [init options...]
```

- `init options` — параметры инициализации.

Посмотрите описание команды для запуска нагрузки:

```
ydb workload stock init --help
```

Доступные параметры

Имя параметра	Короткое имя	Описание параметра
`--products <значение>`	`-p <значение>`	Количество видов товаров. Возможные значения от 1 до 500000. Значение по умолчанию: 100.
`--quantity <значение>`	`-q <значение>`	Количество каждого вида товара на складе. Значение по умолчанию: 1000.
`--orders <значение>`	`-o <значение>`	Первоначальное количество заказов в БД. Значение по умолчанию: 100.
`--min-partitions <значение>`	-	Минимальное количество шардов для таблиц. Значение по умолчанию: 40.
`--auto-partition <значение>`	-	Включение/выключение автошардирования. Возможные значения: 0 или 1. Значение по умолчанию: 1.

Создаются 3 строковые таблицы со следующими DDL:

```
CREATE TABLE `stock`(
    product Utf8,
    quantity Int64,
    PRIMARY KEY(product)
    )
    WITH (
        AUTO_PARTITIONING_BY_LOAD = ENABLED,
        AUTO_PARTITIONING_MIN_PARTITIONS_COUNT = <min-partitions>
        );

CREATE TABLE `orders`(
    id Uint64,
    customer Utf8,
    created Datetime,
    processed Datetime,
    PRIMARY KEY(id),
    INDEX ix_cust GLOBAL ON (customer, created)
    )
    WITH (
        READ_REPLICAS_SETTINGS = "per_az:1",
        AUTO_PARTITIONING_BY_LOAD = ENABLED,
        AUTO_PARTITIONING_MIN_PARTITIONS_COUNT = <min-partitions>,
        UNIFORM_PARTITIONS = <min-partitions>,
        AUTO_PARTITIONING_MAX_PARTITIONS_COUNT = 1000
        );

CREATE TABLE `orderLines`(
    id_order Uint64,
    product Utf8,
```

```
    quantity Int64,
    PRIMARY KEY(id_order, product)
    )
    WITH (
        AUTO_PARTITIONING_BY_LOAD = ENABLED,
        AUTO_PARTITIONING_MIN_PARTITIONS_COUNT = <min-partitions>,
        UNIFORM_PARTITIONS = <min-partitions>,
        AUTO_PARTITIONING_MAX_PARTITIONS_COUNT = 1000
        );
```

Примеры инициализации нагрузки

Создание БД с 1000 видов товаров, где каждого товара 10 000 штук, нет заказов:

```
ydb workload stock init -p 1000 -q 10000 -o 0
```

Создание БД с 10 видами товаров, где каждого товара 100 штук, есть 10 заказов и минимальное количество шардов 100:

```
ydb workload stock init -p 10 -q 100 -o 10 --min-partitions 100
```

Запуск нагрузочного теста

Для запуска нагрузки необходимо выполнить команду:

```
ydb workload stock run [workload type...] [global workload options...] [specific workload options...]
```

В течение теста на экран выводится статистика по нагрузке для каждого временного окна.

- `workload type` — виды нагрузки.
- `global workload options` - общие параметры для всех видов нагрузки.
- `specific workload options` - параметры конкретного вида нагрузки.

Посмотрите описание команды для запуска нагрузки:

```
ydb workload run --help
```

Нагрузка user-hist

Данный вид нагрузки читает заданное количество заказов покупателя с id = 10 000.

YQL запрос:

```
DECLARE $cust AS Utf8;
DECLARE $limit AS UInt32;

SELECT id, customer, created
FROM orders VIEW ix_cust
WHERE customer = 'Name10000'
ORDER BY customer DESC, created DESC
LIMIT $limit;
```

Для запуска данного вида нагрузки необходимо выполнить команду:

```
ydb workload stock run user-hist [global workload options...] [specific workload options...]
```

- `global workload options` - общие параметры для всех видов нагрузки.
- `specific workload options` - параметры конкретного вида нагрузки

Параметры для user-hist

Имя параметра	Короткое имя	Описание параметра
`--limit <значение>`	`-l <значение>`	Необходимое количество заказов. Значение по умолчанию: 10.

Нагрузка rand-user-hist

Данный вид нагрузки читает заданное количество заказов случайно выбранных покупателей.

YQL запрос:

```
DECLARE $cust AS Utf8;
DECLARE $limit AS UInt32;

SELECT id, customer, created
FROM orders VIEW ix_cust
WHERE customer = $cust
ORDER BY customer DESC, created DESC
LIMIT $limit;
```

Для запуска данного вида нагрузки необходимо выполнить команду:

```
ydb workload stock run rand-user-hist [global workload options...] [specific workload options...]
```

- `global workload options` - общие параметры для всех видов нагрузки.
- `specific workload options` - параметры конкретного вида нагрузки

Параметры для rand-user-hist

Имя параметра	Короткое имя	Описание параметра
`--limit <значение>`	`-l <значение>`	Необходимое количество заказов. Значение по умолчанию: 10.

Нагрузка add-rand-order

Данный вид нагрузки создает случайно сгенерированный заказ. В заказ помещаются несколько различных товаров по 1 штуке. Количество видов товара в заказе генерируется случайно по экспоненциальному распределению.

YQL запрос:

```
DECLARE $ido AS UInt64;
DECLARE $cust AS Utf8;
DECLARE $lines AS List<Struct<product:Utf8,quantity:Int64>>;
DECLARE $time AS DateTime;

INSERT INTO `orders`(id, customer, created) VALUES
    ($ido, $cust, $time);
UPSERT INTO `orderLines`(id_order, product, quantity)
    SELECT $ido, product, quantity FROM AS_TABLE( $lines );
```

Для запуска данного вида нагрузки необходимо выполнить команду:

```
ydb workload stock run add-rand-order [global workload options...] [specific workload options...]
```

- `global workload options` - общие параметры для всех видов нагрузки.
- `specific workload options` - параметры конкретного вида нагрузки

Параметры для add-rand-order

Имя параметра	Короткое имя	Описание параметра
`--products <значение>`	`-p <значение>`	Количество видов товара в тесте. Значение по умолчанию: 100.

Нагрузка put-rand-order

Данный вид нагрузки создает случайно сгенерированный заказ и обрабатывает его. В заказ помещаются несколько различных товаров по 1 штуке. Количество видов товара в заказе генерируется случайно по экспоненциальному распределению. Обработка заказа заключается в уменьшении количества заказанных товаров на складе.

YQL запрос:

```
DECLARE $ido AS UInt64;
DECLARE $cust AS Utf8;
DECLARE $lines AS List<Struct<product:Utf8,quantity:Int64>>;
DECLARE $time AS DateTime;

INSERT INTO `orders`(id, customer, created) VALUES
    ($ido, $cust, $time);

UPSERT INTO `orderLines`(id_order, product, quantity)
    SELECT $ido, product, quantity FROM AS_TABLE( $lines );

$prods = SELECT * FROM orderLines AS p WHERE p.id_order = $ido;

$cnt = SELECT COUNT(*) FROM $prods;

$newq =
    SELECT
        p.product AS product,
        COALESCE(s.quantity, 0) - p.quantity AS quantity
    FROM $prods AS p
    LEFT JOIN stock AS s
    ON s.product = p.product;

$check = SELECT COUNT(*) AS cntd FROM $newq as q WHERE q.quantity >= 0;

UPSERT INTO stock
    SELECT product, quantity FROM $newq WHERE $check=$cnt;

$upo = SELECT id, $time AS tm FROM orders WHERE id = $ido AND $check = $cnt;

UPSERT INTO orders SELECT id, tm AS processed FROM $upo;

SELECT * FROM $newq AS q WHERE q.quantity < 0
```

Для запуска данного вида нагрузки необходимо выполнить команду:

```
ydb workload stock run put-rand-order [global workload options...] [specific workload options...]
```

- `global workload options` - общие параметры для всех видов нагрузки.

- `specific workload options` - параметры конкретного вида нагрузки

Параметры для put-rand-order

Имя параметра	Короткое имя	Описание параметра
`--products <значение>`	`-p <значение>`	Количество видов товара в тесте. Значение по умолчанию: 100.

Нагрузка put-same-order

Данный вид нагрузки создает заказ с одним и тем же набором товаров и обрабатывает его. Обработка заказа заключается в уменьшении количества заказанных товаров на складе.

YQL запрос:

```
DECLARE $ido AS UInt64;
DECLARE $cust AS Utf8;
DECLARE $lines AS List<Struct<product:Utf8,quantity:Int64>>;
DECLARE $time AS DateTime;

INSERT INTO `orders`(id, customer, created) VALUES
    ($ido, $cust, $time);

UPSERT INTO `orderLines`(id_order, product, quantity)
    SELECT $ido, product, quantity FROM AS_TABLE( $lines );

$prods = SELECT * FROM orderLines AS p WHERE p.id_order = $ido;

$cnt = SELECT COUNT(*) FROM $prods;

$newq =
    SELECT
        p.product AS product,
        COALESCE(s.quantity, 0) - p.quantity AS quantity
    FROM $prods AS p
    LEFT JOIN stock AS s
    ON s.product = p.product;

$check = SELECT COUNT(*) as cntd FROM $newq AS q WHERE q.quantity >= 0;

UPSERT INTO stock
    SELECT product, quantity FROM $newq WHERE $check=$cnt;

$upo = SELECT id, $time AS tm FROM orders WHERE id = $ido AND $check = $cnt;

UPSERT INTO orders SELECT id, tm AS processed FROM $upo;

SELECT * FROM $newq AS q WHERE q.quantity < 0
```

Для запуска данного вида нагрузки необходимо выполнить команду:

```
ydb workload stock run put-same-order [global workload options...] [specific workload options...]
```

- `global workload options` - общие параметры для всех видов нагрузки.
- `specific workload options` - параметры конкретного вида нагрузки

Параметры для put-same-order

Имя параметра	Короткое имя	Описание параметра
`--products <значение>`	`-p <значение>`	Количество видов товара в каждом заказе. Значение по умолчанию: 100.

Примеры запуска нагрузок

- Запуск `add-rand-order` нагрузки на 5 секунд в 10 потоков с 1000 видами товаров.

```
ydb workload stock run add-rand-order -s 5 -t 10 -p 1000
```

Возможный результат:

```
Elapsed Txs/Sec Retries Errors  p50(ms) p95(ms) p99(ms) pMax(ms)
1           132 0       0       69      108     132     157
2           157 0       0       63      88      97      104
3           156 0       0       62      84      104     120
4           160 0       0       62      77      90      94
5           174 0       0       61      77      97      100

Txs     Txs/Sec Retries Errors  p50(ms) p95(ms) p99(ms) pMax(ms)
779       155.8 0       0       62      89      108     157
```

- Запуск `put-same-order` нагрузки на 5 секунд в 5 потоков с 2 видами товаров в заказе с распечаткой только итоговых результатов.

```
ydb workload stock run put-same-order -s 5 -t 5 -p 1000 --quiet
```

Возможный результат:

```
Txs     Txs/Sec Retries Errors  p50(ms) p95(ms) p99(ms) pMax(ms)
16          3.2 67      3       855     1407    1799    1799
```

- Запуск `rand-user-hist` нагрузки на 5 секунд в 100 потоков с распечаткой времени каждого временного окна.

```
ydb workload stock run rand-user-hist -s 5 -t 10 --print-timestamp
```

Возможный результат:

```
Elapsed Txs/Sec Retries Errors  p50(ms) p95(ms) p99(ms) pMax(ms)        Timestamp
1          1046 0       0       7       16      25      50      2022-02-08T17:47:26Z
2          1070 0       0       7       17      22      28      2022-02-08T17:47:27Z
3          1041 0       0       7       17      22      28      2022-02-08T17:47:28Z
4          1045 0       0       7       17      23      31      2022-02-08T17:47:29Z
5           998 0       0       8       18      23      42      2022-02-08T17:47:30Z

Txs     Txs/Sec Retries Errors  p50(ms) p95(ms) p99(ms) pMax(ms)
5200       1040 0       0       8       17      23      50
```

Интерпретация результатов

- `Elapsed` - номер временного окна. По умолчанию временное окно равно 1 секунде.
- `Txs/sec` - количество успешных транзакций нагрузки во временном окне.
- `Retries` - количество повторных попыток исполнения транзакции клиентом во временном окне.
- `Errors` - количество ошибок, случившихся во временном окне.
- `p50(ms)` - 50-й перцентиль latency запросов в мс.
- `p95(ms)` - 95-й перцентиль latency запросов в мс.
- `p99(ms)` - 99-й перцентиль latency запросов в мс.
- `pMax(ms)` - 100-й перцентиль latency запросов в мс.
- `Timestamp` - временная метка конца временного окна.

ClickBench нагрузка

Нагрузка базируется на данных и запросах из репозитория https://github.com/ClickHouse/ClickBench, а запросы и схема таблицы адаптированы под YDB.

Тест генерирует типичную рабочую нагрузку в следующих областях: анализ потока кликов и трафика, веб-аналитика, машинно-генерируемые данные, структурированные журналы и данные о событиях. Он охватывает типичные запросы в аналитике и информационных панелях реального времени.

Набор данных для бенчмарка был получен на основе фактической записи трафика одной из крупнейших в мире платформ веб-аналитики. Он анонимизирован, при этом сохраняет все основные распределения данных. Набор запросов был импровизирован, чтобы отразить реалистичные рабочие нагрузки, в то время как запросы не исходят непосредственно из производства.

Общие параметры команд

Все команды поддерживают общий параметр `--path`, который задает путь к таблице в базе данных:

```
ydb  workload clickbench --path clickbench/hits ...
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--path` или `-p`	Путь к таблице	`clickbench/hits`

Инициализация нагрузочного теста

Перед запуском теста создайте таблицу:

```
ydb workload clickbench --path clickbench/hits init
```

Посмотрите описание команды для запуска нагрузки:

```
ydb workload clickbench init --help
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--store <значение>`	Тип хранилища таблиц. Возможные значения: `row`, `column`, `external-s3`	`column`
`--external-s3-prefix <значение>`	Актуально только для внешних таблиц. Корневой путь к набору данных в S3-хранилище	
`--external-s3-endpoint <значение>` или `-e <значение>`	Актуально только для внешних таблиц. Ссылка на S3-Bucket с данными	
`--string`	Использовать для текстовых полей тип `String`	`Utf8`
`--datetime`	Использовать для полей, связанных со временем типа `Date`, `Datetime` и `Timestamp`	`Date32`, `Datetime64` и `Timestamp64`.
`--partition-size`	Максимальный размер партиции в мегабайтах (`AUTO_PARTITIONING_PARTITION_SIZE_MB`) для строчных таблиц.	2000
`--clear`	Если по указанному пути таблица уже была создана, она будет удалена.	
`--dry-run`	Не выполнять инициализационные запросы, а только вывести их текст.	

Загрузка данных в таблицу

Загрузите данные в таблицу. Для этого скачайте архив с данными, затем загрузите данные в таблицу:

```
wget https://datasets.clickhouse.com/hits_compatible/hits.csv.gz
ydb workload clickbench --path clickbench/hits import files --input hits.csv.gz
```

В качестве исходных файлов можно использовать как распакованные и запакованные csv и tsv файлы, так и директории с такими файлами.

Доступные параметры

Имя	Описание	Значение по умолчанию
`--input <путь>` или `-i <путь>`	Путь к исходным файлам с данными. Поддерживаются как распакованные и запакованные csv и tsv файлы, так и директории с такими файлами. Данные могут быть загружены с официального сайта ClickBench: csv.gz, tsv.gz. Для ускорения загрузки можно разбить эти файлы на более мелкие части, в этом случае части буду загружаться параллельно.	

`--state <путь>`	Путь к файлу состояния загрузки. Если загрузка была прервана по какой-то причине, при новом запуске загрузка будет продолжена с того же места.	
`--clear-state`	Актуально, если задан параметр `--state`. Очистить файл состояния и начать загрузку сначала.	
`--dry-run`	Не выполнять загрузку данных в базу данных, только подготовить данные и вывести об этом сообщение.	

Общие параметры команды import

Имя	Описание	Значение по умолчанию
`--upload-threads <значение>` или `-t <значение>`	Количество потоков исполнения для подготовки данных.	Рассчитывается автоматически и равно числу доступных ядер на клиентской машине.
`--bulk-size <значение>`	Размер порции для отправки данных в строках.	10000
`--max-in-flight <значение>`	Максимальное количество порций данных, одновременно находящихся в обработке.	128
`--file-output-path <значение>` или `-f <путь>`	Если эта опция установлена, данные не будут загружены в базу данных, а будут сохранены в каталоге <путь>.	

Запуск нагрузочного теста

Запустите нагрузку:

```
ydb workload clickbench  --path clickbench/hits run
```

В течение теста на экран выводится статистика по нагрузке для каждого запроса.

Посмотрите описание команды для запуска нагрузки:

```
ydb workload clickbench run --help
```

Общие параметры для всех видов нагрузки

Имя	Описание	Значение по умолчанию
`--dry-run`	Не выполнять инициализационные запросы, а только вывести их текст.	
`--check-canonical` или `-c`	Использовать специальную версию запросов (они имеют детерминированные ответы) и сверять результаты с каноническими.	
`--output <значение>`	Имя файла, в котором будут сохранены результаты выполнения запросов.	`results.out`
`--iterations <значение>`	Количество выполнений каждого из запросов нагрузки.	`1`
`--json <имя>`	Имя файла, в котором будет сохранена статистика выполнения запросов в формате `json`.	Файл не сохраняется
`--ministat <имя>`	Имя файла, в котором будет сохранена статистика выполнения запросов в формате `ministat`.	Файл не сохраняется
`--csv <имя>`	Имя файла для сохранения CSV-версии таблицы с результатами.	Файл не сохраняется
`--plan <имя>`	Имя файла для сохранения плана запроса. Если задано, то сохраняются файлы `<имя>.<номер запроса>.explain` и `<имя>.<номер запроса>.<номер итерации>` с планами в нескольких форматах: `ast`, `json`, `svg` и `table`.	Планы не сохраняются.
`--query-prefix <префикс>`	Префикс запроса. Каждый префикс будет добавлен отдельной строкой в начало каждого запроса. Если нужно указать несколько префиксов, используйте параметр несколько раз.	По умолчанию не задан
`--retries`	Количество повторных попыток выполнения каждого запроса	`0`
`--include`	Имена, номера или диапазоны номеров запросов, которые нужно выполнить в рамках нагрузки. Указываются через запятую, например: `1,2,4-6`.	Все запросы
`--exclude`	Имена, номера или диапазоны номеров запросов, которые нужно исключить из нагрузки. Указываются через запятую, например: `1,2,4-6`.	
`--verbose` или `-v`	Выводить больше информации на экран в процессе выполнения запросов.	
`--global-timeout <значение>`	Общий таймаут на выполнение всех запросов. Задаётся в текстовом формате, например: `0.5s`, `1m`, `100us` и т.д.	Отсутствует. Время не ограничено.

`--request-timeout <значение>`	Таймаут на выполнение каждой итерации каждого запроса. Задаётся в текстовом формате, например: `0.5s`, `1m`, `100us` и т.д.	Отсутствует. Время не ограничено.
`--threads <значение>` или `-t <значение>`	Количество потоков, генерирующих нагрузку. Ноль означает, что запросы будут выполняться в основном потоке; при ином значении запросы будут перемешаны.	`0`
`--stats <значение>`	Режим сбора расширенной статистики выполнения. Возможные значения: `full`, `profile`.	`full`

Опции, специфичные для ClickBench

Имя	Описание	Значение по умолчанию
`--syntax <значение>`	Какой вариант синтаксиса запросов использовать: `yql`. Про работу с синтаксисом YQL можно прочитать здесь.	`yql`

Очистка данных теста

Запустите очистку:

```
ydb workload clickbench --path clickbench/hits clean
```

Команда не имеет параметров.

Key-Value нагрузка

Простой вид нагрузки, использующий БД YDB как Key-Value хранилище.

Виды нагрузки

Данный нагрузочный тест содержит несколько видов нагрузки:

- upsert — при помощи операции `UPSERT` вставляет в заранее созданную командой init таблицу строки, которые представляют из себя кортежи `(key1, key2, ... keyK, value1, value2, ... valueN)`, числа K и N задаются в параметрах.
- insert — функционал такой же, как у нагрузки upsert, но для вставки используется операция `INSERT`.
- select — читает данные при помощи операции `SELECT * WHERE key = $key`. Запрос всегда затрагивает все колонки таблицы, но не всегда является точечным, количеством вариаций primary ключа можно управлять с помощью параметров.
- read-rows - читает данные при помощи операции ReadRows, выполняющей более быстрые чтения по ключу, чем `SELECT`. Запрос всегда затрагивает все колонки таблицы, но не всегда является точечным, количеством вариаций primary ключа можно управлять с помощью параметров.
- mixed - одновременно пишет и читает данные, дополнительно проверяя что все записанные данные успешно читаются.

Инициализация нагрузочного теста

Для начала работы необходимо создать таблицы, при создании можно указать, сколько строк необходимо вставить при инициализации:

```
ydb workload kv init [init options...]
```

- `init options` — параметры инициализации.

Посмотрите описание команды для инициализации таблицы:

```
ydb workload kv init --help
```

Доступные параметры

Имя параметра	Описание параметра
`--init-upserts <значение>`	Количество операций вставки, которые нужно сделать при инициализации. Значение по умолчанию: 1000.
`--min-partitions`	Минимальное количество шардов для таблиц. Значение по умолчанию: 40.
`--partition-size`	Максимальный размер одной партиции (настройка `AUTO_PARTITIONING_PARTITION_SIZE_MB`). Значение по умолчанию 2000.
`--auto-partition`	Включение/выключение автошардирования. Возможные значения: 0 или 1. Значение по умолчанию: 1.
`--max-first-key`	Максимальное значение primary ключа таблицы. Значение по умолчанию: .
`--len`	Размер строк в байтах, которые вставляются в таблицу, как values. Значение по умолчанию: 8.
`--cols`	Количество колонок в таблице. Значение по умолчанию: 2, считая Key.
`--int-cols`	Количество первых колонок в таблице которые будут иметь тип `Uint64`, последующие колонки будут иметь тип `String`. Значение по умолчанию 1.
`--key-cols`	Количество первых колонок в таблице входящих в ключ. Значение по умолчанию 1.
`--rows`	Количество затрагиваемых строк в одном запросе. Значение по умолчанию: 1.

Для создания таблицы используется следующая команда:

```
CREATE TABLE `kv_test`(
    c0 Uint64,
    c1 Uint64,
    ...
    cI Uint64,
    cI+1 String,
    ...
    cN String,
    PRIMARY KEY(c0, c1, ... cK)) WITH (
        AUTO_PARTITIONING_BY_LOAD = ENABLED,
        AUTO_PARTITIONING_MIN_PARTITIONS_COUNT = partsNum,
        UNIFORM_PARTITIONS = partsNum,
        AUTO_PARTITIONING_PARTITION_SIZE_MB = partSize,
        AUTO_PARTITIONING_MAX_PARTITIONS_COUNT = 1000
    )
)
```

Примеры инициализации нагрузки

Пример команды создания таблицы с 1000 строками:

```
ydb workload kv init --init-upserts 1000
```

Удаление таблицы

После завершения работы можно удалить таблицу:

```
ydb workload kv clean
```

Исполняется следующая YQL команда:

```
DROP TABLE `kv_test`
```

Примеры использования clean

```
ydb workload kv clean
```

Запуск нагрузочного теста

Для запуска нагрузки необходимо выполнить команду:

```
ydb workload kv run [workload type...] [global workload options...] [specific workload options...]
```

В течение теста на экран выводится статистика по нагрузке для каждого временного окна.

- `workload type` — виды нагрузки.
- `global workload options` — общие параметры для всех видов нагрузки.
- `specific workload options` — параметры конкретного вида нагрузки.

Посмотрите описание команды для запуска нагрузки:

```
ydb workload kv run --help
```

Общие параметры для всех видов нагрузки

Имя параметра	Короткое имя	Описание параметра
`--seconds <значение>`	`-s <значение>`	Продолжительность теста, сек. Значение по умолчанию: 10.
`--threads <значение>`	`-t <значение>`	Количество параллельных потоков, создающих нагрузку. Значение по умолчанию: 10.
`--rate <значение>`	-	Суммарная частота запросов всех потоков, в транзакциях в секунду Значение по умолчанию: 0 (не ограничена).
`--quiet`	-	Выводит только итоговый результат теста.
`--print-timestamp`	-	Печатать время вместе со статистикой каждого временного окна.
`--client-timeout`	-	Транспортный таймаут в миллисекундах.
`--operation-timeout`	-	Таймаут на операцию в миллисекундах.
`--cancel-after`	-	Таймаут отмены операции в миллисекундах.
`--window`	-	Длительность окна сбора статистики в секундах. Значение по умолчанию: 1.
`--max-first-key`	-	Максимальное значение primary ключа таблицы. Значение по умолчанию: .
`--cols`	-	Количество колонок в таблице. Значение по умолчанию: 2, считая Key.
`--int-cols`	-	Количество первых колонок в таблице которые будут иметь тип `Uint64`, последующие колонки будут иметь тип `String`. Значение по умолчанию 1.
`--key-cols`	-	Количество первых колонок в таблице входящих в ключ. Значение по умолчанию 1.
`--rows`	-	Количество затрагиваемых строк в одном запросе. Значение по умолчанию: 1.

Нагрузка upsert

Данный вид нагрузки вставляет кортежи (key1, key2, ... keyK, value1, value2, ... valueN)

Для запуска данного вида нагрузки необходимо выполнить команду:

```
ydb workload kv run upsert [global workload options...] [specific workload options...]
```

- `global workload options` - общие параметры для всех видов нагрузки.
- `specific workload options` - параметры конкретного вида нагрузки.

Например для параметров `--rows 2 --cols 3 --int-cols 2` YQL Запрос будет выглядеть так:

```
DECLARE $c0_0 AS Uint64;
DECLARE $c0_1 AS Uint64;
DECLARE $c0_2 AS String;
DECLARE $c1_0 AS Uint64;
```

```
DECLARE $c1_1 AS Uint64;
DECLARE $c1_2 AS String;
UPSERT INTO `kv_test` (c0, c1, c2) VALUES ($c0_0, $c0_1, $c0_2), ($c1_0, $c1_1, $c1_2)
```

Параметры для upsert

Имя параметра	Описание параметра
`--len`	Размер строк в байтах, которые вставляются в таблицу, как values. Значение по умолчанию: 8.

Нагрузка insert

Данный вид нагрузки вставляет кортежи (key1, key2, ... keyK, value1, value2, ... valueN)

Для запуска данного вида нагрузки необходимо выполнить команду:

```
ydb workload kv run insert [global workload options...] [specific workload options...]
```

- `global workload options` - общие параметры для всех видов нагрузки.
- `specific workload options` - параметры конкретного вида нагрузки.

Например для параметров `--rows 2 --cols 3 --int-cols 2` YQL Запрос будет выглядеть так:

```
DECLARE $c0_0 AS Uint64;
DECLARE $c0_1 AS Uint64;
DECLARE $c0_2 AS String;
DECLARE $c1_0 AS Uint64;
DECLARE $c1_1 AS Uint64;
DECLARE $c1_2 AS String;
INSERT INTO `kv_test` (c0, c1, c2) VALUES ($c0_0, $c0_1, $c0_2), ($c1_0, $c1_1, $c1_2)
```

Параметры для insert

Имя параметра	Описание параметра
`--len`	Размер строк в байтах, которые вставляются в таблицу, как values. Значение по умолчанию: 8.

Нагрузка select

Данный вид нагрузки создает SELECT запросы, возвращающие строки по точному совпадению primary ключа.

Для запуска данного вида нагрузки необходимо выполнить команду:

```
ydb workload kv run select [global workload options...]
```

- `global workload options` - общие параметры для всех видов нагрузки.

Например для параметров `--rows 2 --cols 3 --int-cols 2 --key-cols 2` YQL Запрос будет выглядеть так:

```
DECLARE $r0_0 AS Uint64;
DECLARE $r0_1 AS Uint64;
DECLARE $r1_0 AS Uint64;
DECLARE $r1_1 AS Uint64;
SELECT c0, c1, c2 FROM `kv_test` WHERE c0 = $r0_0 AND c1 = $r0_1 OR c0 = $r1_0 AND c1 = $r1_1
```

Нагрузка read-rows

Данный вид нагрузки создает ReadRows запросы, возвращающие строки по точному совпадению primary ключа.

Для запуска данного вида нагрузки необходимо выполнить команду:

```
ydb workload kv run read-rows [global workload options...]
```

- `global workload options` - общие параметры для всех видов нагрузки.

Нагрузка mixed

Данный вид нагрузки одновременно вставляет и читает кортежи (key1, key2, ... keyK, value1, value2, ... valueN), дополнительно проверяя что все записанные данные успешно читаются.

Для запуска данного вида нагрузки необходимо выполнить команду:

```
ydb workload kv run mixed [global workload options...] [specific workload options...]
```

- `global workload options` - общие параметры для всех видов нагрузки.
- `specific workload options` - параметры конкретного вида нагрузки.

Параметры для mixed

Имя параметра	Описание параметра
`--len`	Размер строк в байтах, которые вставляются в таблицу, как values. Значение по умолчанию: 8.

`--change-partitions-size`	Включение/выключение случайного изменения настройки `AUTO_PARTITIONING_PARTITION_SIZE_MB`. Возможные значения: 0 или 1. Значение по умолчанию: 0.
`--do-select`	Включение/выключение чтений при помощи запроса select. Возможные значения: 0 или 1. Значение по умолчанию: 1.
`--do-read-rows`	Включение/выключение чтений при помощи запроса read-rows. Возможные значения: 0 или 1. Значение по умолчанию: 1.

Topic нагрузка

Нагружает топики YDB, используя их в качестве очередей сообщений. Для имитации реальной нагрузки вы можете изменять различные входные параметры: число сообщений, размер сообщений, целевую скорость записи, число читателей и писателей.

В процессе работы на консоль выдаются результаты, которые включают количество записанных сообщений, скорость записи сообщений и пр.

Чтобы нагрузить топик:

1. Инициализируйте нагрузку.
2. Запустите один из доступных видов нагрузки:
   - write — генерация сообщений и их запись в топик в асинхронном режиме;
   - read — асинхронное чтение сообщений из топика;
   - full — одновременное асинхронное чтение и запись сообщений.

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Инициализация нагрузочного теста

Перед запуском нагрузки ее необходимо инициализировать. При инициализации будет создан топик `workload-topic` с указанными параметрами. Инициализация выполняется следующей командой:

```
ydb [global options...] workload topic init [options...]
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.

Параметры подкоманды:

Имя параметра	Описание параметра
`--topic`	Имя топика. Значение по умолчанию: `workload-topic`.
`--partitions`, `-p`	Количество партиций топика. Значение по умолчанию: `128`.
`--consumers`, `-c`	Количество читателей топика. Значение по умолчанию: `1`.
`--consumer-prefix`	Префикс имени читателей. Значение по умолчанию: `workload-consumer`. Например, если количество читателей `--consumers` равно `2` и префикс `--consumer-prefix` равен `workload-consumer`, то будут использованы следующие имена читателей: `workload-consumer-0`, `workload-consumer-1`.

Чтобы создать топик с `256` партициями и `2` читателями, выполните команду:

```
ydb --profile quickstart workload topic init --partitions 256 --consumers 2
```

Запуск нагрузки

Нагрузка на запись

Этот вид нагрузки генерирует и записывает сообщения в топик в асинхронном режиме.

Общий вид команды для запуска нагрузки на запись:

```
ydb [global options...] workload topic run write [options...]
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.

Посмотреть описание команды для запуска нагрузки на запись:

```
ydb workload topic run write --help
```

Параметры подкоманды:

Имя параметра	Описание параметра
`--seconds`, `-s`	Продолжительность теста в секундах. Значение по умолчанию: `60`.
`--window`, `-w`	Длительность окна сбора статистики в секундах. Значение по умолчанию: `1`.
`--quiet`, `-q`	Выводить только итоговый результат теста.

`--print-timestamp`	Печатать время вместе со статистикой каждого временного окна.
`--warmup`	Время прогрева теста в секундах. В течение этого времени не вычисляется статистика, оно необходимо для устранения переходных процессов при старте. Значение по умолчанию: `5`.
`--percentile`	Процентиль в выводе статистики. Значение по умолчанию: `50`.
`--topic`	Имя топика. Значение по умолчанию: `workload-topic`.
`--threads`, `-t`	Количество потоков писателя. Каждый поток будет писать во все партиции указанного топика. Значение по умолчанию: `1`.
`--message-size`, `-m`	Размер сообщения в байтах. Возможно задание в КБ, МБ, ГБ путем добавления суффиксов `K`, `M`, `G` соответственно. Значение по умолчанию: `10K`.
`--message-rate`	Целевая суммарная скорость записи, сообщений в секунду. Исключает использование параметра `--byte-rate`. Значение по умолчанию: `0` (нет ограничения).
`--byte-rate`	Целевая суммарная скорость записи, байт в секунду. Исключает использование параметра `--message-rate`. Возможно задание в КБ/с, МБ/с, ГБ/с путем добавления суффиксов `K`, `M`, `G` соответственно. Значение по умолчанию: `0` (нет ограничения).
`--codec`	Кодек, используемый для сжатия сообщений на клиенте перед отправкой на сервер. Сжатие увеличивает затраты CPU на клиенте при записи и чтении сообщений, но обычно позволяет уменьшить объем передаваемых по сети и хранимых данных. При последующем чтении сообщений подписчиками они автоматически разжимаются использованным при записи кодеком, не требуя указания каких-либо параметров. Возможные значения: `RAW` - без сжатия (по умолчанию), `GZIP`, `ZSTD`.
`--use-tx`	Использовать транзакции при записи сообщений. По умолчанию транзакции использоваться не будут.
`--tx-commit-interval`	Интервал коммита транзакций в миллисекундах. Коммит транзакции произойдёт, когда выполнится одно из условий: пройдёт заданное в параметре `--tx-commit-interval` количество времени или будет получено заданное в параметре `--tx-commit-messages` количество сообщений. Значение по умолчанию: `1000`.
`--tx-commit-messages`	Количество сообщений, необходимое для коммита транзакции. Коммит транзакции произойдёт, когда выполнится одно из условий: пройдёт заданное в параметре `--tx-commit-interval` количество времени или будет получено заданное в параметре `--tx-commit-messages` количество сообщений. Значение по умолчанию: `1 000 000`.

Чтобы записать в `100` потоков писателей с целевой скоростью `80` МБ/с в течение `10` секунд, выполните следующую команду:

```
ydb --profile quickstart workload topic run write --threads 100 --byte-rate 80M
```

В процессе работы будет выводиться статистика по промежуточным временным окнам, а по окончании теста — итоговая статистика за все время работы:

```
Window  Write speed     Write time      Inflight
#       msg/s   MB/s    percentile,ms   percentile,msg
1       20      0       1079            72
2       8025    78      1415            78
3       7987    78      1431            79
4       7888    77      1471            101
5       8126    79      1815            116
6       7018    68      1447            79
7       8938    87      2511            159
8       7055    68      1463            78
9       7062    69      1455            79
10      9912    96      3679            250
Window  Write speed     Write time      Inflight
#       msg/s   MB/s    percentile,ms   percentile,msg
Total   7203    70      3023            250
```

- `Window` — порядковый номер временного окна сбора статистики;
- `Write speed` — скорость записи сообщений, сообщений в секунду и МБ/с;
- `Write time` — процентиль времени записи сообщения, мс.
- `Inflight` — максимальное число сообщений, ожидающих подтверждения по всем партициям.

Если в тесте используются транзакции, то дополнительно будет выведена статистика времени коммита транзакции в колонке `Commit time`.

Нагрузка на чтение

Этот вид нагрузки асинхронно читает сообщения из топика. Чтобы в топике появились сообщения, перед началом чтения запустите нагрузку на запись.

Общий вид команды для запуска нагрузки на чтение:

```
ydb [global options...] workload topic run read [options...]
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.

Посмотреть описание команды для запуска нагрузки на чтение:

```
ydb workload topic run read --help
```

Параметры подкоманды:

Имя параметра	Описание параметра
`--seconds`, `-s`	Продолжительность теста в секундах. Значение по умолчанию: `60`.
`--window`, `-w`	Длительность окна сбора статистики в секундах. Значение по умолчанию: `1`.
`--quiet`, `-q`	Выводить только итоговый результат теста.
`--print-timestamp`	Печатать время вместе со статистикой каждого временного окна.
`--warmup`	Время прогрева теста в секундах. В течение этого времени не вычисляется статистика, оно необходимо для устранения переходных процессов при старте. Значение по умолчанию: `5`.
`--percentile`	Процентиль в выводе статистики. Значение по умолчанию: `50`.
`--topic`	Имя топика. Значение по умолчанию: `workload-topic`.
`--consumers`, `-c`	Количество читателей. Значение по умолчанию: `1`.
`--consumer-prefix`	Префикс имени читателей. Значение по умолчанию: `workload-consumer`. Например, если количество читателей `--consumers` равно `2` и префикс `--consumer-prefix` равен `workload-consumer`, то будут использованы следующие имена читателей: `workload-consumer-0`, `workload-consumer-1`.
`--threads`, `-t`	Количество потоков читателя. Значение по умолчанию: `1`.

Чтобы выполнить чтение из топика с помощью `2` читателей, каждый из который имеет `100` потоков, выполните следующую команду:

```
ydb --profile quickstart workload topic run read --consumers 2 --threads 100
```

В процессе работы будет выводиться статистика по промежуточным временным окнам, а по окончании теста — итоговая статистика за все время работы:

```
Window  Lag             Lag time        Read speed      Full time
#       percentile,msg  percentile,ms   msg/s   MB/s    percentile,ms
1       0               0               0       0       0
2       30176           0               66578   650     0
3       30176           0               68999   674     0
4       30176           0               66907   653     0
5       27835           0               67628   661     0
6       30176           0               67938   664     0
7       30176           0               71628   700     0
8       20338           0               61367   599     0
9       30176           0               61770   603     0
10      30176           0               58291   569     0
Window  Lag             Lag time        Read speed      Full time
#       percentile,msg  percentile,ms   msg/s   MB/s    percentile,ms
Total   30176           0               80267   784     0
```

- `Window` — порядковый номер временного окна сбора статистики.
- `Lag` — максимальное в окне сбора статистики отставание читателя. Учитываются сообщения по всем партициям.
- `Lag time` — процентиль времени задержки сообщений в мс.
- `Read` — Скорость чтения сообщений читателем, сообщений в секунду и МБ/с.
- `Full time` — процентиль времени полной обработки сообщения (от записи писателем до чтения читателем) в мс.

Нагрузка на чтение и запись

Этот вид нагрузки одновременно асинхронно пишет и читает сообщения. Выполнение данной команды эквивалентно одновременному запуску нагрузок на чтение и запись.

Общий вид команды для запуска нагрузки на чтение и запись:

```
ydb [global options...] workload topic run full [options...]
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.

Посмотреть описание команды для запуска нагрузки на чтение и запись:

```
ydb workload topic run full --help
```

Параметры подкоманды:

Имя параметра	Описание параметра
`--seconds`, `-s`	Продолжительность теста в секундах. Значение по умолчанию: `60`.
`--window`, `-w`	Длительность окна сбора статистики в секундах. Значение по умолчанию: `1`.
`--quiet`, `-q`	Выводить только итоговый результат теста.
`--print-timestamp`	Печатать время вместе со статистикой каждого временного окна.
`--warmup`	Время прогрева теста в секундах. В течение этого времени не вычисляется статистика, оно необходимо для устранения переходных процессов при старте. Значение по умолчанию: `5`.
`--percentile`	Процентиль в выводе статистики. Значение по умолчанию: `50`.
`--topic`	Имя топика. Значение по умолчанию: `workload-topic`.
`--producer-threads`, `-p`	Количество потоков писателя. Значение по умолчанию: `1`.
`--message-size`, `-m`	Размер сообщения в байтах. Возможно задание в КБ, МБ, ГБ путем добавления суффиксов `K`, `M`, `G` соответственно. Значение по умолчанию: `10K`.
`--message-rate`	Целевая суммарная скорость записи, сообщений в секунду. Исключает использование параметра `--message-rate`. Значение по умолчанию: `0` (нет ограничения).
`--byte-rate`	Целевая суммарная скорость записи, байт в секунду. Исключает использование параметра `--byte-rate`. Возможно задание в КБ/с, МБ/с, ГБ/с путем добавления суффиксов `K`, `M`, `G` соответственно. Значение по умолчанию: `0` (нет ограничения).
`--codec`	Кодек, используемый для сжатия сообщений на клиенте перед отправкой на сервер. Сжатие увеличивает затраты CPU на клиенте при записи и чтении сообщений, но обычно позволяет уменьшить объем передаваемых по сети и хранимых данных. При последующем чтении сообщений подписчиками они автоматически разжимаются использованным при записи кодеком, не требуя указания каких-либо параметров. Возможные значения: `RAW` - без сжатия (по умолчанию), `GZIP`, `ZSTD`.
`--consumers`, `-c`	Количество читателей. Значение по умолчанию: `1`.
`--consumer-prefix`	Префикс имени читателей. Значение по умолчанию: `workload-consumer`. Например, если количество читателей `--consumers` равно `2` и префикс `--consumer-prefix` равен `workload-consumer`, то будут использованы следующие имена читателей: `workload-consumer-0`, `workload-consumer-1`.
`--threads`, `-t`	Количество потоков читателя. Значение по умолчанию: `1`.
`--use-tx`	Использовать транзакции при записи сообщений. По умолчанию транзакции использоваться не будут.
`--tx-commit-interval`	Интервал коммита транзакций в миллисекундах. Коммит транзакции произойдёт, когда выполнится одно из условий: пройдёт заданное в параметре `--tx-commit-interval` количество времени или будет получено заданное в параметре `--tx-commit-messages` количество сообщений. Значение по умолчанию: `1000`.
`--tx-commit-messages`	Количество сообщений, необходимое для коммита транзакции. Коммит транзакции произойдёт, когда выполнится одно из условий: пройдёт заданное в параметре `--tx-commit-interval` количество времени или будет получено заданное в параметре `--tx-commit-messages` количество сообщений. Значение по умолчанию: `1 000 000`.

Пример команды чтения с помощью `2` читателей в `50` потоков и записи `100` потоков писателей с целевой скоростью `80` МБ/с и длительностью `10` секунд:

```
ydb --profile quickstart workload topic run full --producer-threads 100 --consumers 2 --consumer-threads 50 --byte-rate 80M
```

В процессе работы будет выводиться статистика по промежуточным временным окнам, а по окончании теста — итоговая статистика за все время работы:

```
Window  Write speed     Write time      Inflight        Lag             Lag time        Read speed      Full
time
#       msg/s   MB/s    percentile,ms   percentile,msg  percentile,msg  percentile,ms   msg/s   MB/s    perce
ntile,ms
1       0       0       0               0               0               0               0       0       0
2       1091    10      2143            8               2076            20607           40156   392     30941
3       1552    15      2991            12              7224            21887           41040   401     31886
4       1733    16      3711            15              10036           22783           38488   376     32577
5       1900    18      4319            15              10668           23551           34784   340     33372
6       2793    27      5247            21              9461            24575           33267   325     34893
7       2904    28      6015            22              12150           25727           34423   336     35507
8       2191    21      5087            21              12150           26623           29393   287     36407
9       1952    19      2543            10              7627            27391           33284   325     37814
10      1992    19      2655            9               10104           28671           29101   284     38797
Window  Write speed     Write time      Inflight        Lag             Lag time        Read speed      Full
time
#       msg/s   MB/s    percentile,ms   percentile,msg  percentile,msg  percentile,ms   msg/s   MB/s    perce
ntile,ms
Total   1814    17      5247            22              12150           28671           44827   438     40252
```

- `Window` — порядковый номер временного окна сбора статистики.
- `Write speed` — скорость записи сообщений, сообщений в секунду и МБ/с.
- `Write time` — процентиль времени записи сообщения в мс.
- `Inflight` — максимальное число сообщений, ожидающих подтверждения по всем партициям.
- `Lag` — максимальное число сообщений, ожидающих чтения, в окне сбора статистики. Учитываются сообщения по всем партициям.
- `Lag time` — процентиль времени задержки сообщений в мс.
- `Read` — Скорость чтения сообщений читателем, сообщений в секунду и МБ/с.
- `Full time` — процентиль времени полной обработки сообщения, от записи писателем до чтения читателем в мс.

Удаление топика

После завершения работы можно удалить тестовый топик. Общий вид команды для удаления топика:

```
ydb [global options...] workload topic clean [options...]
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.

Параметры подкоманды:

Имя параметра	Описание параметра
`--topic`	Имя топика. Значение по умолчанию: `workload-topic`.

Чтобы удалить тестовый топик `workload-topic`, выполните следующую команду:

```
ydb --profile quickstart workload topic clean
```

Transfer нагрузка

Запускает нагрузку в виде транзакций YDB, в которых участвуют топик и таблица. Данные вычитываются из топика и записываются в таблицу. Для имитации реальной нагрузки можно задавать различные входные параметры: число сообщений, размер сообщений, целевую скорость записи, число читателей и писателей, число партиций. В процессе работы на консоль выводятся результаты: количество записанных сообщений, скорость записи сообщений и т.д.

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Инициализация тестового окружения

Перед запуском нагрузки необходимо инициализировать тестовое окружение. Для этого можно использовать команду `ydb workload transfer topic-to-table init`. Она создаст топик и таблицу с нужными параметрами.

Общий вид команды:

```
ydb [global options...] workload transfer topic-to-table init [options...]
```

- `global options` — глобальные параметры.
- `options` - параметры подкоманды.

Посмотреть описание команды:

```
ydb workload transfer topic-to-table init --help
```

Параметры подкоманды:

Имя параметра	Описание параметра	Значение по умолчанию
`--topic`	Имя топика	`transfer-topic`
`--consumer-prefix`	Префикс имени читателей	`workload-consumer`
`--table`	Имя таблицы	`transfer-table`
`--consumers`	Количество читателей топика	`1`
`--topic-partitions`	Количество партиций топика	`128`
`--table-partitions`	Количество партиций таблицы	`128`

После выполнения подкоманды `init` будут созданы таблица, топик и читатели. Имена читателей создаются по правилу `${CONSUMER_PREFIX}-${INDEX}`. Значение `${INDEX}` — это целое число от 0 и до значения параметра `--consumers` минус 1.

Например, команда

```
ydb --profile quickstart workload transfer topic-to-table init --consumers 2 --topic-partitions 143 --table-p
artitions 237
```

создаст топик `transfer-topic` с 2 читателями и 143 партициями и таблицу `transfer-table` с 237 партициями. Имена читателей `workload-consumer-0` и `workload-consumer-1`.

Запуск нагрузочного теста

Тест моделирует нагрузку от приложения, которое получает сообщения из топика, обрабатывает их и записывает результаты обработки в таблицу БД.

Во время работы программы имитируются рабочие потоки двух видов:

- Входной поток: в топик пишутся сообщения в режиме без транзакций. Пользователь может управлять скоростью записи, размером сообщений, количеством писателей.
- Поток обработки: из топика читаются сообщения и записываются в таблицу с использованием транзакций YDB.

В рамках одной транзакции в потоке обработки выполняются следующие действия:

- пока не истёк период `--commit-period`, читаются сообщения из топика;
- по истечении периода, выполняется одна команда `UPSERT` над таблицей и команда `COMMIT` для фиксации транзакции.

Общий вид команды:

```
ydb [global options...] workload transfer topic-to-table run [options...]
```

- `global options` — глобальные параметры.
- `options` - параметры подкоманды.

Посмотреть описание команды:

```
ydb workload transfer topic-to-table run --help
```

Параметры подкоманды:

Имя параметра	Описание параметра	Значение по умолчанию
`--seconds`, `-s`	Продолжительность теста в секундах	`60`
`--window`, `-w`	Длительность окна сбора статистики в секундах	`1`
`--quiet`, `-q`	Выводить только итоговый результат теста	`0`
`--print-timestamp`	Печатать время вместе со статистикой каждого временного окна	`0`
`--percentile`	Процентиль в выводе статистики	`50`
`--warmup`	Время прогрева теста в секундах. В течение этого времени не вычисляется статистика, оно необходимо для устранения переходных процессов при старте	`5`
`--topic`	Имя топика	`transfer-topic`
`--consumer-prefix`	Префикс имени читателей	`workload-consumer`
`--table`	Имя таблицы	`transfer-table`
`--producer-threads`, `-p`	Количество потоков писателей	`1`
`--consumer-threads`, `-t`	Количество потоков читателей	`1`
`--consumers`, `-c`	Количество читателей	`1`
`--message-size`, `-m`	Размер сообщения в байтах. Возможно задание в КБ, МБ, ГБ путем добавления суффиксов `K`, `M`, `G` соответственно	`10240`
`--message-rate`	Целевая суммарная скорость записи. В сообщениях в секунду. Исключает использование параметра `--byte-rate`	`0`
`--byte-rate`	Целевая суммарная скорость записи. В байтах в секунду. Исключает использование параметра `--message-rate`. Возможно задание в КБ/с, МБ/с, ГБ/с путем добавления суффиксов `K`, `M`, `G`, соответственно	`0`
`--tx-commit-interval`	Период между вызовами `COMMIT` в транзакции. В миллисекундах	`1000`
`--tx-commit-messages`	Период между вызовами `COMMIT` в транзакции. В количестве сообщений	`1000000`
`--only-topic-in-tx`	В транзакциях принудительно участвуют только партиции топиков. Исключает использование параметра `--only-table-in-tx`	`0`
`--only-table-in-tx`	В транзакциях принудительно участвуют только шарды таблиц. Исключает использование параметра `--only-topic-in-tx`	`0`

Например, команда

```
ydb --profile quickstart workload transfer topic-to-table run
```

запустит тест продолжительностью 60 секунд. Данные за первые 5 секунд не будут учитываться в статистике работы. Пример вывода на консоль

```
Window  Write speed     Write time      Inflight        Read speed      Topic time      Select time     Upser
t time     Commit time
#       msg/s   MB/s    percentile,ms   percentile,msg  msg/s   MB/s    percentile,ms   percentile,ms   perce
ntile,ms   percentile,ms
1       0       0       0               0               0       0       0               0               0
0
2       0       0       0               0               0       0       0               0               0
0
3       0       0       0               0               0       0       0               0               0
0
4       0       0       0               0               0       0       0               0               0
0
5       0       0       0               0               0       0       0               0               0
0
6       103     1       1023            83              103     1       1025            0               0
0
7       103     1       999             78              103     1       1001            0               0
0
8       103     1       1003            93              103     1       1002            0               0
0
9       103     1       1003            88              103     1       1003            0               0
0
10      103     1       999             79              103     1       999             0               0
0
11      103     1       1119            89              0       0       0               0               0
0
12      103     1       1023            90              206     2       1028            90              223
695
13      103     1       975             84              103     1       976             0               0
0
14      103     1       1003            91              103     1       1006            0               0
```

```
0
15      103     1       1003            93              103     1       1005            0               0
0
16      103     1       1103            89              103     1       1100            0               0
0
17      103     1       1063            89              103     1       1061            0               0
0
...
```

- `Window` — порядковый номер временного окна сбора статистики.
- `Write speed` — скорость записи сообщений писателями. В сообщениях в секунду и в мегабайтах в секунду.
- `Write time` — процентиль времени записи сообщения в мс.
- `Inflight` — максимальное число сообщений, ожидающих подтверждения по всем партициям.
- `Lag` — максимальное число сообщений, ожидающих чтения, в окне сбора статистики. Учитываются сообщения по всем партициям.
- `Lag time` — процентиль времени задержки сообщений в мс.
- `Read speed` — скорость чтения сообщений читателями. В сообщениях в секунду и в мегабайтах в секунду.
- `Select time`, `Upsert time`, `Commit time` — процентиль времени выполнения операций Select, Upsert, Commit в мс.

Удаление тестового окружения

После завершения теста можно удалить тестовое окружение.

Общий вид команды:

```
ydb [global options...] workload transfer topic-to-table clean [options...]
```

- `global options` — глобальные параметры.
- `options` - параметры подкоманды.

Посмотреть описание команды:

```
ydb workload transfer topic-to-table clean --help
```

Параметры подкоманды:

Имя параметра	Описание параметра	Значение по умолчанию
`--topic`	Имя топика	`transfer-topic`
`--table`	Имя таблицы	`transfer-table`

Например, команда

```
ydb --profile quickstart workload transfer topic-to-table clean
```

удалит топик `transfer-topic`, его читателей и таблицу `transfer-table`.

TPC-C нагрузка

Нагрузка базируется на официальной спецификации TPC-C. Все запросы и схемы таблиц адаптированы под YDB.

TPC-C — это промышленный стандарт для оценки производительности систем обработки транзакций в реальном времени (OLTP). Он моделирует работу розничной компании с настраиваемым числом складов. Каждый склад включает 10 районов, в каждом районе — 3000 клиентов. На складах хранятся товары, клиенты размещают заказы с несколькими позициями. Компания отслеживает платежи, доставку, историю заказов и периодически проводит инвентаризацию.

В результате создаётся нагрузка из распределённых транзакций различных типов и сложности, выполняемых параллельно.

В TPC-C склад является основной единицей масштабирования: чем больше складов задано, тем больше объём данных и интенсивнее поток транзакций. Каждый склад содержит примерно 100 МБ данных. Максимальное число складов, которое способен обработать кластер YDB, зависит от аппаратных ресурсов, конфигурации и сетевых задержек между узлами. В качестве практического ориентира можно использовать соотношение — около 50 складов на одно вычислительное ядро CPU.

Пример использования TPC-C:

```
ydb workload tpcc --path tpcc/10wh init -w 10
ydb workload tpcc --path tpcc/10wh import -w 10
ydb workload tpcc --path tpcc/10wh run -w 10
```

По умолчанию, если команды `import` и `run` запускаются в интерактивном терминале, они отображают прогресс выполнения с помощью текстового пользовательского интерфейса (TUI).

Общие параметры команд

Все команды поддерживают общий параметр `--path`, задающий путь к каталогу в базе данных с таблицами бенчмарка:

```
ydb workload tpcc --path tpcc/10wh ...
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--path` или `-p`	Путь к каталогу с таблицами бенчмарка в базе данных.	`/`

Инициализация нагрузочного теста

Перед запуском бенчмарка создайте таблицы:

```
ydb workload tpcc --path tpcc/10wh init --warehouses 10
```

Подробнее о параметрах команды:

```
ydb workload tpcc init --help
```

Доступные параметры

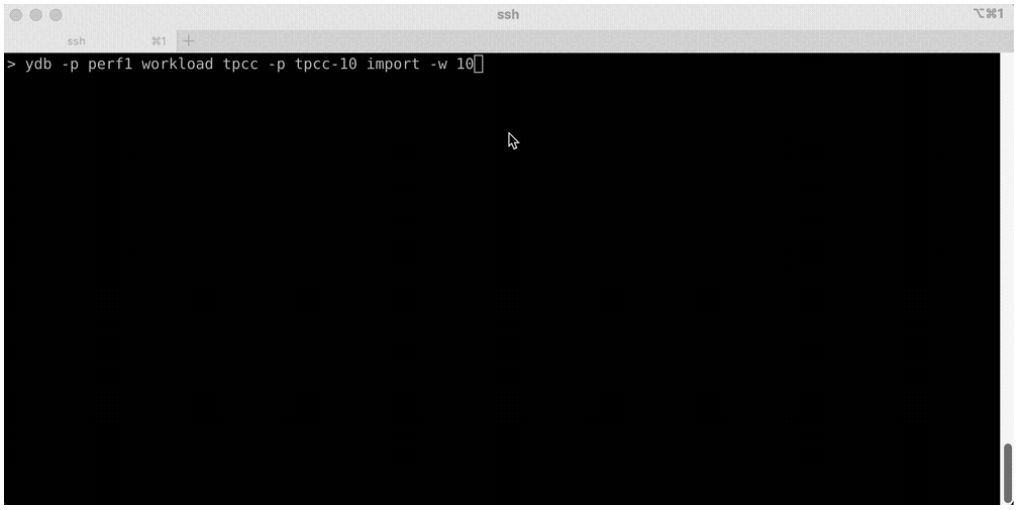
Имя	Описание	Значение по умолчанию
`--warehouses` или `-w`	Количество складов TPC-C.	10

Загрузка данных в таблицу

Данные будут сгенерированы и загружены в таблицы непосредственно ydb:

```
ydb workload tpcc --path tpcc/10wh import --warehouses 10
```

Пример использования:



Подробнее о параметрах команды:

```
ydb workload tpcc import --help
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--warehouses` или `-w`	Количество складов TPC-C.	10
`--compact`	Выполнить сжатие таблиц после загрузки данных.	
`--threads`	Количество потоков для загрузки данных TPC-C в базу данных.	auto
`--no-tui`	Отключить TUI (по умолчанию включен при интерактивном запуске).	

Оптимальное количество потоков загрузки зависит от размера и конфигурации кластера YDB. В качестве ориентира: для кластеров с несколькими сотнями ядер CPU разумно начать примерно с 50 потоков загрузки. При необходимости это число можно увеличить.

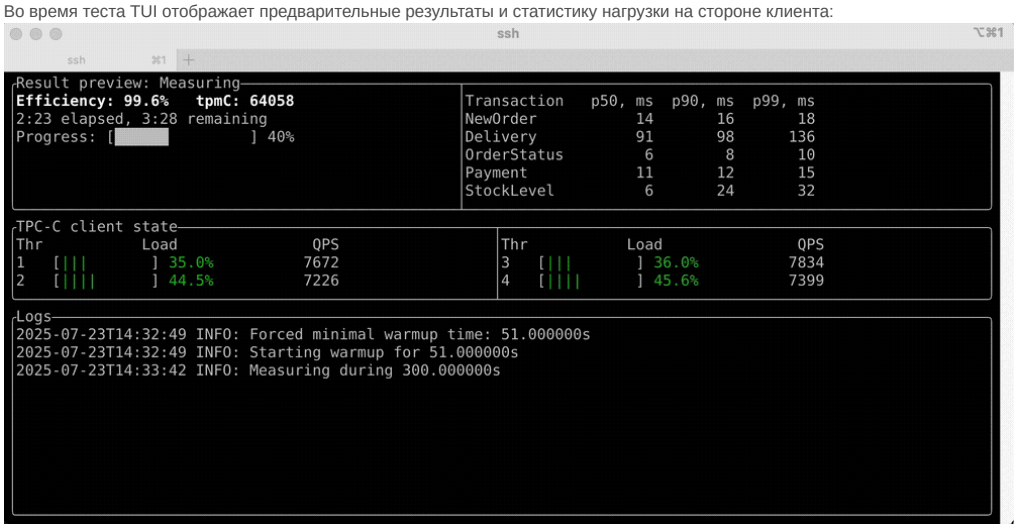
Однако на клиентской стороне рекомендуется ограничивать число потоков загрузки 50–75% от общего количества доступных ядер процессора.

Запуск нагрузочного теста

Запустите нагрузку:

```
ydb workload tpcc --path tpcc/10wh run --warehouses 10
```

Во время теста TUI отображает предварительные результаты и статистику нагрузки на стороне клиента:



Подробнее о параметрах команды:

```
ydb workload tpcc run --help
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--warehouses` или `-w`	Количество складов TPC-C.	10
`--warmup`	Время прогрева. Например: 10s, 5m, 1h.	auto
`--time` или `-t`	Время выполнения. Например: 10s, 5m, 1h.	2h
`--max-sessions` или `-m`	Мягкое ограничение на количество сессий БД.	auto
`--threads`	Количество потоков, выполняющих запросы.	auto
`--format` или `-f`	Формат вывода: 'Pretty', 'Json'.	Pretty
`--no-tui`	Отключить TUI (по умолчанию включен при интерактивном запуске).	

Оптимальное количество сессий зависит от размера и конфигурации кластера YDB. В качестве ориентира: умножьте общее количество ядер CPU, выделенных для вычислительных узлов YDB (dynnodes), на 5–10.

Мы рекомендуем начинать с коэффициента 5. Если во время выполнения бенчмарка ресурсы YDB используются недостаточно, можно увеличить число сессий.

Результаты

Результаты теста включают tpmC, эффективность и задержки по типам транзакций. Согласно спецификации:

> «Показателем производительности TPC-C является “деловая пропускная способность”, измеряемая количеством новых заказов, обработанных в минуту. Каждая единица нагрузки состоит из нескольких транзакций, имитирующих деловую активность, каждая из которых должна укладываться в определённое время отклика. Показатель измеряется в “транзакциях в минуту-C” (tpmC)».

Спецификация ограничивает теоретическую максимальную производительность одним складом: 12.86 tpmC. Чтобы увеличить общую нагрузку и итоговый tpmC, масштабируйте количество складов.

Формула расчёта эффективности:

Очистка данных теста

Удаление таблиц бенчмарка:

```
ydb workload tpcc --path tpcc/10wh clean
```

Команда не имеет параметров.

Проверка целостности данных

Команда `check` проверяет консистентность данных TPC-C после команды `import` или `run`. Её основное назначение — помощь в разработке, для конечных пользователей она не представляет особого интереса.

Пример:

```
ydb workload tpcc --path tpcc/10wh check
```

Подробнее о параметрах команды:

```
ydb workload tpcc check --help
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--warehouses` или `-w`	Количество складов TPC-C.	10
`--just-imported`	Включает дополнительные проверки. Используется только сразу после импорта, до запуска каких-либо тестов.	

TPC-H нагрузка

Нагрузка базируется на документе TPC-H документация, а запросы и схемы таблиц адаптированы под YDB.

Тест генерирует типичную рабочую нагрузку в области помощи в принятии решений.

Общие параметры команд

Все команды поддерживают общий параметр `--path`, который задает путь к каталогу с таблицами в базе данных:

```
ydb  workload tpch --path tpch/s1 ...
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--path` или `-p`	Путь к каталогу с таблицами.	`/`

Инициализация нагрузочного теста

Перед запуском теста создайте таблицу:

```
ydb workload tpch --path tpch/s1 init
```

Посмотрите описание команды для инициализации табилц:

```
ydb workload tpch init --help
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--store <значение>`	Тип хранилища таблиц. Возможные значения: `row`, `column`, `external-s3`.	`column`
`--external-s3-prefix <значение>`	Актуально только для внешних таблиц. Корневой путь к набору данных в S3-хранилище.	
`--external-s3-endpoint <значение>` или `-e <значение>`	Актуально только для внешних таблиц. Ссылка на S3-Bucket с данными.	
`--string`	Использовать для текстовых полей тип `String`.	`Utf8`
`--datetime`	Использовать для полей, связанных со временем типа `Date`, `Datetime` и `Timestamp`.	`Date32`, `Datetime64` и `Timestamp64`
`--partition-size`	Максимальный размер партиции в мегабайтах (AUTO_PARTITIONING_PARTITION_SIZE_MB) для строчных таблиц	2000
`--float-mode <значение>`	Какой тип данных использовать для полей вещественного типа. Возможные значения: `double` и `decimal`. `double` - использовать тип `Double`, `decimal` - `Decimal` с размерностью, заданной стандартом теста.	`double`
`--scale`	Задает процент от полного размера данных и нагрузки теста, который будет использоваться.	1

| `--clear` | Если по указанному пути таблица уже была создана, она будет удалена. | |

Загрузка данных в таблицу

Загрузите данные в таблицу. Данные будут сгенерированы непосредственно ydb:

```
ydb workload tpch --path tpch/s1 import generator --scale 1
```

Посмотрите описание команды для загрузки данных:

```
ydb workload tpch import --help
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--scale <значение>`	Масштаб данных. Обычно используются степени десяти. Поддерживается и дробный масштаб, хотя он и не описан в спецификации TPC-H.	
`--tables <значение>`	Список таблиц для генерации, разделенный запятыми. Доступные таблицы: `customer`, `nation`, `order_line`, `part_psupp`, `region`, `supplier`.	Все таблицы.

`--proccess-count <значение>` или `-C <значение>`	Генерация данных может быть разбита на несколько процессов, этот параметр задает количество процессов.	1
`--proccess-index <значение>` или `-i <значение>`	Генерация данных может быть разбита на несколько процессов, этот параметр задает номер процесса.	0
`--state <путь>`	Путь к файлу состояния загрузки. Если загрузка была прервана по какой-то причине, при новом запуске загрузка будет продолжена с того же места.	
`--clear-state`	Актуально, если задан параметр `--state`. Очистить файл состояния и начать загрузку сначала.	
`--dry-run`	Не выполнять загрузку данных в базу данных, только подготовить данные и вывести об этом сообщение.	

Общие параметры команды import

Имя	Описание	Значение по умолчанию
`--upload-threads <значение>` или `-t <значение>`	Количество потоков исполнения для подготовки данных.	Рассчитывается автоматически и равно числу доступных ядер на клиентской машине.
`--bulk-size <значение>`	Размер порции для отправки данных в строках.	10000
`--max-in-flight <значение>`	Максимальное количество порций данных, одновременно находящихся в обработке.	128
`--file-output-path <значение>` или `-f <путь>`	Если эта опция установлена, данные не будут загружены в базу данных, а будут сохранены в каталоге <путь>.	

Запуск нагрузочного теста

Запустите нагрузку:

```
ydb workload tpch --path tpch/s1 run
```

В течение теста на экран выводится статистика по нагрузке для каждого запроса.

Посмотрите описание команды для запуска нагрузки:

```
ydb workload tpch run --help
```

Общие параметры для всех видов нагрузки

Имя	Описание	Значение по умолчанию
`--dry-run`	Не выполнять инициализационные запросы, а только вывести их текст.	
`--check-canonical` или `-c`	Использовать специальную версию запросов (они имеют детерминированные ответы) и сверять результаты с каноническими.	
`--output <значение>`	Имя файла, в котором будут сохранены результаты выполнения запросов.	`results.out`
`--iterations <значение>`	Количество выполнений каждого из запросов нагрузки.	`1`
`--json <имя>`	Имя файла, в котором будет сохранена статистика выполнения запросов в формате `json`.	Файл не сохраняется
`--ministat <имя>`	Имя файла, в котором будет сохранена статистика выполнения запросов в формате `ministat`.	Файл не сохраняется
`--csv <имя>`	Имя файла для сохранения CSV-версии таблицы с результатами.	Файл не сохраняется
`--plan <имя>`	Имя файла для сохранения плана запроса. Если задано, то сохраняются файлы `<имя>.<номер запроса>.explain` и `<имя>.<номер запроса>.<номер итерации>` с планами в нескольких форматах: `ast`, `json`, `svg` и `table`.	Планы не сохраняются.
`--query-prefix <префикс>`	Префикс запроса. Каждый префикс будет добавлен отдельной строкой в начало каждого запроса. Если нужно указать несколько префиксов, используйте параметр несколько раз.	По умолчанию не задан
`--retries`	Количество повторных попыток выполнения каждого запроса	`0`
`--include`	Имена, номера или диапазоны номеров запросов, которые нужно выполнить в рамках нагрузки. Указываются через запятую, например: `1,2,4-6`.	Все запросы
`--exclude`	Имена, номера или диапазоны номеров запросов, которые нужно исключить из нагрузки. Указываются через запятую, например: `1,2,4-6`.	

`--verbose` или `-v`	Выводить больше информации на экран в процессе выполнения запросов.	
`--global-timeout <значение>`	Общий таймаут на выполнение всех запросов. Задаётся в текстовом формате, например: `0.5s`, `1m`, `100us` и т.д.	Отсутствует. Время не ограничено.
`--request-timeout <значение>`	Таймаут на выполнение каждой итерации каждого запроса. Задаётся в текстовом формате, например: `0.5s`, `1m`, `100us` и т.д.	Отсутствует. Время не ограничено.
`--threads <значение>` или `-t <значение>`	Количество потоков, генерирующих нагрузку. Ноль означает, что запросы будут выполняться в основном потоке; при ином значении запросы будут перемешаны.	`0`
`--stats <значение>`	Режим сбора расширенной статистики выполнения. Возможные значения: `full`, `profile`.	`full`

Опции, специфичные для TPC-H

Имя	Описание	Значение по умолчанию
`--syntax <значение>`	Какой вариант синтаксиса запросов использовать: `yql`. Про работу с синтаксисом YQL можно прочитать здесь.	`yql`
`--float_mode <значение>`	Режим работы с числами с плавающей точкой. Может быть `float`, `decimal` или `decimal_ydb`. Если установлено значение `float` - будет использоваться float, `decimal` означает, что будет использоваться decimal с каноническим размером, указанным в спецификации TPC-H(`Decimal(12, 2)`), а `decimal_ydb` означает, что все float будут преобразованы в `Decimal(22, 9)`. Подробнее про тип Decimal можно посмотреть в документации.	`float`
`--scale <значение>`	Scale factor. См. спецификацию TPC-H п.4.1.3. Поддерживаются и дробные значения, хотя они и не описаны в спецификации TPC-H. Для масштабов `0.1`, `0.2`, `0.5`, `1`, `10`, `100`, `1000`, `10000` заданы каноничные ответы (см. описание опции `--check-canonical`).	1

Очистка данных теста

Запустите очистку:

```
ydb workload tpch --path tpch/s1 clean
```

Команда не имеет параметров.

TPC-DS нагрузка

Нагрузка базируется на документе TPC-DS документация, а запросы и схемы таблиц адаптированы под YDB.

Тест генерирует типичную рабочую нагрузку в области помощи в принятии решений.

Общие параметры команд

Все команды поддерживают общий параметр `--path`, который задает путь к каталогу с таблицами в базе данных:

```
ydb  workload tpcds --path tpcds/s1 ...
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--path` или `-p`	Путь к каталогу с таблицами.	`/`

Инициализация нагрузочного теста

Перед запуском теста создайте таблицу:

```
ydb workload tpcds --path tpcds/s1 init
```

Посмотрите описание команды для запуска нагрузки:

```
ydb workload tpcds init --help
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--store <значение>`	Тип хранилища таблиц. Возможные значения: `row`, `column`, `external-s3`.	`column`
`--external-s3-prefix <значение>`	Актуально только для внешних таблиц. Корневой путь к набору данных в S3-хранилище.	
`--external-s3-endpoint <значение>` или `-e <значение>`	Актуально только для внешних таблиц. Ссылка на S3-Bucket с данными.	
`--string`	Использовать для текстовых полей тип `String`.	`Utf8`
`--datetime`	Использовать для полей, связанных со временем типа `Date`, `Datetime` и `Timestamp`.	`Date32`, `Datetime64` и `Timestamp64`
`--partition-size`	Максимальный размер партиции в мегабайтах (AUTO_PARTITIONING_PARTITION_SIZE_MB) для строчных таблиц	2000
`--float-mode <значение>`	Какой тип данных использовать для полей вещественного типа. Возможные значения: `double` и `decimal`. `double` - использовать тип `Double`, `decimal` - `Decimal` с размерностью, заданной стандартом теста.	`double`
`--scale`	Задает процент от полного размера данных и нагрузки теста, который будет использоваться.	1

| `--clear` | Если по указанному пути таблица уже была создана, она будет удалена. | |

Загрузка данных в таблицу

Загрузите данные в таблицу. Данные будут сгенерированы непосредственно ydb:

```
ydb workload tpcds --path tpcds/s1 import generator --scale 1
```

Посмотрите описание команды для загрузки данных:

```
ydb workload tpcds import --help
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--scale <значение>`	Масштаб данных. Обычно используются степени десяти. Так же поддерживается и дробный масштаб, неописанный в спецификации TPC-DS. Он может быть полезен для быстрого тестирования небольших баз YDB. Примеры: `0.1`, `0.3`	

`--tables <значение>`	Список таблиц для генерации, разделенный запятыми. Доступные таблицы: `customer`, `nation`, `order_line`, `part_psupp`, `region`, `supplier`.	Все таблицы
`--proccess-count <значение>` или `-C <значение>`	Генерация данных может быть разбита на несколько процессов, этот параметр задает количество процессов.	1
`--proccess-index <значение>` или `-i <значение>`	Генерация данных может быть разбита на несколько процессов, этот параметр задает номер процесса.	0.
`--state <путь>`	Путь к файлу состояния загрузки. Если загрузка была прервана по какой-то причине, при новом запуске загрузка будет продолжена с того же места.	
`--clear-state`	Актуально, если задан параметр `--state`. Очистить файл состояния и начать загрузку сначала.	
`--dry-run`	Не выполнять загрузку данных в базу данных, только подготовить данные и вывести об этом сообщение.	

Общие параметры команды import

Имя	Описание	Значение по умолчанию
`--upload-threads <значение>` или `-t <значение>`	Количество потоков исполнения для подготовки данных.	Рассчитывается автоматически и равно числу доступных ядер на клиентской машине.
`--bulk-size <значение>`	Размер порции для отправки данных в строках.	10000
`--max-in-flight <значение>`	Максимальное количество порций данных, одновременно находящихся в обработке.	128
`--file-output-path <значение>` или `-f <путь>`	Если эта опция установлена, данные не будут загружены в базу данных, а будут сохранены в каталоге <путь>.	

Запуск нагрузочного теста

Запустите нагрузку:

```
ydb workload tpcds --path tpcds/s1 run
```

В течение теста на экран выводится статистика по нагрузке для каждого запроса.

Посмотрите описание команды для запуска нагрузки:

```
ydb workload tpcds run --help
```

Общие параметры для всех видов нагрузки

Имя	Описание	Значение по умолчанию
`--dry-run`	Не выполнять инициализационные запросы, а только вывести их текст.	
`--check-canonical` или `-c`	Использовать специальную версию запросов (они имеют детерминированные ответы) и сверять результаты с каноническими.	
`--output <значение>`	Имя файла, в котором будут сохранены результаты выполнения запросов.	`results.out`
`--iterations <значение>`	Количество выполнений каждого из запросов нагрузки.	`1`
`--json <имя>`	Имя файла, в котором будет сохранена статистика выполнения запросов в формате `json`.	Файл не сохраняется
`--ministat <имя>`	Имя файла, в котором будет сохранена статистика выполнения запросов в формате `ministat`.	Файл не сохраняется
`--csv <имя>`	Имя файла для сохранения CSV-версии таблицы с результатами.	Файл не сохраняется
`--plan <имя>`	Имя файла для сохранения плана запроса. Если задано, то сохраняются файлы `<имя>.<номер запроса>.explain` и `<имя>.<номер запроса>.<номер итерации>` с планами в нескольких форматах: `ast`, `json`, `svg` и `table`.	Планы не сохраняются.
`--query-prefix <префикс>`	Префикс запроса. Каждый префикс будет добавлен отдельной строкой в начало каждого запроса. Если нужно указать несколько префиксов, используйте параметр несколько раз.	По умолчанию не задан
`--retries`	Количество повторных попыток выполнения каждого запроса	`0`
`--include`	Имена, номера или диапазоны номеров запросов, которые нужно выполнить в рамках нагрузки. Указываются через запятую, например: `1,2,4-6`.	Все запросы

`--exclude`	Имена, номера или диапазоны номеров запросов, которые нужно исключить из нагрузки. Указываются через запятую, например: `1,2,4-6`.	
`--verbose` или `-v`	Выводить больше информации на экран в процессе выполнения запросов.	
`--global-timeout <значение>`	Общий таймаут на выполнение всех запросов. Задаётся в текстовом формате, например: `0.5s`, `1m`, `100us` и т.д.	Отсутствует. Время не ограничено.
`--request-timeout <значение>`	Таймаут на выполнение каждой итерации каждого запроса. Задаётся в текстовом формате, например: `0.5s`, `1m`, `100us` и т.д.	Отсутствует. Время не ограничено.
`--threads <значение>` или `-t <значение>`	Количество потоков, генерирующих нагрузку. Ноль означает, что запросы будут выполняться в основном потоке; при ином значении запросы будут перемешаны.	`0`
`--stats <значение>`	Режим сбора расширенной статистики выполнения. Возможные значения: `full`, `profile`.	`full`

Опции, специфичные для TPC-DS

Имя	Описание	Значение по умолчанию
`--syntax <значение>`	Какой вариант синтаксиса запросов использовать: `yql`. Про работу с синтаксисом YQL можно прочитать здесь.	`yql`
`--float_mode <значение>`	Режим работы с числами с плавающей точкой. Может быть `float`, `decimal` или `decimal_ydb`. Если установлено значение `float`, будет использоваться тип float; `decimal` означает использование Decimal с каноническим размером, указанным в спецификации TPC-DS (различается для разных полей и таблиц, см. спецификацию); `decimal_ydb` означает преобразование всех float в Decimal(22, 9). Подробнее про тип Decimal см. в документации.	`float`
`--scale <значение>`	Scale factor. См. спецификацию TPC-DS, глава 3. Используется в запросах TPC-DS. Так же поддерживаются и дробные значения, неописанные в спецификации TPC-DS, например `0.1`, `0.3`. Для масштабов `1`, `10`, `100`, `1000` заданы канонические ответы (см. описание опции `--check-canonical`).	1

Очистка данных теста

Запустите очистку:

```
ydb workload tpcds --path tpcds/s1 clean
```

Команда не имеет параметров.

Пользовательская нагрузка

Позволяет провести нагрузочное тестирование по заданному пользовательскому сценарию.

Сценарий тестирования представляет собой директорию, называемую `suite`, которая содержит набор поддиректорий: `init`, `import` и `run`, соответствующих командам `ydb workload query`.

- Директория `init` может иметь произвольную структуру и содержит SQL-запросы создания и настройки объектов в базе данных, таких как таблицы или индексы.
- Директория `import` содержит данные для загрузки в таблицы. Каждой таблице соответствует поддиректория в директории `import`.
- Директория `run` может иметь произвольную структуру и содержит SQL-запросы для нагрузочного тестирования.

Пример директории `suite` на GitHub.

Пользователь указывает путь к директории `suite` с помощью опции `--suite-path` в каждой команде.

Общий синтаксис

Общий синтаксис вызова команд пользовательской нагрузки:

```
ydb [global options...] workload query [--path <путь/в/бд>] <команда> [options...]
```

- `global options` — глобальные параметры.
- `--path` — путь в базе данных для объектов нагрузочного тестирования.
- `<команда>` — одна из команд: `init`, `import`, `run`, `clean`.
- `options` — параметры конкретной команды.

Общие параметры команд

Все команды поддерживают общий параметр `--path`, который задает путь в базе данных для объектов нагрузочного тестирования. Все создаваемые объекты будут созданы по этому пути. Также путь будет учтен при выполнении теста и очистке.

Доступные параметры

Имя	Описание	Значение по умолчанию
`--path` или `-p`	Путь в базе данных для объектов нагрузочного тестирования.	`/`

Инициализация нагрузочного теста

Перед запуском теста создайте таблицы и другие объекты базы данных:

```
ydb workload query --path user_path init --suite-path ~/user-suite
```

Команда `init` действует следующим образом:

- Если задан `suite` и в нем присутствует директория `init`, производится сканирование директории `init` с учетом её вложенных директорий поиском в глубину с лексиографическим упорядочиванием файлов и поддиректорий на каждом уровне.
- Выбираются файлы с расширениями `sql` и `yql`.
- Содержимое каждого файла считывается и последовательно исполняется как SQL-запрос.
- После этого выполняются запросы, заданные напрямую из командной строки с помощью параметра `--query`.
- Если какой-либо из запросов приводит к ошибке, то выполнение команды прекращается.

Предполагается, что на этом этапе выполняются относительно "лёгкие" запросы, например, создание таблиц и индексов. Не рекомендуется включать в состав файлов в директории `init` скрипты с наполнением таблиц данными. Эти операции следует перенести на фазу `import`.

В запросах могут быть использованы макросы:

- `{db}` — Абсолютный путь в базе данных к директории тестирования. Представляет собой комбинацию значений опций `--database` и `--path`.

Посмотрите описание команды для инициализации таблиц:

```
ydb workload query init --help
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--suite-path <путь>`	Путь к `suite`-директории.	
`--query <запрос>` или `-q <запрос>`	DDL-запрос для выполнения. Может быть использован несколько раз.	
`--clear`	Если по указанному пути создаваемая таблица уже существует, она будет предварительно удалена.	
`--dry-run`	Не выполнять инициализационные запросы, а только распечатать их.	

Загрузка данных в таблицу

Загрузите данные в таблицы:

```
ydb workload query --path user_path import --suite-path ~/user-suite
```

Команда `import` действует следующим образом:

- Если задан `suite` и в нем присутствует директория `import`, она выполняет последовательный обход её поддиректорий.
- Предполагается, что каждая поддиректория соответствует таблице в базе данных. Путь к поддиректории от директории `suite` совпадает с путем к таблице от корня базы данных.
- В поддиректории используются файлы в поддерживаемых форматах данных, а именно `csv`, `tsv`, `csv.gz`, `tsv.gz` и `parquet`.
- Выполняется загрузка данных из файлов в соответствующие таблицы. Загрузка идет во много потоков, причем параллелится как загрузка разных файлов, так и частей одного файла. Порядок загрузки не гарантируется.

Процесс загрузки может продолжаться длительное время и быть по какой-то причине прерван. Для того чтобы иметь возможность продолжить загрузку с момента остановки, может быть применен механизм состояния. При помощи опции `--state` можно указать путь к файлу состояния. Если файл состояния существует, то он будет использован для продолжения загрузки. Если файла состояния нет, то он будет создан в процессе загрузки. Также он может быть очищен с помощью опции `--clear-state`.

Посмотрите описание команды для загрузки данных:

```
ydb workload query import --help
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--suite-path <путь>`	Путь к `suite`-директории.	
`--state <путь>`	Путь к файлу состояния загрузки. Если загрузка была прервана по какой-то причине, при новом запуске загрузка будет продолжена с того же места.	
`--clear-state`	Актуально, если задан параметр `--state`. Очистить файл состояния и начать загрузку сначала.	
`--dry-run`	Не выполнять загрузку данных в базу данных, только подготовить данные и вывести об этом сообщение.	

Общие параметры команды import

Имя	Описание	Значение по умолчанию
`--upload-threads <значение>` или `-t <значение>`	Количество потоков исполнения для подготовки данных.	Рассчитывается автоматически и равно числу доступных ядер на клиентской машине.
`--bulk-size <значение>`	Размер порции для отправки данных в строках.	10000
`--max-in-flight <значение>`	Максимальное количество порций данных, одновременно находящихся в обработке.	128
`--file-output-path <значение>` или `-f <путь>`	Если эта опция установлена, данные не будут загружены в базу данных, а будут сохранены в каталоге <путь>.	

Запуск нагрузочного теста

Запустите нагрузку:

```
ydb workload query --path user_path run --suite-path ~/user-suite
```

Команда `run` действует следующим образом:

- Если в `suite` присутствует директория `run`, команда выполняет обход её и всех её вложенных директорий поиском в глубину, на каждом уровне упорядочивая файлы и поддиректории по алфавиту. В случае отсутствия директории `run`, выполнение прекращается.
- Выбирает файлы с расширениями `sql` и `yql` и включает их в список SQL-запросов.
- После этого добавляет в список запросы, заданные напрямую из командной строки с помощью параметра `--query`.
- В зависимости от опций либо выполняет все запросы последовательно, каждый запрос заданное число раз, либо параллельно в несколько потоков с перемешиванием запросов.
- Ошибки в запросах будут учтены в статистике, но не приводят к остановке тестирования.

В течение теста на экран выводится статистика по нагрузке для каждого запроса и агрегированная статистика по всем запросам.

Посмотрите описание команды для запуска нагрузки:

```
ydb workload query run --help
```

Общие параметры для всех видов нагрузки

Имя	Описание	Значение по умолчанию
`--dry-run`	Не выполнять инициализационные запросы, а только вывести их текст.	
`--check-canonical` или `-c`	Использовать специальную версию запросов (они имеют детерминированные ответы) и сверять результаты с каноническими.	
`--output <значение>`	Имя файла, в котором будут сохранены результаты выполнения запросов.	`results.out`
`--iterations <значение>`	Количество выполнений каждого из запросов нагрузки.	`1`
`--json <имя>`	Имя файла, в котором будет сохранена статистика выполнения запросов в формате `json`.	Файл не сохраняется
`--ministat <имя>`	Имя файла, в котором будет сохранена статистика выполнения запросов в формате `ministat`.	Файл не сохраняется
`--csv <имя>`	Имя файла для сохранения CSV-версии таблицы с результатами.	Файл не сохраняется
`--plan <имя>`	Имя файла для сохранения плана запроса. Если задано, то сохраняются файлы `<имя>.<номер запроса>.explain` и `<имя>.<номер запроса>.<номер итерации>` с планами в нескольких форматах: `ast`, `json`, `svg` и `table`.	Планы не сохраняются.
`--query-prefix <префикс>`	Префикс запроса. Каждый префикс будет добавлен отдельной строкой в начало каждого запроса. Если нужно указать несколько префиксов, используйте параметр несколько раз.	По умолчанию не задан
`--retries`	Количество повторных попыток выполнения каждого запроса	`0`
`--include`	Имена, номера или диапазоны номеров запросов, которые нужно выполнить в рамках нагрузки. Указываются через запятую, например: `1,2,4-6`.	Все запросы
`--exclude`	Имена, номера или диапазоны номеров запросов, которые нужно исключить из нагрузки. Указываются через запятую, например: `1,2,4-6`.	
`--verbose` или `-v`	Выводить больше информации на экран в процессе выполнения запросов.	
`--global-timeout <значение>`	Общий таймаут на выполнение всех запросов. Задаётся в текстовом формате, например: `0.5s`, `1m`, `100us` и т.д.	Отсутствует. Время не ограничено.
`--request-timeout <значение>`	Таймаут на выполнение каждой итерации каждого запроса. Задаётся в текстовом формате, например: `0.5s`, `1m`, `100us` и т.д.	Отсутствует. Время не ограничено.
`--threads <значение>` или `-t <значение>`	Количество потоков, генерирующих нагрузку. Ноль означает, что запросы будут выполняться в основном потоке; при ином значении запросы будут перемешаны.	`0`
`--stats <значение>`	Режим сбора расширенной статистики выполнения. Возможные значения: `full`, `profile`.	`full`

Опции, специфичные для Query

Имя	Описание	Значение по умолчанию
`--suite-path <путь>`	Путь к `suite`-директории.	
`--query <запрос>` или `-q <запрос>`	Запрос для выполнения. Может быть использован несколько раз.	

Канонические результаты

Для каждого запроса из файла может быть задан канонический (ожидаемый) результат. Если ответ на запрос не соответствует каноническому, это будет отражено в статистике и в выводе команды. Также будут приведены отличия.

Канонический результат задается с помощью файла с тем же именем и дополнительным расширением `.result`. Это файлы CSV с заголовками и некоторым дополнительным синтаксисом:

- Если запрос имеет более одного набора результатов, файл результатов должен содержать соответствующее количество наборов данных, разделенных пустыми строками.

  Например, запрос

  ```
  SELECT COUNT(*) AS count FROM table_1;
  SELECT MIN(col1) AS min, MAX(col1) AS max FROM table_2;
  ```

  Может иметь следующий файл результатов:

```
count
10

min,max
4,5
```

- Последняя строка может быть задана как `...`, что означает, что результат запроса может иметь больше строк, чем в каноническом результате, но первые строки результата должны соответствовать каноническим.
- По умолчанию числа с плавающей точкой сравниваются с относительной точностью `1e-3` процента, но в каноническом результате вы можете указать любую абсолютную или относительную точность для каждого значения, например: `1.5+-0.01`, `2.4e+10+-1%`.

Для запросов, заданных опцией `--query`, канонический результат не может быть задан.

Канонический результат не будет использоваться, если не установлен флаг `--check-canonical`.

Очистка данных теста

Запустите очистку:

```
ydb workload query --path user_path clean
```

Выполняет рекурсивное удаление объектов базы данных по указанному в параметре `--path` пути.

Команда не имеет параметров.

Пример запуска пользовательского теста

Скачайте пример из репозитория

```
git clone https://github.com/ydb-platform/ydb.git
cd ydb/ydb/tests/functional/tpc/data/e1
```

Запустите инициализацию

```
ydb workload query --path test_path init --suite-path .
```

Результат:

```
Init tables ...
Init tables ...Ok
```

Запустите импорт данных

```
ydb workload query --path test_path import --suite-path .
```

Результат:

```
Fill table test_table_1...
Fill table test_table_1 OK 0 / 0 (0.353816s)
Fill table test_table_2...

Fill table test_table_2 OK 0 / 0 (0.102385s)
```

Запустите нагрузочный тест

```
ydb workload query --path user_path run --suite-path e1
```

Результат:

```
first_query_set.1.sql:
    iteration 0:    ok    0.131573s seconds
first_query_set.2.yql:
    iteration 0:    ok    0.089327s seconds
second_query_set.join.sql:
    iteration 0:    ok    0.145536s seconds

Results for 1 iterations
┌───────────────────────────┬──────────┬─────────┬─────────┬─────────┬─────────┬───────────┬─────────┬───────
──┬─────────┬─────────┬────────────────┬────────────────┬────────────────┬───────────┬──────────────┬────────
────┬──────────┐
│ Query #                   │ ColdTime │ Min     │ Max     │ Mean    │ Median  │ UnixBench │ Std     │ RttMin
│ RttMax  │ RttAvg  │ CompilationMin │ CompilationMax │ CompilationAvg │ GrossTime │ SuccessCount │ FailsCoun
t │ DiffsCount │
├───────────────────────────┼──────────┼─────────┼─────────┼─────────┼─────────┼───────────┼─────────┼───────
──┼─────────┼─────────┼────────────────┼────────────────┼────────────────┼───────────┼──────────────┼────────
────┼──────────┤
│ first_query_set.1.sql     │   0.086  │   0.086 │   0.086 │   0.086 │   0.086 │   0.086   │   0.000 │   0.04
5 │   0.045 │   0.045 │   0.041        │   0.041        │   0.041        │   0.132   │ 1            │
│            │
├───────────────────────────┼──────────┼─────────┼─────────┼─────────┼─────────┼───────────┼─────────┼───────
──┼─────────┼─────────┼────────────────┼────────────────┼────────────────┼───────────┼──────────────┼────────
```

```
──┼──────────┤
| first_query_set.2.yql     |   0.081  |   0.081 |   0.081 |   0.081 |   0.080 |   0.081   |   0.001 |   0.00
8 |   0.008 |   0.008 |   0.039        |   0.039        |   0.039        |   0.089   | 1           |
|           |
├──────────────────────────┼──────────┼─────────┼─────────┼─────────┼─────────┼───────────┼─────────┼──────
──┼────────┼─────────┼────────────────┼────────────────┼────────────────┼───────────┼─────────────┼──────
──┼──────────┤
| second_query_set.join.sql |   0.131  |   0.131 |   0.131 |   0.131 |   0.131 |   0.131   |   0.000 |   0.01
4 |   0.014 |   0.014 |   0.082        |   0.082        |   0.082        |   0.146   | 1           |
|           |
├──────────────────────────┼──────────┼─────────┼─────────┼─────────┼─────────┼───────────┼─────────┼──────
──┼────────┼─────────┼────────────────┼────────────────┼────────────────┼───────────┼─────────────┼──────
──┼──────────┤
| Sum                       |   0.299  |   0.299 |   0.299 |   0.299 |   0.297 |   0.299   |   0.000 |   0.06
8 |   0.068 |   0.068 |   0.161        |   0.161        |   0.162        |   0.367   | 3           |
|           |
├──────────────────────────┼──────────┼─────────┼─────────┼─────────┼─────────┼───────────┼─────────┼──────
──┼────────┼─────────┼────────────────┼────────────────┼────────────────┼───────────┼─────────────┼──────
──┼──────────┤
| Avg                       |   0.100  |   0.100 |   0.100 |   0.100 |   0.099 |   0.100   |   0.000 |   0.02
3 |   0.023 |   0.023 |   0.054        |   0.054        |   0.054        |   0.122   | 3           |
|           |
├──────────────────────────┼──────────┼─────────┼─────────┼─────────┼─────────┼───────────┼─────────┼──────
──┼────────┼─────────┼────────────────┼────────────────┼────────────────┼───────────┼─────────────┼──────
──┼──────────┤
| GAvg                      |   0.097  |   0.097 |   0.097 |   0.097 |   0.097 |   0.097   |   0.000 |   0.01
8 |   0.018 |   0.001 |   0.051        |   0.051        |   0.001        |   0.000   | 3           |
|           |
└──────────────────────────┴──────────┴─────────┴─────────┴─────────┴─────────┴───────────┴─────────┴──────
──┴────────┴─────────┴────────────────┴────────────────┴────────────────┴───────────┴─────────────┴──────
──┴──────────┘

Results saved to results.out
```

Очистите таблицы

```
ydb workload query --path user_path clean
```

Fulltext нагрузка

Позволяет тестировать производительность полнотекстового поиска YDB на документных наборах данных. Поддерживает как реальные датасеты (например, MS MARCO), так и синтетически сгенерированный текст при помощи цепи Маркова.

Марковская модель считает, что каждое следующее слово зависит от одного или нескольких предыдущих. Количество предыдущих слов, на основании которых определяется следующее, называется порядком модели. Сначала мы используем обычный генератор случайных чисел с равномерным распределением, чтобы определить длину текста, который будем генерировать. Далее при помощи марковской модели генерируем слово за словом, пока не наберём нужное количество слов. Такое построение данных необходимо, чтобы текст был похож на какой-то реальный человеческий текст, а не представлял собой просто беспорядочный набор букв. Марковская модель может использоваться для:

- генерации текстов документов, которые хранятся в базе данных;
- генерации поисковых запросов (select);
- генерации запросов на запись (upsert).

Структура команды

```
ydb [global options...] workload fulltext [options...] <subcommand>
```

Подкоманды:

```
fulltext            YDB fulltext workload
├─ init               Инициализация таблиц для нагрузки
├─ import             Загрузка данных и построение полнотекстового индекса
|   ├─ files           Импорт данных из файлов
|   └─ generator       Генерация случайного текста с помощью модели цепи Маркова
├─ run                Запуск нагрузки
|   ├─ select          Поиск документов с помощью полнотекстовых запросов
|   └─ upsert          Вставка или обновление документов в таблице
├─ clean              Удаление таблиц, созданных для нагрузочного тестирования
└─ model              Построение модели цепи Маркова из текстового датасета
```

Общие параметры команд

Все команды поддерживают следующий параметр:

Имя	Описание	Значение по умолчанию
`--path` или `-p`	Путь в базе данных, по которому будут созданы таблицы нагрузки.	`fulltext_workload`

Инициализация нагрузки

Создайте таблицы для нагрузочного тестирования:

```
ydb workload fulltext init
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--min-partitions <значение>`	Минимальное количество партиций таблицы.	`40`
`--partition-size <значение>`	Целевой размер партиции, в МБ.	`2000`
`--auto-partition <значение>`	Включить автопартицирование по нагрузке (`1` — включено, `0` — выключено).	`1`
`--clear`	Удалить и пересоздать таблицу, если она уже существует.	

Загрузка данных

После инициализации загрузите данные в таблицы и постройте полнотекстовый индекс. Доступны две подкоманды: `files` — для импорта из готового датасета, и `generator` — для генерации синтетических данных.

После завершения импорта автоматически строится полнотекстовый индекс с именем `index` по столбцу `text`.

Импорт из файлов

Импорт документов из файлов (CSV, TSV или Parquet, возможно в сжатом формате gzip). Датасет должен содержать столбцы `id` и `text`. Для хранения колонки `id` внутри таблицы используется тип Uint64, для хранения колонки `text` используется тип String.

Пример запуска:

```
ydb workload fulltext import files
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--input <путь>` или `-i <путь>`	Путь к файлу или директории с датасетом. Поддерживаемые форматы: CSV/TSV (с опциональным gzip-сжатием), Parquet. Импортируются только столбцы `id` и `text`.	Обязательный

Общие параметры команды import

Имя	Описание	Значение по умолчанию
`--upload-threads <значение>` или `-t <значение>`	Количество потоков исполнения для подготовки данных.	Рассчитывается автоматически и равно числу доступных ядер на клиентской машине.
`--bulk-size <значение>`	Размер порции для отправки данных в строках.	10000
`--max-in-flight <значение>`	Максимальное количество порций данных, одновременно находящихся в обработке.	128
`--file-output-path <значение>` или `-f <путь>`	Если эта опция установлена, данные не будут загружены в базу данных, а будут сохранены в каталоге <путь>.	

Генерация синтетических данных

Генерация случайного текста с помощью модели цепи Маркова и загрузка его в таблицу. Перед использованием необходимо построить модель или загрузить готовую.

- Чтобы загрузить уже готовую модель:

```
wget https://storage.yandexcloud.net/ydb-public/markov_dict.tsv.gz
```

- Сделать свою модель из данных wikipedia:

```
from datasets import load_dataset

ds = load_dataset(
   "rumbleFTW/wikipedia-20220301-en-raw",
   split="train[:1000000]",
   streaming=False,
   )
ds.to_csv('wikipedia_sample.csv.gz', compression='gzip', index=False)
```

Марковская модель считает что каждое следующее слово зависит от одного или нескольких предыдущих, количество предыдущих слов на основании которого определяется следующее, называется порядком модели. Сначала мы используем обычный случайный генератор с равномерным распределением, чтобы определить длину текста в диапазоне от `--min-sentence-len` до `--max-sentence-len`, который будем генерировать далее при помощью марковской модели слово за словом, пока не наберем нужное количество слов. Такое построение данных необходимо чтобы текст был похож на какой то реальный человеческий текст, а не был просто беспорядочным набором букв.

```
ydb workload fulltext import generator
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--model <путь>` или `-m <путь>`	Путь к файлу модели цепи Маркова (`.tsv.gz`).	Обязательный
`--rows <значение>`	Количество строк для генерации.	`100000`
`--min-sentence-len <значение>`	Минимальное количество слов в генерируемом документе.	`100`
`--max-sentence-len <значение>`	Максимальное количество слов в генерируемом документе.	`1000`

При генерации текста параметры `--min-sentence-len` и `--max-sentence-len` используются для того чтобы случайно сгенерировать целевую длину текста, для генерации используется равномерное распределение.

Общие параметры команды import

Имя	Описание	Значение по умолчанию
`--upload-threads <значение>` или `-t <значение>`	Количество потоков исполнения для подготовки данных.	Рассчитывается автоматически и равно числу доступных ядер на клиентской машине.
`--bulk-size <значение>`	Размер порции для отправки данных в строках.	10000
`--max-in-flight <значение>`	Максимальное количество порций данных, одновременно находящихся в обработке.	128
`--file-output-path <значение>` или `-f <путь>`	Если эта опция установлена, данные не будут загружены в базу данных, а будут сохранены в каталоге <путь>.	

Запуск нагрузки

Запустите нагрузочное тестирование в одном из двух режимов: `select` (полнотекстовый поиск) или `upsert` (вставка новых документов).

Нагрузочное тестирование поисковых запросов

Выполняет полнотекстовые поисковые запросы к индексированной таблице. Запросы могут генерироваться при помощи цепи Маркова или считываться из заранее загруженной таблицы запросов.

```
ydb workload fulltext run select --model markov_dict.tsv.gz
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--model <путь>` или `-m <путь>`	Путь к файлу модели цепи Маркова (`.tsv.gz`) для генерации запросов. Необходимо указать либо `--model` , либо `--query-table` .	
`--query-table <имя>`	Имя таблицы с заранее загруженными запросами. Таблица должна содержать столбец `query` . Необходимо указать либо `--model` , либо `--query-table` .	
`--index <имя>`	Имя полнотекстового индекса для использования.	`index`
`--min-query-len <значение>`	Минимальное количество слов в генерируемом запросе.	`1`
`--max-query-len <значение>`	Максимальное количество слов в генерируемом запросе.	`5`
`--top-size <значение>`	Количество строк для выборки из таблицы при формировании набора слов для запросов.	`1000`
`--limit <значение>`	Ограничение количества результатов на запрос. `0` — без ограничения.	`0`

Общие параметры для всех видов нагрузки

Имя	Описание	Значение по умолчанию
`--dry-run`	Не выполнять инициализационные запросы, а только вывести их текст.	
`--check-canonical` или `-c`	Использовать специальную версию запросов (они имеют детерминированные ответы) и сверять результаты с каноническими.	
`--output <значение>`	Имя файла, в котором будут сохранены результаты выполнения запросов.	`results.out`
`--iterations <значение>`	Количество выполнений каждого из запросов нагрузки.	`1`
`--json <имя>`	Имя файла, в котором будет сохранена статистика выполнения запросов в формате `json` .	Файл не сохраняется
`--ministat <имя>`	Имя файла, в котором будет сохранена статистика выполнения запросов в формате `ministat` .	Файл не сохраняется
`--csv <имя>`	Имя файла для сохранения CSV-версии таблицы с результатами.	Файл не сохраняется
`--plan <имя>`	Имя файла для сохранения плана запроса. Если задано, то сохраняются файлы `<имя>.<номер запроса>.explain` и `<имя>.<номер запроса>.<номер итерации>` с планами в нескольких форматах: `ast` , `json` , `svg` и `table` .	Планы не сохраняются.
`--query-prefix <префикс>`	Префикс запроса. Каждый префикс будет добавлен отдельной строкой в начало каждого запроса. Если нужно указать несколько префиксов, используйте параметр несколько раз.	По умолчанию не задан
`--retries`	Количество повторных попыток выполнения каждого запроса	`0`
`--include`	Имена, номера или диапазоны номеров запросов, которые нужно выполнить в рамках нагрузки. Указываются через запятую, например: `1,2,4-6` .	Все запросы
`--exclude`	Имена, номера или диапазоны номеров запросов, которые нужно исключить из нагрузки. Указываются через запятую, например: `1,2,4-6` .	
`--verbose` или `-v`	Выводить больше информации на экран в процессе выполнения запросов.	
`--global-timeout <значение>`	Общий таймаут на выполнение всех запросов. Задаётся в текстовом формате, например: `0.5s` , `1m` , `100us` и т.д.	Отсутствует. Время не ограничено.
`--request-timeout <значение>`	Таймаут на выполнение каждой итерации каждого запроса. Задаётся в текстовом формате, например: `0.5s` , `1m` , `100us` и т.д.	Отсутствует. Время не ограничено.
`--threads <значение>` или `-t <значение>`	Количество потоков, генерирующих нагрузку. Ноль означает, что запросы будут выполняться в основном потоке; при ином значении запросы будут перемешаны.	`0`

`--stats <значение>`	Режим сбора расширенной статистики выполнения. Возможные значения: `full`, `profile`.	`full`

Нагрузочное тестирование на запись

Непрерывно вставляет новые документы в таблицу, генерируя текст с помощью модели цепи Маркова.

```
ydb workload fulltext run upsert --model markov_dict.tsv.gz
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--model <путь>` или `-m <путь>`	Путь к файлу модели цепи Маркова (`.tsv.gz`).	Обязательный
`--index <имя>`	Имя полнотекстового индекса для использования.	`index`
`--bulk-size <значение>`	Количество строк в одном пакете upsert.	`100`
`--min-sentence-len <значение>`	Минимальное количество слов в генерируемом документе.	`100`
`--max-sentence-len <значение>`	Максимальное количество слов в генерируемом документе.	`1000`

Общие параметры для всех видов нагрузки

Имя	Описание	Значение по умолчанию
`--dry-run`	Не выполнять инициализационные запросы, а только вывести их текст.	
`--check-canonical` или `-c`	Использовать специальную версию запросов (они имеют детерминированные ответы) и сверять результаты с каноническими.	
`--output <значение>`	Имя файла, в котором будут сохранены результаты выполнения запросов.	`results.out`
`--iterations <значение>`	Количество выполнений каждого из запросов нагрузки.	`1`
`--json <имя>`	Имя файла, в котором будет сохранена статистика выполнения запросов в формате `json`.	Файл не сохраняется
`--ministat <имя>`	Имя файла, в котором будет сохранена статистика выполнения запросов в формате `ministat`.	Файл не сохраняется
`--csv <имя>`	Имя файла для сохранения CSV-версии таблицы с результатами.	Файл не сохраняется
`--plan <имя>`	Имя файла для сохранения плана запроса. Если задано, то сохраняются файлы `<имя>.<номер запроса>.explain` и `<имя>.<номер запроса>.<номер итерации>` с планами в нескольких форматах: `ast`, `json`, `svg` и `table`.	Планы не сохраняются.
`--query-prefix <префикс>`	Префикс запроса. Каждый префикс будет добавлен отдельной строкой в начало каждого запроса. Если нужно указать несколько префиксов, используйте параметр несколько раз.	По умолчанию не задан
`--retries`	Количество повторных попыток выполнения каждого запроса	`0`
`--include`	Имена, номера или диапазоны номеров запросов, которые нужно выполнить в рамках нагрузки. Указываются через запятую, например: `1,2,4-6`.	Все запросы
`--exclude`	Имена, номера или диапазоны номеров запросов, которые нужно исключить из нагрузки. Указываются через запятую, например: `1,2,4-6`.	
`--verbose` или `-v`	Выводить больше информации на экран в процессе выполнения запросов.	
`--global-timeout <значение>`	Общий таймаут на выполнение всех запросов. Задаётся в текстовом формате, например: `0.5s`, `1m`, `100us` и т.д.	Отсутствует. Время не ограничено.
`--request-timeout <значение>`	Таймаут на выполнение каждой итерации каждого запроса. Задаётся в текстовом формате, например: `0.5s`, `1m`, `100us` и т.д.	Отсутствует. Время не ограничено.
`--threads <значение>` или `-t <значение>`	Количество потоков, генерирующих нагрузку. Ноль означает, что запросы будут выполняться в основном потоке; при ином значении запросы будут перемешаны.	`0`
`--stats <значение>`	Режим сбора расширенной статистики выполнения. Возможные значения: `full`, `profile`.	`full`

Построение модели цепи Маркова

Перед использованием генератора или режимов `run upsert` / `run select` с генерируемыми запросами необходимо построить модель цепи Маркова из текстового датасета. Модель фиксирует вероятности переходов между словами и используется для генерации реалистичного текста.

```
ydb workload fulltext model --input wikipedia_sample.csv.gz --output markov_dict.tsv.gz --order 3
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--input <путь>` или `-i <путь>`	Путь к файлу или директории с датасетом. Поддерживаемые форматы: `.csv[.gz]`, `.tsv[.gz]`. Файл должен содержать столбец `text`.	Обязательный
`--output <путь>` или `-o <путь>`	Путь к выходному файлу словаря модели.	`markov_dict.tsv.gz`
`--order <значение>` или `-n <значение>`	Порядок цепи Маркова (размер контекста n-граммы). Порядок 1 использует контекст из одного слова для предсказания следующего слова, порядок 2 из двух слов и т.д. Допустимые значения: от 1 до 5.	`1`

Параметр `--order` задаёт количество предыдущих слов, на основании которых принимается решение о том, какое слово выбрать следующим. Обычно этот параметр принимает значения от 1 до 5: меньшие значения приводят к более случайным текстам, а большие — к большему сходству с исходным текстом. Размер марковской модели растёт экспоненциально с ростом значения `--order`.

Очистка данных

Удаление таблиц, созданных для нагрузочного тестирования

```
ydb workload fulltext clean
```

Примеры использования

Пример с генерируемым датасетом

1. Скачать либо обучить Марковскую модель:
   - Скачать модель из S3:

     ```
     wget https://storage.yandexcloud.net/ydb-public/markov_dict.tsv.gz
     ```

   - Обучение модели на данных с Wikipedia:

     ```
     from datasets import load_dataset

     ds = load_dataset(
        "rumbleFTW/wikipedia-20220301-en-raw",
        split="train[:1000000]",
        streaming=False,
        )
     ds.to_csv('wikipedia_sample.csv.gz', compression='gzip', index=False)
     ```

     ```
     ydb workload fulltext model --input wikipedia_sample.csv.gz --output markov_dict.tsv.gz --order 3
     ```

2. Инициализируйте таблицы для нагрузочного тестирования:

   ```
   ydb workload fulltext init
   ```

3. Генерация и загрузка в БД синтетических документов:

   ```
   ydb workload fulltext import generator --model markov_dict.tsv.gz
   ```

4. Запуск нагрузки на поиск:

   ```
   ydb workload fulltext run select --model markov_dict.tsv.gz
   ```

   Пример вывода:

   ```
   Window      Txs Txs/Sec Retries Errors  p50(ms) p95(ms) p99(ms) pMax(ms)
   1            23 23      0       0       242     407     453     453
   2            18 18      0       1       611     807     915     915
   3             6 6       0       7       161     991     991     991
   4            35 35      0       0       173     803     923     923
   5            59 59      0       1       135     647     759     803
   6            26 26      0       0       257     335     339     339
   7            15 15      0       1       651     963     975     975
   8            11 11      0       1       871     943     963     963
   9            18 18      0       6       190     967     999     999
   10           31 31      0       1       433     755     995     995

   Total       Txs Txs/Sec Retries Errors  p50(ms) p95(ms) p99(ms) pMax(ms)
   10          242 24.2    0       18      259     915     991     999
   ```

   Описание колонок:

- Window – Порядковый номер временного окна (например, каждая секунда или фиксированный интервал).
- Txs – Количество успешно выполненных транзакций в данном окне (или общее количество по всем окнам в нижней секции).
- Txs/Sec – Количество транзакций в секунду для данного окна (или среднее значение для итоговой строки).
- Retries – Количество автоматических повторных попыток из-за временных ошибок (например, конфликтов или троттлинга).
- Errors – Количество неисправимых ошибок.
- p50(ms) – Медианная (50-й перцентиль) задержка выполнения транзакции в миллисекундах.
- p95(ms) – Задержка на уровне 95-го перцентиля (миллисекунды).
- p99(ms) – Задержка на уровне 99-го перцентиля (миллисекунды).
- pMax(ms) – Максимальная наблюдаемая задержка в пределах окна (или глобальный максимум для итоговой строки).

5. Запуск нагрузки на запись:

```
ydb workload fulltext run upsert --model markov_dict.tsv.gz
```

Пример вывода:

```
Window      Txs Txs/Sec Retries Errors  p50(ms) p95(ms) p99(ms) pMax(ms)
1           255 255     0       0       31      34      117     123
2           279 279     0       0       32      37      42      45
3           282 282     0       0       31      35      37      40
4           277 277     0       0       32      37      39      42
5           281 281     0       0       32      38      40      42
6           278 278     0       0       32      39      41      43
7           279 279     0       0       32      38      39      43
8           275 275     0       0       33      39      42      46
9           282 282     0       0       32      37      39      40
10          293 293     0       0       31      37      39      41

Total       Txs Txs/Sec Retries Errors  p50(ms) p95(ms) p99(ms) pMax(ms)
```

6. Удаление таблиц, созданных для нагрузочного тестирования:

```
ydb workload fulltext clean
```

Пример с датасетом MS MARCO

1. Скачайте пакет данных (содержит `documents.tsv.gz`, `queries.tsv.gz`, `markov_dict.tsv.gz` и `query_relevances.tsv.gz`):

```
wget https://storage.yandexcloud.net/ydb-public/quality-bundle.tar
tar -xf quality-bundle.tar
```

2. Инициализируйте таблицы для нагрузочного тестирования:

```
ydb workload fulltext init
```

3. Импортируйте документы из датасета:

```
ydb workload fulltext import files
```

4. Запуск нагрузки на поиск:

```
ydb workload fulltext run select --quality
```

Пример вывода:

```
Search quality measurement...
Search quality measurement completed for 100 queries in 1 seconds.
nDCG@10:  0.270
Errors:   0
Window      Txs Txs/Sec Retries Errors  p50(ms) p95(ms) p99(ms) pMax(ms)
1            23 23      0       0       242     407     453     453
2            18 18      0       1       611     807     915     915
3             6 6       0       7       161     991     991     991
4            35 35      0       0       173     803     923     923
5            59 59      0       1       135     647     759     803
6            26 26      0       0       257     335     339     339
7            15 15      0       1       651     963     975     975
8            11 11      0       1       871     943     963     963
9            18 18      0       6       190     967     999     999
10           31 31      0       1       433     755     995     995

Total       Txs Txs/Sec Retries Errors  p50(ms) p95(ms) p99(ms) pMax(ms)
10          242 24.2    0       18      259     915     991     999
```

5. Запуск нагрузки на запись:

```
ydb workload fulltext run upsert
```

Пример вывода:

```
Window      Txs Txs/Sec Retries Errors  p50(ms) p95(ms) p99(ms) pMax(ms)
1           255 255     0       0       31      34      117     123
2           279 279     0       0       32      37      42      45
3           282 282     0       0       31      35      37      40
4           277 277     0       0       32      37      39      42
5           281 281     0       0       32      38      40      42
6           278 278     0       0       32      39      41      43
7           279 279     0       0       32      38      39      43
8           275 275     0       0       33      39      42      46
9           282 282     0       0       32      37      39      40
10          293 293     0       0       31      37      39      41

Total       Txs Txs/Sec Retries Errors  p50(ms) p95(ms) p99(ms) pMax(ms)
10         2781 278.1   0       0       32      38      41      123
Window      Txs Txs/Sec Retries Errors  p50(ms) p95(ms) p99(ms) pMax(ms)
1           255 255     0       0       31      34      117     123
2           279 279     0       0       32      37      42      45
3           282 282     0       0       31      35      37      40
4           277 277     0       0       32      37      39      42
5           281 281     0       0       32      38      40      42
6           278 278     0       0       32      39      41      43
7           279 279     0       0       32      38      39      43
8           275 275     0       0       33      39      42      46
9           282 282     0       0       32      37      39      40
10          293 293     0       0       31      37      39      41

Total       Txs Txs/Sec Retries Errors  p50(ms) p95(ms) p99(ms) pMax(ms)
10         2781 278.1   0       0       32      38      41      123
```

6. Удаление таблиц, созданных для нагрузочного тестирования:

```
ydb workload fulltext clean
```

Vector нагрузка

Позволяет тестировать производительность и полноту векторного поиска YDB с помощью точного и приближённого поиска. Поддерживается тестирование обычных векторных индексов и векторных индексов с фильтрацией.

Нагрузка позволяет загружать векторы из реального датасета (например, Wikipedia embeddings) или генерировать случайные синтетические векторы. После загрузки данных можно построить векторный индекс, запустить поисковые запросы для измерения полноты и производительности и очистить таблицы нагрузки.

Структура команды

```
ydb [global options...] workload vector [options...] <subcommand>
```

Подкоманды:

```
vector                YDB vector workload.
├─ init                 Инициализация таблиц для нагрузки
├─ import               Загрузка данных в таблицы нагрузки
|   ├─ files             Импорт векторов из файлов
|   └─ generator         Генерация случайных векторов
├─ build-index          Создание и инициализация векторного индекса
├─ drop-index           Удаление векторного индекса
├─ run                  Запуск нагрузки
|   ├─ select           Поиск векторов и измерение производительности/полноты
|   └─ upsert           Вставка или обновление строк с векторами
└─ clean                Удаление таблиц, созданных для нагрузочного тестирования
```

Инициализация нагрузки

Создайте таблицу для нагрузочного тестирования:

```
ydb workload vector init
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--table <имя>`	Имя основной таблицы для хранения векторов.	`vector_index_workload`
`--min-partitions <значение>`	Минимальное количество партиций таблицы.	`40`
`--partition-size <значение>`	Целевой размер партиции, в МБ.	`2000`
`--auto-partition <значение>`	Включить автопартицирование по нагрузке (`1` — включено, `0` — выключено).	`1`
`--prefixed`	Добавить столбец `prefix` в таблицу для использования с фильтрующим (prefixed) векторным индексом.	
`--clear`	Удалить и пересоздать таблицу, если она уже существует.	
`--dry-run`	Только вывести DDL-запрос, не выполняя его.	

Создаваемая таблица имеет следующую схему:

- `id Uint64 NOT NULL` — первичный ключ;
- `embedding String` — сериализованный вектор-эмбеддинг;
- `prefix Uint64 NOT NULL` (только при указании `--prefixed`) — столбец фильтра для prefixed-индексов.

Загрузка данных

После инициализации загрузите данные в таблицу. Доступны две подкоманды: `files` — для импорта векторов из готового датасета, и `generator` — для генерации синтетических случайных векторов.

После завершения импорта автоматически строится векторный индекс с именем `index` по столбцу `embedding` (если не указано `--index-type None`).

Импорт из файлов

Импорт векторов из файлов (CSV, TSV или Parquet, возможно в сжатом формате gzip). Датасет должен содержать столбцы `id` и `embedding`. Дополнительные столбцы игнорируются.

В файлах CSV/TSV эмбеддинги должны быть представлены в виде списка чисел с плавающей точкой, например `"[ 1.0, 2.0, 3.0 ]"`. В файлах Parquet эмбеддинги могут быть как списком `float32`, так и уже сериализованными бинарными эмбеддингами YDB.

Пример запуска:

```
ydb workload vector import files
```

Доступные параметры

Имя	Описание	Значение по умолчанию

`--input <путь>` или `-i <путь>`	Путь к файлу или директории с датасетом. Поддерживаемые форматы: CSV/TSV (с опциональным gzip-сжатием), Parquet. Импортируются только столбцы `id` и `embedding`.	Обязательный
`--format <формат>`	Формат файлов. Один из `csv`, `tsv`, `parquet`. Если указан, из директории импортируются только файлы соответствующего формата. Если не указан — формат определяется по расширению файла.	
`--embedding-column-name <имя>`	Альтернативное имя исходного столбца с эмбеддингами.	`embedding`
`--table <имя>`	Имя таблицы для загрузки данных.	`vector_index_workload`
`--index <имя>`	Имя векторного индекса, который будет построен после импорта.	`index`
`--index-type <тип>`	Тип индекса, строимого после импорта. Возможные значения: `None`, `KmeansTree`. Указание `None` отключает построение индекса.	`KmeansTree`
`--vector-type <тип>`	Тип векторов. Один из `float`, `int8`, `uint8`, `bit`.	`float`
`--vector-dimension <значение>`	Размерность векторов.	`1024`
`--distance <значение>`	Функция расстояния или сходства. Один из `inner_product`, `cosine`, `euclidean`, `manhattan`.	`inner_product`
`--kmeans-tree-levels <значение>`	Количество уровней в дереве k-means. Если не задано, определяется сервером автоматически. См. тип kmeans-tree.	Автоопределение
`--kmeans-tree-clusters <значение>`	Количество кластеров k-means. Если не задано, определяется сервером автоматически. См. тип kmeans-tree.	Автоопределение
`--kmeans-tree-covering <значение>`	Создавать покрывающий индекс (`1` — да, `0` — нет).	`0`
`--kmeans-tree-prefixed <значение>`	Создавать prefixed (фильтрующий) индекс (`1` — да, `0` — нет). Таблица должна быть создана с параметром `--prefixed`.	`0`

Примечание

Подробнее о параметрах построения индекса `--kmeans-tree-*` смотрите в разделе Тип kmeans-tree.

Общие параметры команды import

Имя	Описание	Значение по умолчанию
`--upload-threads <значение>` или `-t <значение>`	Количество потоков исполнения для подготовки данных.	Рассчитывается автоматически и равно числу доступных ядер на клиентской машине.
`--bulk-size <значение>`	Размер порции для отправки данных в строках.	10000
`--max-in-flight <значение>`	Максимальное количество порций данных, одновременно находящихся в обработке.	128
`--file-output-path <значение>` или `-f <путь>`	Если эта опция установлена, данные не будут загружены в базу данных, а будут сохранены в каталоге <путь>.	

Генерация синтетических данных

Генерация случайных векторов и загрузка их в таблицу. Компоненты векторов выбираются из равномерного распределения и сериализуются в бинарный формат эмбеддингов YDB.

```
ydb workload vector import generator
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--rows <значение>`	Количество строк для генерации.	`10000`
`--prefix-count <значение>`	Количество различных значений префикса для prefixed-индекса.	`100`
`--seed <значение>`	Зерно для генератора случайных чисел.	`42`
`--table <имя>`	Имя таблицы для загрузки данных.	`vector_index_workload`
`--index <имя>`	Имя векторного индекса, который будет построен после импорта.	`index`

`--index-type <тип>`	Тип индекса. Возможные значения: `None`, `KmeansTree`.	`KmeansTree`
`--vector-type <тип>`	Тип векторов. Один из `float`, `int8`, `uint8`, `bit`.	`float`
`--vector-dimension <значение>`	Размерность векторов.	`1024`
`--distance <значение>`	Функция расстояния или сходства.	`inner_product`
`--kmeans-tree-levels <значение>`	Количество уровней в дереве k-means. Если не задано, определяется сервером автоматически.	Автоопределение
`--kmeans-tree-clusters <значение>`	Количество кластеров k-means. Если не задано, определяется сервером автоматически.	Автоопределение
`--kmeans-tree-covering <значение>`	Создавать покрывающий индекс.	`0`
`--kmeans-tree-prefixed <значение>`	Создавать prefixed-индекс.	`0`

Примечание

Подробнее о параметрах построения индекса `--kmeans-tree-*` смотрите в разделе Тип kmeans-tree.

Общие параметры команды import

Имя	Описание	Значение по умолчанию
`--upload-threads <значение>` или `-t <значение>`	Количество потоков исполнения для подготовки данных.	Рассчитывается автоматически и равно числу доступных ядер на клиентской машине.
`--bulk-size <значение>`	Размер порции для отправки данных в строках.	10000
`--max-in-flight <значение>`	Максимальное количество порций данных, одновременно находящихся в обработке.	128
`--file-output-path <значение>` или `-f <путь>`	Если эта опция установлена, данные не будут загружены в базу данных, а будут сохранены в каталоге <путь>.	

Построение векторного индекса

Если данные загружались с параметром `--index-type None`, или если нужно построить дополнительный индекс с другими параметрами, векторный индекс можно построить на существующей таблице командой `build-index`.

```
ydb workload vector build-index --distance cosine
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--table <имя>`	Имя таблицы, на которой строится индекс.	`vector_index_workload`
`--index <имя>`	Имя создаваемого индекса.	`index`
`--vector-type <тип>`	Тип векторов. Один из `float`, `int8`, `uint8`, `bit`.	`float`
`--vector-dimension <значение>`	Размерность векторов.	`1024`
`--distance <значение>`	Функция расстояния или сходства.	`inner_product`
`--kmeans-tree-levels <значение>`	Количество уровней в дереве k-means. Если не задано, определяется сервером автоматически.	Автоопределение
`--kmeans-tree-clusters <значение>`	Количество кластеров k-means. Если не задано, определяется сервером автоматически.	Автоопределение
`--dry-run`	Только вывести DDL-запрос, не выполняя его.	

Примечание

Подробнее о параметрах построения индекса `--kmeans-tree-*` смотрите в разделе Тип kmeans-tree.

Удаление векторного индекса

Удаление ранее созданного векторного индекса.

```
ydb workload vector drop-index
```

Доступные параметры

Имя	Описание	Значение по умолчанию

`--table <имя>`	Имя таблицы, в которой находится индекс.	`vector_index_workload`
`--index <имя>`	Имя удаляемого индекса.	`index`
`--dry-run`	Только вывести DDL-запрос, не выполняя его.	

Запуск нагрузки

Запустите нагрузочное тестирование в одном из двух режимов: `select` (векторный поиск) или `upsert` (вставка новых векторов).

Нагрузочное тестирование поисковых запросов

Выполняет запросы векторного поиска к индексированной таблице. Опционально измеряет полноту, сравнивая результаты приближённого поиска с точным (полным перебором), затем выполняет замер производительности.

```
ydb workload vector run select --recall
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--table <имя>`	Имя основной таблицы с векторами.	`vector_index_workload`
`--index <имя>`	Имя векторного индекса.	`index`
`--query-table <имя>`	Имя таблицы с тестовой выборкой векторов. Если не задано, используются случайные строки из основной таблицы.	
`--targets <значение>`	Размер тестовой выборки.	`100`
`--limit <значение>`	Максимальное число записей в результатах одного запроса.	`5`
`--kmeans-tree-clusters <значение>`	Максимальное число кластеров, проверяемых во время поиска (KMeansTreeSearchTopSize).	`1`
`--recall`	Включить замер полноты приближённого поиска относительно точного.	
`--recall-threads <значение>`	Число параллельных запросов при оценке полноты.	`10`
`--non-indexed`	Использовать настройки поиска из индекса, но искать без индекса, полным перебором.	
`--stale-ro`	Чтение в режиме StaleRO.	

Важно

Обратите внимание на параметр `--kmeans-tree-clusters` — его увеличение значительно повышает полноту поиска ценой его замедления. Попробуйте значения в пределах от 1 до числа кластеров, указанного при создании индекса.

Общие параметры для всех видов нагрузки

Имя	Описание	Значение по умолчанию
`--dry-run`	Не выполнять инициализационные запросы, а только вывести их текст.	
`--check-canonical` или `-c`	Использовать специальную версию запросов (они имеют детерминированные ответы) и сверять результаты с каноническими.	
`--output <значение>`	Имя файла, в котором будут сохранены результаты выполнения запросов.	`results.out`
`--iterations <значение>`	Количество выполнений каждого из запросов нагрузки.	`1`
`--json <имя>`	Имя файла, в котором будет сохранена статистика выполнения запросов в формате `json`.	Файл не сохраняется
`--ministat <имя>`	Имя файла, в котором будет сохранена статистика выполнения запросов в формате `ministat`.	Файл не сохраняется
`--csv <имя>`	Имя файла для сохранения CSV-версии таблицы с результатами.	Файл не сохраняется
`--plan <имя>`	Имя файла для сохранения плана запроса. Если задано, то сохраняются файлы `<имя>.<номер запроса>.explain` и `<имя>.<номер запроса>.<номер итерации>` с планами в нескольких форматах: `ast`, `json`, `svg` и `table`.	Планы не сохраняются.
`--query-prefix <префикс>`	Префикс запроса. Каждый префикс будет добавлен отдельной строкой в начало каждого запроса. Если нужно указать несколько префиксов, используйте параметр несколько раз.	По умолчанию не задан
`--retries`	Количество повторных попыток выполнения каждого запроса	`0`

`--include`	Имена, номера или диапазоны номеров запросов, которые нужно выполнить в рамках нагрузки. Указываются через запятую, например: `1,2,4-6`.	Все запросы
`--exclude`	Имена, номера или диапазоны номеров запросов, которые нужно исключить из нагрузки. Указываются через запятую, например: `1,2,4-6`.	
`--verbose` или `-v`	Выводить больше информации на экран в процессе выполнения запросов.	
`--global-timeout <значение>`	Общий таймаут на выполнение всех запросов. Задаётся в текстовом формате, например: `0.5s`, `1m`, `100us` и т.д.	Отсутствует. Время не ограничено.
`--request-timeout <значение>`	Таймаут на выполнение каждой итерации каждого запроса. Задаётся в текстовом формате, например: `0.5s`, `1m`, `100us` и т.д.	Отсутствует. Время не ограничено.
`--threads <значение>` или `-t <значение>`	Количество потоков, генерирующих нагрузку. Ноль означает, что запросы будут выполняться в основном потоке; при ином значении запросы будут перемешаны.	`0`
`--stats <значение>`	Режим сбора расширенной статистики выполнения. Возможные значения: `full`, `profile`.	`full`

Нагрузочное тестирование на запись

Непрерывно вставляет новые строки с векторами в таблицу, генерируя случайные эмбеддинги «на лету».

```
ydb workload vector run upsert
```

Доступные параметры

Имя	Описание	Значение по умолчанию
`--table <имя>`	Имя таблицы для вставки.	`vector_index_workload`
`--index <имя>`	Имя векторного индекса.	`index`
`--bulk-size <значение>`	Количество строк в одном пакете upsert.	`100`
`--prefixed`	Генерировать вставки со столбцом `prefix` (для prefixed-индексов).	
`--prefix-count <значение>`	Количество различных значений префикса. Используется только при указании `--prefixed`.	`1000`

Общие параметры для всех видов нагрузки

Имя	Описание	Значение по умолчанию
`--dry-run`	Не выполнять инициализационные запросы, а только вывести их текст.	
`--check-canonical` или `-c`	Использовать специальную версию запросов (они имеют детерминированные ответы) и сверять результаты с каноническими.	
`--output <значение>`	Имя файла, в котором будут сохранены результаты выполнения запросов.	`results.out`
`--iterations <значение>`	Количество выполнений каждого из запросов нагрузки.	`1`
`--json <имя>`	Имя файла, в котором будет сохранена статистика выполнения запросов в формате `json`.	Файл не сохраняется
`--ministat <имя>`	Имя файла, в котором будет сохранена статистика выполнения запросов в формате `ministat`.	Файл не сохраняется
`--csv <имя>`	Имя файла для сохранения CSV-версии таблицы с результатами.	Файл не сохраняется
`--plan <имя>`	Имя файла для сохранения плана запроса. Если задано, то сохраняются файлы `<имя>.<номер запроса>.explain` и `<имя>.<номер запроса>.<номер итерации>` с планами в нескольких форматах: `ast`, `json`, `svg` и `table`.	Планы не сохраняются.
`--query-prefix <префикс>`	Префикс запроса. Каждый префикс будет добавлен отдельной строкой в начало каждого запроса. Если нужно указать несколько префиксов, используйте параметр несколько раз.	По умолчанию не задан
`--retries`	Количество повторных попыток выполнения каждого запроса	`0`
`--include`	Имена, номера или диапазоны номеров запросов, которые нужно выполнить в рамках нагрузки. Указываются через запятую, например: `1,2,4-6`.	Все запросы
`--exclude`	Имена, номера или диапазоны номеров запросов, которые нужно исключить из нагрузки. Указываются через запятую, например: `1,2,4-6`.	

`--verbose` или `-v`	Выводить больше информации на экран в процессе выполнения запросов.	
`--global-timeout <значение>`	Общий таймаут на выполнение всех запросов. Задаётся в текстовом формате, например: `0.5s`, `1m`, `100us` и т.д.	Отсутствует. Время не ограничено.
`--request-timeout <значение>`	Таймаут на выполнение каждой итерации каждого запроса. Задаётся в текстовом формате, например: `0.5s`, `1m`, `100us` и т.д.	Отсутствует. Время не ограничено.
`--threads <значение>` или `-t <значение>`	Количество потоков, генерирующих нагрузку. Ноль означает, что запросы будут выполняться в основном потоке; при ином значении запросы будут перемешаны.	`0`
`--stats <значение>`	Режим сбора расширенной статистики выполнения. Возможные значения: `full`, `profile`.	`full`

Очистка данных

Удаление таблиц, созданных для нагрузочного тестирования:

```
ydb workload vector clean
```

Алгоритм тестирования

Режим `run select` выполняет следующие шаги:

1. Формируется тестовая выборка:
   - Если задана таблица с тестовой выборкой (`--query-table`), из неё выбираются первые `--targets` записей по порядку первичного ключа.
   - В противном случае выбираются случайные `--targets` записей из основной таблицы.
2. Производится оценка полноты (если указан `--recall`):
   - Для каждого элемента тестовой выборки запускается по два запроса.
   - Первый запрос — точный векторный поиск (полный перебор) по близости к векторной колонке, формируя эталонное множество результатов `R_exact`.
   - Второй запрос — приближённый векторный поиск на основе векторного индекса, получая множество результатов `R_approx`.
   - Если выбран индекс с фильтрацией, то оба поиска проводятся только среди записей с соответствующими значениями колонок фильтров.
   - Вычисляется полнота приближённого поиска по формуле (здесь `|A|` — число элементов в множестве A, а `A ∩ B` — пересечение множеств A и B).
3. Производится оценка скорости:
   - В течение заданного времени и в заданное число параллельных потоков запускаются запросы поиска по индексу.
   - Каждый запрос запускается для случайного элемента тестовой выборки.
   - Вычисляется средняя производительность (RPS) и время ответа на разных квантилях.

Примеры использования

Пример с генерируемыми данными

1. Инициализируйте таблицу для нагрузочного тестирования:

   ```
   ydb workload vector init
   ```

2. Сгенерируйте и загрузите синтетические векторы:

   ```
   ydb workload vector import generator --rows 100000 --distance cosine
   ```

3. Запустите нагрузку на поиск:

   ```
   ydb workload vector run select --recall
   ```

   Пример вывода:

   ```
   Recall: 0.8950
   Window      Txs Txs/Sec Retries Errors  p50(ms) p95(ms) p99(ms) pMax(ms)
   1            100 100     0       0       5       8       12      15
   2             98 98      0       0       5       9       13      16
   ...

   Total       Txs Txs/Sec Retries Errors  p50(ms) p95(ms) p99(ms) pMax(ms)
   10          980 98.0    0       0       5       9       14      18
   ```

   Описание колонок:

   - `Window` — Порядковый номер временного окна (например, каждая секунда или фиксированный интервал).
   - `Txs` — Количество успешно выполненных транзакций в данном окне (или общее количество по всем окнам в итоговой строке).
   - `Txs/Sec` — Количество транзакций в секунду для данного окна (или среднее значение для итоговой строки).
   - `Retries` — Количество автоматических повторных попыток из-за временных ошибок (например, конфликтов или троттлинга).

- `Errors` — Количество неисправимых ошибок.
- `p50(ms)` — Медианная (50-й перцентиль) задержка выполнения транзакции в миллисекундах.
- `p95(ms)` — Задержка на уровне 95-го перцентиля (миллисекунды).
- `p99(ms)` — Задержка на уровне 99-го перцентиля (миллисекунды).
- `pMax(ms)` — Максимальная наблюдаемая задержка в пределах окна (или глобальный максимум для итоговой строки).

4. Запустите нагрузку на запись:

```
ydb workload vector run upsert
```

5. Удаление таблиц, созданных для нагрузочного тестирования:

```
ydb workload vector clean
```

Пример с внешним датасетом

1. Подготовьте таблицы и загрузите данные. Пример создания таблицы и загрузки данных приведён в рецепте Векторный индекс с загрузкой внешнего набора данных.

   Примеры создания векторных индексов доступны на странице документации Векторные индексы.

2. Запустите нагрузку на поиск:

```
ydb -e grpc://hostname:2135 -d /Root/testdb workload vector run select \
    --table wikipedia --index idx_vector \
    --query-table wikipedia_sample --recall
```

   Пример вывода:

```
Recall: 0.8950
Window      Txs Txs/Sec Retries Errors  p50(ms) p95(ms) p99(ms) pMax(ms)
1           100 100     0       0       5       8       12      15
2             98 98     0       0       5       9       13      16
...

Total       Txs Txs/Sec Retries Errors  p50(ms) p95(ms) p99(ms) pMax(ms)
10          980 98.0    0       0       5       9       14      18
```

Примечания

- Поисковая нагрузка генерирует SQL-запросы выборки ближайших строк по векторному расстоянию из таблицы `--table` по индексу `--index`.
- Указывать названия колонок (векторной, фильтра, первичного ключа) и функцию расстояния не нужно — они извлекаются автоматически из определения индекса.
- Случайный выбор тестовых векторов из таблицы `--table` работает только в том случае, если первичный ключ таблицы — целочисленный.
- Если используется `--query-table`, в обеих таблицах имена векторной колонки и колонки фильтра должны совпадать.
- Оценка полноты поиска (`--recall`) проводится отдельным этапом перед тестированием производительности и показывает среднюю долю совпадения результатов поиска по индексу и полным перебором (от 0 до 1).

Подготовка тестовой выборки из большой таблицы

> **Важно**
>
> Данный этап необходим только в случае отсутствия готовой тестовой выборки и если первичный ключ основной таблицы не является целочисленным, так как в этом случае формирование случайной выборки на лету во время теста не работает.

В этом случае тестовую выборку можно подготовить, выбрав случайные строки из основной таблицы. Ниже приведён пример создания таблицы для хранения тестовой выборки:

```
CREATE TABLE vector_index_sample (
    id Uint64 NOT NULL,
    prefix Uint64 NOT NULL,
    embedding String NOT NULL,
    PRIMARY KEY (id)
);
```

Наполнение таблицы случайной выборкой приблизительно 1000 строк из большой таблицы:

```
INSERT INTO vector_index_sample
SELECT id, prefix, embedding FROM large_table
WHERE RandomNumber(id) < 0xFFFFFFFFFFFFFFFF / <число_строк_в_таблице> * 1000;
```

> **Совет**
>
> Приблизительное число строк в таблице можно посмотреть в статистике таблицы, не выполняя `SELECT COUNT(*)`.

Работа с конфигурацией

> **Примечание**
>
> До версии YDB CLI 2.20.0 команды `ydb admin cluster config` имели формат `ydb admin config`.

В этом разделе приведены команды для работы с конфигурацией кластера YDB.

- Применение конфигурации `dynconfig.yaml` на кластер:

```
ydb admin cluster config replace -f dynconfig.yaml
```

- Проверка возможности применения конфигурации `dynconfig.yaml` на кластер (проверить все валидаторы, версия конфигурации в yaml-файле должна быть выше на 1, чем версия конфигурации кластера, имя кластера должно совпадать):

```
ydb admin cluster config replace -f dynconfig.yaml --dry-run
```

- Применение конфигурации `dynconfig.yaml` на кластер игнорируя проверку версий и кластера (версия и кластер всё равно будут перезаписаны на корректные):

```
ydb admin cluster config replace -f dynconfig.yaml --force
```

- Получение основной конфигурации кластера:

```
ydb admin cluster config fetch
```

- Генерация всех возможных конечных конфигураций для `dynconfig.yaml`:

```
ydb admin cluster config resolve --all -f dynconfig.yaml
```

- Генерация конечной конфигурации для `dynconfig.yaml` при лейблах `tenant=/Root/test` и `canary=true`:

```
ydb admin cluster config resolve -f dynconfig.yaml --label tenant=/Root/test --label canary=true
```

- Генерация конечной конфигурации для `dynconfig.yaml` для лейблов с узла 1003:

```
ydb admin cluster config resolve -f dynconfig.yaml --node-id 100
```

- Генерация файла динамической конфигурации на основе статической конфигурации на кластере:

```
ydb admin cluster config genereate
```

- Инициализация директории с конфигурацией, используя путь до конфигурационного файла:

```
ydb admin node config init --config-dir <путь до директории> --from-config <путь до файла конфигурации>
```

- Инициализация директории с конфигурацией, используя конфигурацию на кластере:

```
ydb admin node config init --config-dir <путь до директории> --seed-node <эндпоинт узла кластера>
```

Работа с временной конфигурацией

В этом разделе перечислены команды, которые используются для работы с временной конфигурацией.

- Получение всех временных конфигураций кластера:

```
ydb admin volatile-config fetch --all --output-directory <dir>
```

- Получение временной конфигурации с id 1 с кластера:

```
ydb admin volatile-config fetch --id 1
```

- Применение временной конфигурации `volatile.yaml` на кластер:

```
ydb admin volatile-config add -f volatile.yaml
```

- Удаление временной конфигурации с id 1 и 3 на кластере:

```
ydb admin volatile-config drop --id 1 --id 3
```

- Удаление всех временных конфигурации на кластере:

```
ydb admin volatile-config drop --all
```

Параметры

- `-f, --filename <filename.yaml>` — считать input из файла, `-` для STDIN. Для команд принимающих n файлов (прим. resolve) можно указать несколько раз, тип файла будет определён по полю metadata
- `--output-directory <dir>` — сдампить/порезолвить файлы в директорию
- `--strip-metadata` — выкинуть поле metadata из вывода
- `--all` — расширяет вывод команд до всей конфигурации (см. продвинутое конфигурирование)
- `--allow-unknown-fields` — позволяет игнорировать неизвестные поля в конфигурации

Сценарии

Обновить основную конфигурацию кластера

```
# Получить конфигурацию кластера
ydb admin cluster config fetch > dynconfig.yaml
# Отредактировать конфигурацию вашим любимым редактором
vim dynconfig.yaml
# Применить конфигурацию dynconfig.yaml на кластер
ydb admin cluster config replace -f dynconfig.yaml
```

аналогично в одну строчку:

```
ydb admin cluster config fetch | yq '.config.actor_system_config.scheduler.resolution = 128' | ydb admin clus
ter config replace -f -
```

вывод команды:

```
OK
```

Посмотреть конфигурацию для определённого набора лейблов

```
ydb admin cluster config resolve --remote --label tenant=/Root/db1 --label canary=true
```

вывод команды:

```
---
label_sets:
- dynamic:
    type: COMMON
    value: true
config:
  actor_system_config:
    use_auto_config: true
    node_type: COMPUTE
    cpu_count: 4
```

Посмотреть конфигурацию для определённого узла

```
ydb admin cluster config resolve --remote --node-id <node_id>
```

вывод команды:

```
---
label_sets:
- dynamic:
    type: COMMON
    value: true
config:
  actor_system_config:
    use_auto_config: true
    node_type: COMPUTE
    cpu_count: 4
```

Сохранить все конфигурации локально

```
ydb admin cluster config fetch --all --output-directory <configs_dir>
ls <configs_dir>
```

вывод команды:

```
dynconfig.yaml volatile_1.yaml volatile_3.yaml
```

Посмотреть все конфигурации локально

```
ydb admin cluster config fetch --all
```

вывод команды:

```
---
metadata:
  kind: main
  cluster: unknown
  version: 1
config:
  actor_system_config:
    use_auto_config: true
    node_type: COMPUTE
    cpu_count: 4
allowed_labels: {}
selector_config: []
---
metadata:
  kind: volatile
  cluster: unknown
  version: 1
  id: 1
# some comment example
selectors:
- description: test
  selector:
    tenant: /Root/db1
  config:
    actor_system_config: !inherit
      use_auto_config: true
      cpu_count: 12
```

Посмотреть конечную конфигурацию для определённого узла из сохраненной локально исходной конфигурации

```
ydb admin cluster config resolve -k <configs_dir> --node-id <node_id>
```

вывод команды:

```
---
label_sets:
- dynamic:
    type: COMMON
    value: true
config:
  actor_system_config:
    use_auto_config: true
    node_type: COMPUTE
    cpu_count: 4
```

Управление конфигурацией

В YDB CLI доступны команды для управления динамической конфигурацией на разных уровнях системы.

Общий вид команды:

```
ydb [global options...] admin [scope] config [subcommands...]
```

- `global options` — глобальные параметры.
- `scope` — область применения конфигурации (cluster, node).
- `subcommands` — подкоманды для управления конфигурацией.

Посмотрите описание команды:

```
ydb admin --help
```

Доступные области конфигурации

Конфигурация кластера

Управление конфигурацией на уровне кластера:

```
ydb admin cluster config [subcommands...]
```

Доступные подкоманды:

- fetch - получение текущей конфигурации кластера.
- generate - генерация конфигурации V2 из V1 для миграции между ними.
- replace - замена текущей конфигурации кластера.
- version - отображение версии конфигурации на узлах (V1/V2).

Конфигурация узла

Управление конфигурацией на уровне узла:

```
ydb admin node config [subcommands...]
```

Доступные подкоманды:

- init - инициализация директории для конфигурации узла.

Команды управления конфигурацией кластера

Команды управления конфигурацией кластера предназначены для работы с конфигурацией на уровне всего кластера YDB. Эти команды позволяют администраторам просматривать, изменять и управлять настройками, которые применяются ко всем узлам кластера.

> **Внимание**
>
> Команды из этого раздела могут нанести вред вашему кластеру при неправильном использовании. Из-за потенциально опасного характера этих команд **ВСЕ** глобальные параметры должны быть заданы явно. Профили отключены по умолчанию и используются только при явном указании (`--profile <имя-профиля>`). Некоторые команды не требуют глобальных опций, которые в противном случае являются обязательными.

Общий синтаксис вызова команд управления конфигурацией кластера:

```
ydb [global options] admin cluster config [command options] <subcommand>
```

где:

- `ydb` — команда запуска YDB CLI из командной строки операционной системы;
- `[global options]` — глобальные опции, одинаковые для всех команд YDB CLI;
- `admin cluster config` — команда управления конфигурацией кластера;
- `[command options]` — опции команды, специфичные для каждой команды и подкоманды;
- `<subcommand>` — подкоманда.

Команды

Ниже представлен список доступных подкоманд для управления конфигурацией кластера. Любую команду можно вызвать с опцией `--help` для получения справки по ней.

Команда / подкоманда	Краткое описание
admin cluster config fetch	Получение текущей конфигурации (псевдонимы: `get`, `dump`)
admin cluster config replace	Замена текущей конфигурации
admin cluster config generate	Генерация конфигурации V2 из конфигурации V1
admin cluster config vesion	Отображение версии конфигурации узлов (V1/V2)

admin cluster config generate

С помощью команды `admin cluster config generate` вы можете сгенерировать для кластера YDB конфигурацию V2 на основе его текущей конфигурации V1. Это один из шагов при миграции кластера на конфигурацию V2.

Общий вид команды:

```
ydb [global options...] admin cluster config generate
```

- `global options` — глобальные параметры.

Посмотрите описание команды генерации конфигурации V2:

```
ydb admin cluster config generate --help
```

Примеры

Сгенерируйте конфигурацию V2 на основе конфигурации V1:

```
ydb admin cluster config generate > config.yaml
```

После выполнения данной команды в файле `config.yaml` будет содержаться YAML-документ следующего вида:

```
metadata:
  kind: MainConfig
  cluster: ""
  version: 0
config:
  <настройки кластера>
```

Использование сгенерированной конфигурации V2

После генерации конфигурации V2 выполните следующие шаги:

1. Добавьте необходимые параметры конфигурации в конфигурационный файл.
2. Примените конфигурацию к кластеру с помощью команды `admin cluster config replace`.

Если вы используете данную команду для миграции на V2, следуйте указаниям инструкции.

admin cluster config fetch

С помощью команды `admin cluster config fetch` вы можете получить текущую конфигурацию кластера YDB. В зависимости от используемой кластером версии конфигурации, команда возвращает:

- V1 — только динамическую конфигурацию;
- V2 — всю конфигурацию.

Общий вид команды:

```
ydb [global options...] admin cluster config fetch
```

- `global options` — глобальные параметры.

Посмотрите описание команды получения конфигурации:

```
ydb admin cluster config fetch --help
```

Примеры

Получите текущую конфигурацию кластера:

```
ydb --endpoint grpc://localhost:2135 admin cluster config fetch > config.yaml
```

admin cluster config replace

С помощью команды `admin cluster config replace` вы можете загрузить конфигурацию на кластер YDB.

> **Внимание**
>
> Команды из этого раздела могут нанести вред вашему кластеру при неправильном использовании. Из-за потенциально опасного характера этих команд **ВСЕ** глобальные параметры должны быть заданы явно. Профили отключены по умолчанию и используются только при явном указании (`--profile <имя-профиля>`). Некоторые команды не требуют глобальных опций, которые в противном случае являются обязательными.

В зависимости от используемой кластером версии конфигурации, команда заменяет:

- V1 — только динамическую конфигурацию;
- V2 — всю конфигурацию.

Общий вид команды:

```
ydb [global options...] admin cluster config replace [options...]
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.

Посмотрите описание команды замены конфигурации:

```
ydb admin cluster config replace --help
```

Параметры подкоманды

Имя	Описание
`-f` , `--filename`	Путь к файлу, содержащему конфигурацию.
`--allow-unknown-fields`	Разрешить наличие неизвестных полей в конфигурации. Если флаг не указан, наличие неизвестных полей в конфигурации приводит к ошибке.
`--ignore-local-validation`	Игнорировать базовую валидацию конфигурации на стороне клиента. Если флаг не указан, YDB CLI проводит базовую валидацию конфигурации.

Примеры

Загрузите файл конфигурации на кластер:

```
ydb admin cluster config replace --filename config.yaml
```

Загрузите файл конфигурации на кластер, игнорируя локальные проверки применимости:

```
ydb admin cluster config replace -f config.yaml --ignore-local-validation
```

Загрузите файл конфигурации на кластер, игнорируя проверку конфигурации на неизвестные поля:

```
ydb admin cluster config replace -f config.yaml --allow-unknown-fields
```

Команды управления конфигурацией узла

Команды управления конфигурацией узла предназначены для работы с конфигурацией на уровне отдельных узлов кластера YDB. Эти команды позволяют администраторам кластера инициализировать, обновлять и управлять настройками отдельных узлов.

> **Внимание**
>
> Команды из этого раздела могут нанести вред вашему кластеру при неправильном использовании. Из-за потенциально опасного характера этих команд **ВСЕ** глобальные параметры должны быть заданы явно. Профили отключены по умолчанию и используются только при явном указании (`--profile <имя-профиля>`). Некоторые команды не требуют глобальных опций, которые в противном случае являются обязательными.

Общий синтаксис вызова команд управления конфигурацией узла:

```
ydb [global options] admin node config [command options] <subcommand>
```

, где:

- `ydb` – команда запуска YDB CLI из командной строки операционной системы;
- `[global options]` – глобальные опции, одинаковые для всех команд YDB CLI;
- `admin node config` – команда управления конфигурацией узла;
- `[command options]` – опции команды, специфичные для каждой команды и подкоманд;
- `<subcommand>` – подкоманда.

Команды

Ниже представлен список доступных подкоманд для управления конфигурацией узла. Любая команда может быть вызвана в командной строке с опцией `--help` для получения справки по ней.

Команда / подкоманда	Краткое описание
admin node config init	Инициализация директории для конфигурации узла

admin node config init

При развёртывании нового кластера YDB или добавлении узлов в существующий (расширении) на каждом узле требуется директория для хранения его конфигурации. Команда `admin node config init` создаёт и подготавливает эту директорию, помещая в неё заданный конфигурационный файл или получая его с другого узла кластера (seed node).

Общий вид команды:

```
ydb [global options...] admin node config init [options...]
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.

Посмотрите описание команды инициализации конфигурации узла:

```
ydb admin node config init --help
```

Параметры подкоманды

Имя	Описание
`-d`, `--config-dir`	**Обязательный**. Путь к директории для хранения файла конфигурации.
`-f`, `--from-config`	Путь к файлу с начальной конфигурацией. Необходим при первоначальном развёртывании кластера. Можно использовать и при расширении, если файл с актуальной конфигурацией кластера был предварительно доставлен на узел.
`-s`, `--seed-node`	Эндпоинт узла-источника (seed node), с которого будет получена конфигурация. Используется при расширении кластера.

Примеры

Инициализируйте конфигурационную директорию узла, используя указанный файл конфигурации:

```
ydb admin node config init --config-dir /opt/ydb/cfg-dir --from-config config.yaml
```

Инициализируйте конфигурацию узла, получив её с узла-источника:

```
ydb admin node config init --config-dir /opt/ydb/cfg-dir --seed-node <node.ydb.tech>:2135
```

Использование

После успешной инициализации директории для конфигурации узла вы можете запустить процесс `ydbd` на данном узле, добавив параметр `--config-dir` с указанием пути до директории. С этого момента при обновлении конфигурации кластера система автоматически актуализирует конфигурационный файл в указанной директории, что избавляет от необходимости ручного обновления файла конфигурации на узле.

При перезапуске узла он автоматически загружает актуальную конфигурацию из этой директории.

Команды управления кластером в режиме bridge

Функциональность Корпоративной СУБД Яндекса

Данная функциональность доступна только в Корпоративной СУБД Яндекса. В open-source версии YDB она отсутствует.

Команды управления кластером в режиме bridge позволяют просматривать состояние pile, выполнять плановую и аварийную смену PRIMARY, временно выводить pile на обслуживание и возвращать его в кластер.

Внимание

Команды из этого раздела могут нанести вред вашему кластеру при неправильном использовании. Из-за потенциально опасного характера этих команд **ВСЕ** глобальные параметры должны быть заданы явно. Профили отключены по умолчанию и используются только при явном указании (`--profile <имя-профиля>`). Некоторые команды не требуют глобальных опций, которые в противном случае являются обязательными.

Общий синтаксис вызова команд управления кластером в режиме bridge:

```
ydb [global options...] admin cluster bridge [command options...] <subcommand>
```

где:

- `ydb` — команда запуска YDB CLI из командной строки операционной системы;
- `[global options]` — глобальные параметры, одинаковые для всех команд YDB CLI;
- `admin cluster bridge` — команда управления конфигурацией кластера;
- `[command options]` — параметры команды, специфичные для каждой команды и подкоманды;
- `<subcommand>` — подкоманда.

Команды

Ниже представлен список доступных подкоманд для управления кластером в режиме bridge. Любую команду можно вызвать с опцией `--help` для получения справки по ней.

Команда / подкоманда	Краткое описание
admin cluster bridge list	Вывод состояния pile
admin cluster bridge switchover	Плановая смена `PRIMARY`
admin cluster bridge failover	Аварийное переключение
admin cluster bridge takedown	Вывод pile из кластера
admin cluster bridge rejoin	Возвращение pile в кластер

Команды управления кластером в режиме bridge

Функциональность Корпоративной СУБД Яндекса

Данная функциональность доступна только в Корпоративной СУБД Яндекса. В open-source версии YDB она отсутствует.

Команды управления кластером в режиме bridge позволяют просматривать состояние pile, выполнять плановую и аварийную смену PRIMARY, временно выводить pile на обслуживание и возвращать его в кластер.

Внимание

Команды из этого раздела могут нанести вред вашему кластеру при неправильном использовании. Из-за потенциально опасного характера этих команд **ВСЕ** глобальные параметры должны быть заданы явно. Профили отключены по умолчанию и используются только при явном указании (`--profile <имя-профиля>`). Некоторые команды не требуют глобальных опций, которые в противном случае являются обязательными.

Общий синтаксис вызова команд управления кластером в режиме bridge:

```
ydb [global options...] admin cluster bridge [command options...] <subcommand>
```

где:

- `ydb` — команда запуска YDB CLI из командной строки операционной системы;
- `[global options]` — глобальные параметры, одинаковые для всех команд YDB CLI;
- `admin cluster bridge` — команда управления конфигурацией кластера;
- `[command options]` — параметры команды, специфичные для каждой команды и подкоманды;
- `<subcommand>` — подкоманда.

Команды

Ниже представлен список доступных подкоманд для управления кластером в режиме bridge. Любую команду можно вызвать с опцией `--help` для получения справки по ней.

Команда / подкоманда	Краткое описание
admin cluster bridge list	Вывод состояния pile
admin cluster bridge switchover	Плановая смена `PRIMARY`
admin cluster bridge failover	Аварийное переключение
admin cluster bridge takedown	Вывод pile из кластера
admin cluster bridge rejoin	Возвращение pile в кластер

admin cluster bridge list

> **Функциональность Корпоративной СУБД Яндекса**
>
> Данная функциональность доступна только в Корпоративной СУБД Яндекса. В open-source версии YDB она отсутствует.

С помощью команды `admin cluster bridge list` можно вывести состояние каждого pile в режиме bridge.

Общий вид команды:

```
ydb [global options...] admin cluster bridge list [options...]
```

- `global options` — глобальные параметры CLI.
- `options` — параметры подкоманды.

Просмотр справки по команде:

```
ydb admin cluster bridge list --help
```

Параметры подкоманды

Имя	Описание
`--format <pretty, json, csv>`	Формат вывода. Допустимые значения: `pretty`, `json`, `csv`. Значение по умолчанию: `pretty`.

Примеры

Вывести список pile в человекочитаемом формате:

```
ydb admin cluster bridge list

pile-a: PRIMARY
pile-b: SYNCHRONIZED
```

Вывести состояние в формате JSON:

```
ydb admin cluster bridge list --format json

{
  "pile-a": "PRIMARY",
  "pile-b": "SYNCHRONIZED"
}
```

Вывести состояние в формате CSV:

```
ydb admin cluster bridge list --format csv

pile,state
pile-a,PRIMARY
pile-b,SYNCHRONIZED
```

admin cluster bridge switchover

Функциональность Корпоративной СУБД Яндекса

Данная функциональность доступна только в Корпоративной СУБД Яндекса. В open-source версии YDB она отсутствует.

С помощью команды `admin cluster bridge switchover` выполняется плавное, плановое переключение указанного pile в состояние `PRIMARY` через промежуточное состояние `PROMOTED`. Подробнее см. описание сценария.

Внимание

Команды из этого раздела могут нанести вред вашему кластеру при неправильном использовании. Из-за потенциально опасного характера этих команд **ВСЕ** глобальные параметры должны быть заданы явно. Профили отключены по умолчанию и используются только при явном указании (`--profile <имя-профиля>`). Некоторые команды не требуют глобальных опций, которые в противном случае являются обязательными.

Общий вид команды:

```
ydb [global options...] admin cluster bridge switchover [options...]
```

- `global options` — глобальные параметры CLI.
- `options` — параметры подкоманды.

Просмотр справки по команде:

```
ydb admin cluster bridge switchover --help
```

Параметры подкоманды

Имя	Описание
`--new-primary <pile>`	Имя pile, который должен стать новым PRIMARY.

Требования

- Целевой pile должен находиться в состоянии `SYNCHRONIZED`.

Примеры

Переключение pile `pile-b` из состояния `SYNCHRONIZED` в состояние `PRIMARY` через промежуточное состояние `PROMOTED`:

```
ydb admin cluster bridge switchover --new-primary pile-b
```

Проверка результата

Убедитесь, что спустя некоторое время (через несколько минут) состояния pile изменились корректно, с помощью команды list:

```
ydb admin cluster bridge list

pile-a: SYNCHRONIZED
pile-b: PRIMARY
```

admin cluster bridge failover

Функциональность Корпоративной СУБД Яндекса

Данная функциональность доступна только в Корпоративной СУБД Яндекса. В open-source версии YDB она отсутствует.

С помощью команды `admin cluster bridge failover` можно выполнить аварийное отключение одного или нескольких недоступных pile. При необходимости можно указать pile, который станет новым `PRIMARY`.

Внимание

Команды из этого раздела могут нанести вред вашему кластеру при неправильном использовании. Из-за потенциально опасного характера этих команд **ВСЕ** глобальные параметры должны быть заданы явно. Профили отключены по умолчанию и используются только при явном указании (`--profile <имя-профиля>`). Некоторые команды не требуют глобальных опций, которые в противном случае являются обязательными.

Общий вид команды:

```
ydb [global options...] admin cluster bridge failover [options...]
```

- `global options` — глобальные параметры CLI.
- `options` — параметры подкоманды.

Просмотр справки по команде:

```
ydb admin cluster bridge failover --help
```

Параметры подкоманды

Имя	Описание
`--pile <pile>`	Имя недоступного pile. Повторите параметр, чтобы отключить несколько pile одним обновлением конфигурации.
`--new-primary <pile>`	Имя pile, который должен стать новым `PRIMARY` pile. Укажите, если недоступный pile был `PRIMARY`.

Требования

- Перед failover убедитесь, что недоступные pile остановлены или изолированы и не смогут самостоятельно возобновить обработку запросов. Не выполняйте failover независимо по разные стороны сетевого разделения; см. восстановление после split-brain.
- Оставшиеся pile должны удовлетворять требованиям к кворумам узлов и хранилища для новой конфигурации. Команда не отменяет проверок кворума.
- Если недоступен текущий `PRIMARY`, обязательно укажите `--new-primary` и выберите pile в состоянии `SYNCHRONIZED`. При отсутствии `--new-primary` или выборе pile в состоянии, отличном от `SYNCHRONIZED`, команда вернёт ошибку без каких-либо изменений.
- Кластер не перейдёт в невалидное состояние: при нарушении требований команда ничего не изменяет и сообщает об ошибке.
- Если pile не вышел из строя, но его нужно отключить, используйте плановое отключение — команду `takedown`.
- Команда требует прав администратора кластера. Если включена обязательная аутентификация, то в аварийном состоянии кластера аутентификация по логину и паролю не может быть использована — используйте аутентификацию по клиентским сертификатам. Подробнее см. Особенности аутентификации в аварийном состоянии кластера.

Примеры

Выполнение аварийного отключения для недоступного pile под названием `pile-a`:

```
ydb admin cluster bridge failover --pile pile-a
```

Выполнение аварийного отключения для недоступного `PRIMARY` pile и назначение новым `PRIMARY` синхронизированного pile:

```
ydb admin cluster bridge failover --pile pile-a --new-primary pile-b
```

Аварийное отключение двух недоступных pile в кластере из трёх pile с продолжением работы на оставшемся синхронизированном pile:

```
ydb admin cluster bridge failover --pile pile-a --pile pile-b --new-primary pile-c
```

Оба pile переводятся в `DISCONNECTED` одним обновлением конфигурации. Отдельные команды могут завершиться ошибкой, поскольку первое обновление всё ещё требует кворума в другом недоступном pile.

Проверка результата

С помощью команды list проверьте, что недоступный pile переведён в состояние `DISCONNECTED` и (если был указан `--new-primary`) выбран новый `PRIMARY` pile :

```
ydb admin cluster bridge list
```

```
pile-a: DISCONNECTED
pile-b: PRIMARY
```

admin cluster bridge takedown

Функциональность Корпоративной СУБД Яндекса

Данная функциональность доступна только в Корпоративной СУБД Яндекса. В open-source версии YDB она отсутствует.

С помощью команды `admin cluster bridge takedown` можно выполнить плановое отключение pile. Если отключается текущий `PRIMARY`, необходимо указать новый `PRIMARY`.

Внимание

Команды из этого раздела могут нанести вред вашему кластеру при неправильном использовании. Из-за потенциально опасного характера этих команд **ВСЕ** глобальные параметры должны быть заданы явно. Профили отключены по умолчанию и используются только при явном указании (`--profile <имя-профиля>`). Некоторые команды не требуют глобальных опций, которые в противном случае являются обязательными.

Общий вид команды:

```
ydb [global options...] admin cluster bridge takedown [options...]
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.

Просмотр справки по команде:

```
ydb admin cluster bridge takedown --help
```

Параметры подкоманды

Имя	Описание
`--pile <pile>`	Имя pile, который нужно аккуратно остановить.
`--new-primary <pile>`	Имя pile, который должен стать новым `PRIMARY`, если отключается текущий `PRIMARY`.

Требования

- Если отключается текущий `PRIMARY`, обязательно укажите `--new-primary` и выберите pile в состоянии `SYNCHRONIZED`.

Примеры

Вывод `SYNCHRONIZED` pile `pile-b` из кластера:

```
ydb admin cluster bridge takedown --pile pile-b
```

Вывод `PRIMARY` pile `pile-a` из кластера с переключением pile `pile-b` из состояния `SYNCHRONIZED` в состояние `PRIMARY`:

```
ydb admin cluster bridge takedown --pile pile-a --new-primary pile-b
```

Проверка результата

Проверьте итоговые состояния pile с помощью команды list:

```
ydb admin cluster bridge list

pile-a: PRIMARY
pile-b: DISCONNECTED
```

Если отключался текущий `PRIMARY` с указанием `--new-primary`, убедитесь, что выбранный pile стал `PRIMARY`:

```
ydb admin cluster bridge list

pile-a: DISCONNECTED
pile-b: PRIMARY
```

admin cluster bridge rejoin

Функциональность Корпоративной СУБД Яндекса

Данная функциональность доступна только в Корпоративной СУБД Яндекса. В open-source версии YDB она отсутствует.

С помощью команды `admin cluster bridge rejoin` можно вернуть указанный pile в кластер после обслуживания или восстановления. После выполнения команды ожидается переход pile из состояния `DISCONNECTED` в состояние `NOT_SYNCHRONIZED`, последующая автоматическая синхронизация и переход в состояние `SYNCHRONIZED`.

Внимание

Команды из этого раздела могут нанести вред вашему кластеру при неправильном использовании. Из-за потенциально опасного характера этих команд **ВСЕ** глобальные параметры должны быть заданы явно. Профили отключены по умолчанию и используются только при явном указании (`--profile <имя-профиля>`). Некоторые команды не требуют глобальных опций, которые в противном случае являются обязательными.

Общий вид команды:

```
ydb [global options...] admin cluster bridge rejoin [options...]
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.

Просмотр справки по команде:

```
ydb admin cluster bridge rejoin --help
```

Параметры подкоманды

Имя	Описание
`--pile <pile>`	Имя pile, который нужно вернуть в кластер.

Требования

- Pile перед возвращением должен быть в состоянии `DISCONNECTED`.

Примеры

Возврат pile `pile-a` из состояния `DISCONNECTED`:

```
ydb admin cluster bridge rejoin --pile pile-a
```

Проверка результата

Сразу после выполнения команды ожидается переход pile в состояние `NOT_SYNCHRONIZED`. Проверьте результат с помощью команды list:

```
ydb admin cluster bridge list

pile-a: NOT_SYNCHRONIZED
pile-b: PRIMARY
```

После завершения синхронизации pile переходит в состояние `SYNCHRONIZED`:

```
ydb admin cluster bridge list

pile-a: SYNCHRONIZED
pile-b: PRIMARY
```

Справка по YDB SDK

Для работы с YDB доступны OpenSource SDK для следующих языков программирования:

Язык	Репозиторий на GitHub	Справка по API
Go	ydb-platform/ydb-go-sdk	https://pkg.go.dev/github.com/ydb-platform/ydb-go-sdk/v3
Java	ydb-platform/ydb-java-sdk	N/A
Python	ydb-platform/ydb-python-sdk	https://ydb-platform.github.io/ydb-python-sdk
C#	ydb-platform/ydb-dotnet-sdk	N/A
JavaScript	ydb-platform/ydb-js-sdk	https://ydb.js.org
Rust	ydb-platform/ydb-rs-sdk	N/A
C++	ydb-platform/ydb-cpp-sdk	N/A
PHP	ydb-platform/ydb-php-sdk	N/A

Документация по SDK содержит следующие разделы:

- Установка
- Аутентификация
- Обработка ошибок
- Сравнение возможностей SDK
- Параметризованные запросы
- Форматы данных
- Работа с топиками
- Работа с узлами координации
- Наблюдаемость в YDB SDK

Смотрите также:

- Документация для разработчиков приложений
- Тестовые приложения
- Рецепты кода
- Генерация кода из YQL

Справка по YDB SDK

Для работы с YDB доступны OpenSource SDK для следующих языков программирования:

Язык	Репозиторий на GitHub	Справка по API
Go	ydb-platform/ydb-go-sdk	https://pkg.go.dev/github.com/ydb-platform/ydb-go-sdk/v3
Java	ydb-platform/ydb-java-sdk	N/A
Python	ydb-platform/ydb-python-sdk	https://ydb-platform.github.io/ydb-python-sdk
C#	ydb-platform/ydb-dotnet-sdk	N/A
JavaScript	ydb-platform/ydb-js-sdk	https://ydb.js.org
Rust	ydb-platform/ydb-rs-sdk	N/A
C++	ydb-platform/ydb-cpp-sdk	N/A
PHP	ydb-platform/ydb-php-sdk	N/A

Документация по SDK содержит следующие разделы:

- Установка
- Аутентификация
- Обработка ошибок
- Сравнение возможностей SDK
- Параметризованные запросы
- Форматы данных
- Работа с топиками
- Работа с узлами координации
- Наблюдаемость в YDB SDK

Смотрите также:

- Документация для разработчиков приложений
- Тестовые приложения
- Рецепты кода
- Генерация кода из YQL

Установка SDK

Ниже приведены инструкции по быстрой установке SDK. На рабочей станции должны быть предварительно установлены и сконфигурированы инструменты по работе с выбранным языком программирования, а также пакетные менеджеры.

Описание порядка сборки из исходного кода размещено в репозиториях исходного кода на GitHub, ссылки на которые приведены на странице YDB SDK - Обзор.

Go

Выполните команду из командной строки:

```
go get -u github.com/ydb-platform/ydb-go-sdk/v3
```

Для успешной установки в вашем окружении должен быть установлен Go версии не ниже 1.17.

Java

Добавьте зависимости в Maven-проект, как описано в пункте "Install the SDK" файла `readme.md` в репозитории исходного кода.

Python

Выполните команду из командной строки:

```
python3 -m pip install ydb
```

Если команда не выполнилась успешно, убедитесь, что в вашем окружении установлен Python3 версии 3.8 или более новой, со включенным пакетным менеджером pip.

C#

```
dotnet add package Ydb.Sdk
```

JavaScript

```
npm install @ydbjs/core @ydbjs/query
```

Минимальная поддерживаемая версия Node.js® не ниже 20.19.

Rust

```
cargo add ydb
```

C++

Пакеты Debian (Ubuntu 24.04)

Готовые `.deb` -пакеты для Ubuntu 24.04 (Noble), amd64 прикреплены к каждому релизу на GitHub. Скачайте и установите их локально:

```
# Замените <TAG> на тег релиза со страницы GitHub releases
TAG=<TAG>
BASE="https://github.com/ydb-platform/ydb-cpp-sdk/releases/download/${TAG}"

wget "${BASE}/yandex-googleapis-api-common-protos-1.0.0-Linux.deb"
wget "${BASE}/libydb-cpp-dev_${TAG#v}_amd64.deb"
# Дополнительные плагины:
wget "${BASE}/libydb-cpp-iam-dev_${TAG#v}_amd64.deb"
wget "${BASE}/libydb-cpp-otel-metrics-dev_${TAG#v}_amd64.deb"
wget "${BASE}/libydb-cpp-otel-tracing-dev_${TAG#v}_amd64.deb"

sudo apt-get update
sudo apt-get install -y \
    ./yandex-googleapis-api-common-protos-*.deb \
    ./libydb-cpp-dev_*.deb ./libydb-cpp-iam-dev_*.deb \
    ./libydb-cpp-otel-metrics-dev_*.deb ./libydb-cpp-otel-tracing-dev_*.deb
```

Доступные пакеты:

- `yandex-googleapis-api-common-protos` — обязательная зависимость protobuf;
- `libydb-cpp-dev` — основной SDK: статическая библиотека, публичные заголовки и файлы CMake-пакета;
- `libydb-cpp-iam-dev` — плагин IAM-аутентификации (опционально);
- `libydb-cpp-otel-metrics-dev` — плагин метрик OpenTelemetry (опционально);
- `libydb-cpp-otel-tracing-dev` — плагин трассировки OpenTelemetry (опционально; для заголовков и библиотек OTel требуется `libydb-cpp-otel-metrics-dev`).

Примечание

- Поддерживаемая платформа: Ubuntu 24.04 (Noble), amd64.
- Пакеты собраны без apt-зависимостей; установите `yandex-googleapis-api-common-protos` вместе с основным пакетом SDK.
- Для других платформ используйте инструкции по сборке из исходного кода ниже.

Использование SDK в CMake

После установки пакетов подключите SDK в CMake-проекте:

```
find_package(ydb-cpp-sdk REQUIRED COMPONENTS Driver Table Topic)
target_link_libraries(myapp PRIVATE YDB-CPP-SDK::Driver YDB-CPP-SDK::Table)
```

При конфигурации проекта передайте `-DCMAKE_PREFIX_PATH=/usr/share/yandex`, так как пакеты устанавливаются в префикс Yandex.

Сборка из исходного кода

Склонируйте репозиторий ydb-cpp-sdk и выполните команду из командной строки:

```
cmake --preset release-${compiler} -D YDB_SDK_INSTALL=On
cmake --build --preset default
cmake --install build --prefix ${ydb_install_dir}
```

- `compiler` — ваш компилятор (`clang` или `gcc`);
- `ydb_install_dir` — путь, по которому вы хотите установить SDK.

Перед выполнением команды убедитесь, что все зависимости установлены. Полный список зависимостей для сборки и инструкции см. в README репозитория.

PHP

```
composer require ydb-platform/ydb-php-sdk
```

Аутентификация в SDK

Как описано в статье о подключении к серверу YDB, клиент с каждым запросом должен отправить аутентификационный токен. Аутентификационный токен проверяется сервером и, в случае успешной аутентификации, запрос авторизуется и выполняется, иначе возвращается ошибка `Unauthenticated`.

YDB SDK использует объект, отвечающий за генерацию таких токенов. SDK предоставляет встроенные способы получения такого объекта:

1. Методы с явной передачей параметров, каждый из методов реализует один из режимов аутентификации.
2. Метод определения режима аутентификации и необходимых параметров из переменных окружения.

Обычно объект генерации токенов создается перед инициализацией драйвера YDB и передается параметром в его конструктор. C++ и Go SDK дополнительно позволяют через один драйвер работать с несколькими БД и объектами генерации токенов.

Если объект генерации токенов не определен, драйвер не будет добавлять в запросы какой-либо аутентификационной информации. Такой подход позволяет успешно соединиться с локально развернутыми кластерами YDB без настроенной обязательной аутентификации. Если обязательная аутентификация настроена, то запросы к базе данных без аутентификационного токена будут отклоняться с выдачей ошибки аутентификации.

Методы создания объекта генерации токенов

На приведенные ниже методы можно кликнуть, чтобы перейти к исходному коду релевантного примера в репозитории. Также можно ознакомиться с рецептами кода Аутентификации.

C++

Режим	Метод
Anonymous	NYdb::CreateInsecureCredentialsProviderFactory()
Access Token	NYdb::TDriverConfig::SetAuthToken(token) или NYdb::CreateOAuthCredentialsProviderFactory(token)
Metadata	NYdb::CreateIamCredentialsProviderFactory(NYdb::TIamHost{...})
Service Account Key	NYdb::CreateIamJwtFileCredentialsProviderFactory(NYdb::TIamJwtFilename{...}) или NYdb::CreateIamJwtParamsCredentialsProviderFactory(NYdb::TIamJwtContent{...}), вспомогательно NYdb::CreateFromSaKeyFile(saKeyFile, connectionString)
Static Credentials	NYdb::CreateLoginCredentialsProviderFactory(NYdb::TLoginCredentialsParams{...})
OAuth 2.0 token exchange	NYdb::CreateOauth2TokenExchangeCredentialsProviderFactory(NYdb::TOauth2TokenExchangeParams{...}), NYdb::CreateOauth2TokenExchangeFileCredentialsProviderFactory(configFilePath, tokenEndpoint)
Определяется по переменным окружения	NYdb::CreateFromEnvironment(connectionString)

Python

Режим	Метод
Anonymous	ydb.AnonymousCredentials()
Access Token	ydb.AccessTokenCredentials(token)
Metadata	ydb.iam.MetadataUrlCredentials()
Service Account Key	ydb.iam.ServiceAccountCredentials.from_file(key_file, iam_endpoint=None, iam_channel_credentials=None)
Static Credentials	ydb.StaticCredentials.from_user_password(user, password)
OAuth 2.0 token exchange	ydb.oauth2_token_exchange.Oauth2TokenExchangeCredentials(), ydb.oauth2_token_exchange.Oauth2TokenExchangeCredentials.from_file(cfg_file, iam_endpoint=None)
Определяется по переменным окружения	`ydb.credentials_from_env_variables()`

Go

Режим	Пакет	Метод
Anonymous	ydb-go-sdk/v3	ydb.WithAnonymousCredentials()
Access Token	ydb-go-sdk/v3	ydb.WithAccessTokenCredentials(token)
Metadata	ydb-go-yc	yc.WithMetadataCredentials(ctx)
Service Account Key	ydb-go-yc	yc.WithServiceAccountKeyFileCredentials(key_file)
Static Credentials	ydb-go-sdk/v3	ydb.WithStaticCredentials(user, password)
OAuth 2.0 token exchange	ydb-go-sdk/v3	ydb.WithOauth2TokenExchangeCredentials(options...), ydb.WithOauth2TokenExchangeCredentialsFile(configFilePath)
Определяется по переменным окружения	ydb-go-sdk-auth-environ	environ.WithEnvironCredentials(ctx)

Java

Режим	Метод
Anonymous	tech.ydb.core.auth.NopAuthProvider.INSTANCE
Access Token	new tech.ydb.core.auth.TokenAuthProvider(accessToken);
Metadata	tech.ydb.auth.iam.CloudAuthHelper.getMetadataAuthProvider();
Service Account Key	tech.ydb.auth.iam.CloudAuthHelper.getServiceAccountFileAuthProvider(saKeyFile);
OAuth 2.0 token exchange	tech.ydb.auth.OAuth2TokenExchangeProvider.fromFile(cfgFile);
Определяется по переменным окружения	tech.ydb.auth.iam.CloudAuthHelper.getAuthProviderFromEnviron();

C#

Режим	Метод
Anonymous	Ничего передавать для этого режима не нужно
Access Token	new TokenProvider(accessToken)
Metadata	new Ydb.Sdk.Auth.MetadataProvider()
Service Account Key	new Ydb.Sdk.Auth.ServiceAccountProvider(saKeyFile);
OAuth 2.0 token exchange	Не поддерживается
Определяется по переменным окружения	Не поддерживается

JavaScript

Режим	Метод
Anonymous	AnonymousCredentialsProvider()
Access Token	AccessTokenCredentialsProvider({ token })
Metadata	MetadataCredentialsProvider()
Service Account Key	ServiceAccountCredentialsProvider.fromFile(saKeyFile)
Static Credentials	StaticCredentialsProvider({ username, password }, endpoint)
Определяется по переменным окружения	EnvironCredentialsProvider(connectionString)

Rust

Режим	Метод
Anonymous	ydb::AnonymousCredentials or ydb::AccessTokenCredentials::from("")
Access Token	ydb::AccessTokenCredentials::from("token")
Metadata	ydb::MetadataUrlCredentials
Service Account Key	ydb::ServiceAccountCredentials
Static Credentials	ydb::StaticCredentialsAuth
Определяется по переменным окружения	ydb::FromEnvCredentials
Выполнение внешней команды	ydb::CommandLineCredentials (например, для авторизации с помощью IAM-токена Yandex.Cloud с компьютера разработчика `ydb::CommandLineCredentials.from_cmd("yc iam create-token")`)

PHP

Режим	Метод
Anonymous	AnonymousAuthentication()
Access Token	AccessTokenAuthentication($accessToken)
Oauth Token	OAuthTokenAuthentication($oauthToken)
Metadata	MetadataAuthentication()
Service Account Key	JwtWithJsonAuthentication(key_id, $service_account_id, $privateKeyFile)
Определяется по переменным окружения	EnvironCredentials()
Static Credentials	StaticAuthentication($user, $password)

Порядок определения режима и параметров аутентификации из окружения

Выполняется следующий алгоритм, одинаковый для всех SDK:

1. Если задано значение переменной окружения `YDB_SERVICE_ACCOUNT_KEY_FILE_CREDENTIALS`, то используется режим аутентификации **System Account Key**, а ключ загружается из файла, указанного в данной переменной.
2. Иначе, если задано значение переменной окружения `YDB_ANONYMOUS_CREDENTIALS`, равное 1, то используется анонимный режим аутентификации.
3. Иначе, если задано значение переменной окружения `YDB_METADATA_CREDENTIALS`, равное 1, то используется режим аутентификации **Metadata**.
4. Иначе, если задано значение переменной окружения `YDB_ACCESS_TOKEN_CREDENTIALS`, то используется режим аутентификации **Access token**, в который передается значение данной переменной.
5. Иначе, если задано значение переменной окружения `YDB_OAUTH2_KEY_FILE`, то используется режим аутентификации **OAuth 2.0 token exchange**, а настройки загружаются из JSON файла, указанного в данной переменной.
6. Иначе используется режим аутентификации **Metadata**.

Наличие последним пунктом алгоритма выбора режима **Metadata** позволяет развернуть рабочее приложение на виртуальных машинах и в Cloud Functions Yandex.Cloud без задания каких-либо переменных окружения.

Формат файла настроек для режима аутентификации OAuth 2.0 token exchange

Описание полей JSON файла настроек метода аутентификации **OAuth 2.0 token exchange**. Набор полей зависит от типа исходного токена, `JWT` или `FIXED`.

В таблице ниже `creds_json` обозначает JSON с параметрами для исходного токена, обмениваемого на access token.

Поля, не описанные в этой таблице, игнорируются.

Поле	Тип	Описание	Значение по умолчанию/опциональность
`grant-type`	string	Grant type	`urn:ietf:params:oauth:grant-type:token-exchange`
`res`	string | list of strings	Resource	опциональное
`aud`	string | list of strings	Опция audience для запроса обмена токена	опциональное
`scope`	string | list of strings	Scope	опциональное
`requested-token-type`	string	Тип получаемого токена	`urn:ietf:params:oauth:token-type:access_token`
`subject-credentials`	creds_json	Subject credentials	опциональное
`actor-credentials`	creds_json	Actor credentials	опциональное
`token-endpoint`	string	Token endpoint. В случае с YDB CLI перезаписывается опцией `--iam-endpoint`.	опциональное
Описание полей `creds_json` (JWT)	—	—	—
`type`	string	Тип источника токена. Нужно задать константу `JWT`	—
`alg`	string	Алгоритм подписи JWT. Поддерживаются следующие алгоритмы: ES256, ES384, ES512, HS256, HS384, HS512, PS256, PS384, PS512, RS256, RS384, RS512	—
`private-key`	string	(Приватный) ключ в формате PEM (для алгоритмов `ES*`, `PS*`, `RS*`) или Base64 (для алгоритмов `HS*`) для подписи	—
`kid`	string	Стандартное поле JWT `kid` (key id)	опциональное
`iss`	string	Стандартное поле JWT `iss` (issuer)	опциональное
`sub`	string	Стандартное поле JWT `sub` (subject)	опциональное
`aud`	string	Стандартное поле JWT `aud` (audience)	опциональное
`jti`	string	Стандартное поле JWT `jti` (JWT id)	опциональное
`ttl`	string	Время жизни JWT токена	`1h`
Описание полей `creds_json` (FIXED)	—	—	—
`type`	string	Тип источника токена. Нужно задать константу `FIXED`	—
`token`	string	Значение токена	—
`token-type`	string	Значение типа токена. Это значение попадёт в параметр `subject_token_type/actor_token_type` в запросе обмена токена	—

Пример

Пример для обмена JWT токена

```
{
   "subject-credentials": {
      "type": "JWT",
      "alg": "RS256",
      "private-key": "-----BEGIN RSA PRIVATE KEY-----\n..------END RSA PRIVATE KEY-----\n",
      "kid": "my_key_id",
      "sub": "account_id"
   }
}
```

Особенности YDB Python SDK v2 (устаревшая версия)

Важно

Поведение YDB Python SDK v2 (устаревшая версия) отличается от описанного выше.

- Алгоритм работы функции `construct_credentials_from_environ()` YDB Python SDK v2:
  - Если задано значение переменной окружения `USE_METADATA_CREDENTIALS`, равное 1, то используется режим аутентификации **Metadata**
  - Иначе, если задано значение переменной окружения `YDB_TOKEN`, то используется режим аутентификации **Access Token**, в который передается значение данной переменной
  - Иначе, если задано значение переменной окружения `SA_KEY_FILE`, то используется режим аутентификации **System Account Key**, а ключ загружается из файла, имя которого указано в данной переменной
  - Иначе в запросах не будет добавлена информация об аутентификации.
- В случае, если при инициализации драйвера не передан никакой объект, отвечающий за генерацию токенов, то применяется общий порядок чтения значений переменных окружения.

Параметризованные запросы

YDB поддерживает и рекомендует к использованию так называемые параметризованные запросы. В таких запросах данные передаются отдельно от самого тела запроса, а в SQL-запросе используются специальные параметры для обозначения местоположения данных. Планы параметризованных запросов кешируются на стороне сервера (по умолчанию — до 1000 записей), что позволяет переиспользовать планы для однотипных запросов, которые отличаются лишь значением параметров.

Во многих YDB SDK кеширование параметризованных запросов включено по умолчанию. При необходимости его можно отключить, установив для параметра `KeepInCache` значение `false`.

Запрос с данными в теле запроса:

```
SELECT sa.title AS season_title, sr.title AS series_title
FROM seasons AS sa INNER JOIN series AS sr ON sa.series_id = sr.series_id
WHERE sa.series_id = 15 AND sa.season_id = 3
```

Соответствующий ему параметризованный запрос:

```
DECLARE $seriesId AS Uint64;
DECLARE $seasonId AS Uint64;

SELECT sa.title AS season_title, sr.title AS series_title
FROM seasons AS sa INNER JOIN series AS sr ON sa.series_id = sr.series_id
WHERE sa.series_id = $seriesId AND sa.season_id = $seasonId
```

Параметризованные запросы записываются в форме шаблона, в котором определенного вида имена заменяются конкретными параметрами при каждом выполнении запроса. Лексемы начинающиеся со знака `$` такие, как `$seriesId` и `$seasonId` в запросе выше, используются для обозначения параметров.

Параметризованные запросы обеспечивают следующие преимущества:

- При повторяющихся запросах сервер базы данных имеет возможность кешировать план запроса для параметризованных запросов. Это радикально снижает потребление CPU и повышает пропускную способность системы;
- Использование параметризованных запросов спасает от уязвимостей вида SQL Injection.
- Сокращение кода — меняются только аргументы запроса, но не сам запрос. Не требуется ручная конкатенация строки запроса.

Смотрите также

- Параметризованные запросы и повторная компиляция
- Параметризованные запросы

Форматы данных

YDB поддерживает несколько форматов представления данных и режимов возвращения схемы в результате выполнения запроса через QueryService. Формат и режим возвращения схемы задаются при выполнении запроса и распространяются на каждую инструкцию, которая возвращает данные (например, инструкция SELECT или ключевое слово `RETURNING`). Конфигурация этих настроек может отличаться между различными SDK.

Форматы представления данных

В следующих разделах описаны доступные форматы представления данных:

- Protobuf (Value) — формат по умолчанию, данные возвращаются построчно с автоматической конвертацией в нативные типы языка программирования.
- Apache Arrow — колоночный формат для аналитических задач с высокой производительностью при передаче больших объёмов данных.

Режимы возвращения схемы

При потоковом выполнении запроса результаты передаются частями. Режим возвращения схемы определяет, в каких частях потока содержится информация о схеме данных.

- **Always (по умолчанию)** — схема данных возвращается в каждой части потока результата. Удобен для простой обработки, когда каждая часть обрабатывается независимо.
- **First only** — схема данных возвращается только в первой части потока для каждого результирующего набора, в последующих частях она отсутствует. Позволяет сократить объём передаваемых метаданных при потоковой обработке больших результатов.

Состав возвращаемой схемы зависит от формата данных и описан в разделе каждого формата.

Формат Protobuf (Value)

Формат по умолчанию. Данные возвращаются построчно в формате Protobuf: каждая строка представляет собой сериализованный набор именованных значений с их YQL-типами. SDK автоматически преобразует полученные значения в нативные типы выбранного языка программирования, предоставляя типобезопасный интерфейс для работы с данными.

Этот формат рекомендуется для:

- Транзакционных (OLTP) задач, где требуется точечное чтение отдельных строк таблицы;
- Приложений, которые обрабатывают небольшие фрагменты таблицы в полном размере;
- Случаев, когда важна нативная интеграция с типами выбранного языка программирования.

Конвертация YQL-типов

Конвертация YQL-типов выполняется в нативные типы языка программирования на стороне SDK и зависит от конкретной его реализации. Подробнее о маппинге типов смотрите в документации используемого SDK.

Схема возвращаемых данных

Схема содержит список столбцов с их YQL-типами. Эта информация достаточна для интерпретации значений на стороне SDK: зная тип столбца, SDK преобразует каждое значение в нативный тип языка программирования.

Примеры использования в SDK

C++

Данный формат соответствует `NYdb::TResultSet::EFormat::Value`. Он используется по умолчанию на стороне сервиса, если формат не задан; ниже — явное указание, как в примере на Python. Сессию `NYdb::NQuery::TSession` обычно получают в колбэке `TQueryClient::RetryQuerySync` / `RetryQuery`.

```
#include <ydb-cpp-sdk/client/query/client.h>
#include <ydb-cpp-sdk/client/types/status/status.h>

NYdb::TStatus ExampleValue(NYdb::NQuery::TSession session) {
    constexpr std::string_view query = "SELECT * FROM example ORDER BY Key LIMIT 100;";

    auto settings = NYdb::NQuery::TExecuteQuerySettings()
        .Format(NYdb::TResultSet::EFormat::Value)
        .SchemaInclusionMode(NYdb::NQuery::ESchemaInclusionMode::FirstOnly);

    auto queryResult = session.ExecuteQuery(
        query,
        NYdb::NQuery::TTxControl::BeginTx().CommitTx(),
        settings).GetValueSync();

    NYdb::NStatusHelpers::ThrowOnError(queryResult);

    for (const NYdb::TResultSet& resultSet : queryResult.GetResultSets()) {
        std::cout << "Rows: " << resultSet.RowsCount() << ", columns: " << resultSet.ColumnsCount() << std::e
ndl;
    }
}
```

Python

Данный формат используется по умолчанию при выполнении запроса через QueryService. Ниже приведён пример явного указания формата возвращаемых данных.

```
pool = ydb.QuerySessionPool(driver)

query = """
    SELECT * FROM example ORDER BY Key LIMIT 100;
"""

result = pool.execute_with_retries(
    query,
    result_set_format=ydb.QueryResultSetFormat.VALUE,
    schema_inclusion_mode=ydb.QuerySchemaInclusionMode.FIRST_ONLY,
)

for result_set in result:
    print(f"Record batch with {len(result_set.rows)} rows and {len(result_set.columns)} columns")
```

Java

Данный формат используется по умолчанию при выполнении запроса через QueryService.

C#

Данный формат используется по умолчанию.

Формат Apache Arrow

Данные возвращаются в колоночном формате Apache Arrow (стандарт IPC версии 5.0) и не преобразуются на стороне SDK, что позволяет эффективно обрабатывать большие объёмы данных.

Этот формат рекомендуется для:

- Аналитических (OLAP) задач, где данные обрабатываются колоночно — агрегации, фильтрации, сканирование по нескольким столбцам больших выборок;
- Систем, которые нативно работают с Apache Arrow;
- Задач, где важна высокая производительность при передаче больших объёмов данных.

Конвертация YQL-типов

Типы данных YQL конвертируются в типы Apache Arrow по следующим правилам.

Числовые типы

Тип YQL	Тип Arrow	Примечание
`Bool`	`uint8`	Значения `true` / `false` кодируются как `1` / `0`
`Int8`	`int8`	
`Int16`	`int16`	
`Int32`	`int32`	
`Int64`	`int64`	
`Uint8`	`uint8`	
`Uint16`	`uint16`	
`Uint32`	`uint32`	
`Uint64`	`uint64`	
`Float`	`float`	
`Double`	`double`	
`Decimal(p, s)`	`fixed_size_binary(16)`	Используются 120 бит, есть маркеры ±inf и NaN
`DyNumber`	`string`	Строковое представление числа

Строковые типы

Тип YQL	Тип Arrow	Примечание
`String`	`binary`	
`Utf8`	`string`	
`Json`	`string`	
`JsonDocument`	`string`	Строковое представление бинарного JSON
`Yson`	`binary`	
`Uuid`	`fixed_size_binary(16)`	16 байт в mixed-endian порядке

Временные типы

Тип YQL	Тип Arrow	Примечание
`Date`	`uint16`	Точность до дня
`Date32`	`int32`	Точность до дня
`Datetime`	`uint32`	Точность до секунды
`Datetime64`	`int64`	Точность до секунды
`Timestamp`	`uint64`	Точность до микросекунды
`Timestamp64`	`int64`	Точность до микросекунды
`Interval`	`int64`	Микросекунды
`Interval64`	`int64`	Микросекунды
`TzDate`	`struct<datetime: uint16, timezone: string>`	Включает строковое имя метки таймзоны
`TzDate32`	`struct<datetime: int32, timezone: string>`	Включает строковое имя метки таймзоны
`TzDatetime`	`struct<datetime: uint32, timezone: string>`	Включает строковое имя метки таймзоны
`TzDatetime64`	`struct<datetime: int64, timezone: string>`	Включает строковое имя метки таймзоны

`TzTimestamp`	`struct<datetime: uint64, timezone: string>`	Включает строковое имя метки таймзоны
`TzTimestamp64`	`struct<datetime: int64, timezone: string>`	Включает строковое имя метки таймзоны

Примечание

В формате Arrow базовые временные типы данных представляют собой беззнаковые целочисленные типы, в отличие от расширенных временных типов, которые являются знаковыми.

Контейнерные типы

Тип YQL	Тип Arrow	Примечание
`List<T>`	`list<item: T>`	
`Tuple<T1, T2, ...>`	`struct<field0: T1, field1: T2, ...>`	
`Struct<name: T, ...>`	`struct<name: T, ...>`	
`Dict<K, V>`	`list<struct<key: K, payload: V>>`	
`Set<T>`	`list<struct<key: T, payload: struct<>>>`	Является `Dict<T, Void>`
`Variant<T1, ..., Tn>`	`dense_union<field0: T1, ...>`	Для n <= 128
`Variant<T1, ..., Tn>`	`dense_union<dense_union<field0: T1, ...>, ...>`	Для 128 < n <= 16384
`Variant<name1: T1, ..., nameN: Tn>`	`dense_union<name1: T1, ...>`	Для n <= 128
`Variant<name1: T1, ..., nameN: Tn>`	`dense_union<dense_union<name1: T1, ...>, ...>`	Для 128 < n <= 16384

Важно

Типы `Variant` над кортежем и над структурой не представимы в формате Apache Arrow, если количество дочерних типов превышает 16384 (128 * 128).

Опциональные и специальные типы

Тип YQL	Тип Arrow	Примечание
`Null`	`null`	Сингулярный тип
`Void`	`struct<>`	Сингулярный тип
`EmptyList`	`struct<>`	Сингулярный тип
`EmptyDict`	`struct<>`	Сингулярный тип
`Tagged<T>`	`T`	Раскрытие с потерей именования
`Optional<T>`	`struct<opt: T>`	Если тип `T` является `Variant`, `Optional`, `Pg` или сингулярным
`Optional<T>`	`T`	Для остальных типов

Типы семейства pg

Все типы семейства `pg` представляются типом Arrow `string` как текстовое представление значений.

Сжатие данных

Для формата Apache Arrow можно настроить сжатие передаваемых данных. Доступны следующие кодеки:

Кодек	Описание
Без сжатия (по умолчанию)	
`ZSTD`	Сжатие Zstandard. Поддерживает настройку уровня сжатия
`LZ4_FRAME`	Сжатие LZ4. Настройка уровня сжатия не поддерживается

Схема возвращаемых данных

Схема содержит два компонента:

- **Список столбцов с YQL-типами** — описывает исходные типы данных в терминах YQL. Позволяет приложению понять семантику данных независимо от особенностей Arrow-представления.
- **Сериализованная схема Arrow RecordBatch** — описывает структуру полученных бинарных данных в терминах Apache Arrow. Необходима для корректной десериализации RecordBatch на стороне клиента.

Наличие двух схем обусловлено тем, что YQL-типы и Arrow-типы не всегда имеют взаимно однозначное соответствие.

Примеры использования Apache Arrow в SDK

C++

В настройках выполнения запроса задаётся формат результата `Arrow`, режим включения схемы `FirstOnly` и (по аналогии с примером на Python) сжатие ZSTD с уровнем 10. Сериализованная схема IPC и бинарные RecordBatch читаются через

`NYdb::TArrowAccessor` — этот API помечен как экспериментальный; дальнейшую десериализацию выполняют средствами Apache Arrow C++. Сессию `NYdb::NQuery::TSession` обычно получают в колбэке `TQueryClient::RetryQuerySync` / `RetryQuery`.

```
#include <ydb-cpp-sdk/client/arrow/accessor.h>
#include <ydb-cpp-sdk/client/query/client.h>

NYdb::TStatus ExampleArrow(NYdb::NQuery::TSession session) {
    constexpr std::string_view query = "SELECT * FROM example ORDER BY Key LIMIT 100;";

    auto settings = NYdb::NQuery::TExecuteQuerySettings()
        .Format(NYdb::TResultSet::EFormat::Arrow)
        .SchemaInclusionMode(NYdb::NQuery::ESchemaInclusionMode::FirstOnly)
        .ArrowFormatSettings(NYdb::NQuery::TArrowFormatSettings()
            .CompressionCodec(NYdb::NQuery::TArrowFormatSettings::TCompressionCodec()
                .Type(NYdb::NQuery::TArrowFormatSettings::TCompressionCodec::EType::Zstd)
                .Level(10)
            )
        );

    auto queryResult = session.ExecuteQuery(
        query,
        NYdb::NQuery::TTxControl::BeginTx().CommitTx(),
        settings).GetValueSync();

    NYdb::NStatusHelpers::ThrowOnError(queryResult);

    for (const NYdb::TResultSet& resultSet : queryResult.GetResultSets()) {
        const std::string& schema = NYdb::TArrowAccessor::GetArrowSchema(resultSet);
        const std::vector<std::string>& batches = NYdb::TArrowAccessor::GetArrowBatches(resultSet);
        std::cout << "Arrow schema size: " << schema.size() << ", batches: " << batches.size()
                  << std::endl;
    }
}
```

Python

```
import pyarrow

pool = ydb.QuerySessionPool(driver)

query = """
    SELECT * FROM example ORDER BY Key LIMIT 100;
"""

format_settings = ydb.ArrowFormatSettings(
    compression_codec=ydb.ArrowCompressionCodec(ydb.ArrowCompressionCodecType.ZSTD, 10)
)

result = pool.execute_with_retries(
    query,
    result_set_format=ydb.QueryResultSetFormat.ARROW,
    schema_inclusion_mode=ydb.QuerySchemaInclusionMode.FIRST_ONLY,
    arrow_format_settings=format_settings,
)

for result_set in result:
    schema = pyarrow.ipc.read_schema(pyarrow.py_buffer(result_set.arrow_format_meta.schema))
    batch = pyarrow.ipc.read_record_batch(pyarrow.py_buffer(result_set.data), schema)
    print(f"Record batch with {batch.num_rows} rows and {batch.num_columns} columns")
```

Java

```
String query = "SELECT * FROM example ORDER BY Key LIMIT 100;";

ExecuteQuerySettings settings = ExecuteQuerySettings.newBuilder()
        .useApacheArrowFormat(ApacheArrowFormat.zstd())
        .build();

try (RootAllocator allocator = new RootAllocator()) {
    retryCtx.supplyResult(session -> session
            .createQuery(query, TxMode.SERIALIZABLE_RW, Params.empty(), settings)
            .execute(new ApacheArrowCompressedPartsHandler(allocator) {
                @Override
                public void onNextPart(QueryResultPart part) {
                    ResultSetReader rs = part.getResultSetReader();
                    while (rs.next()) {
                        String key = rs.getColumn("Key").getText();
                        System.out.println("Read row with key " + key);
                    }
                    System.out.printf("Record batch with %d rows and %d columns%n",
                            rs.getRowCount(), rs.getColumnCount());
                }
            })
    ).join().getStatus().expectSuccess("execute query problem");
}
```

C#

Функциональность на данный момент не поддерживается.

Работа с топиками

В этой статье приведены примеры использования YDB SDK для работы с топиками.

Перед выполнением примеров создайте топик и добавьте читателя.

Примеры работы с топиками

C++

Пример читателя на GitHub

Go

Примеры на GitHub

Java

Примеры на GitHub

Python

Примеры на GitHub

C#

Примеры на GitHub

JavaScript

Примеры на GitHub

Rust

Примеры на GitHub (`topic-writer`, `topic-reader-retry`, `topic-read-in-transaction-example`).

PHP

Функциональность на данный момент не поддерживается.

Инициализация соединения с топиками

C++

Для работы с топиками создаются экземпляры драйвера YDB и клиента.

Драйвер YDB отвечает за взаимодействие приложения и YDB на транспортном уровне. Драйвер должен существовать на всем протяжении жизненного цикла работы с топиками и должен быть инициализирован перед созданием клиента.

Клиент сервиса топиков (исходный код) работает поверх драйвера YDB и отвечает за управляющие операции с топиками, а также создание сессий чтения и записи.

Фрагмент кода приложения для инициализации драйвера YDB:

```
// Create driver instance.
auto driverConfig = NYdb::TDriverConfig()
    .SetEndpoint(opts.Endpoint)
    .SetDatabase(opts.Database)
    .SetAuthToken(std::getenv("YDB_TOKEN"));

NYdb::TDriver driver(driverConfig);
```

В этом примере используется аутентификационный токен, сохранённый в переменной окружения `YDB_TOKEN`. Подробнее про соединение с БД и аутентификацию.

Фрагмент кода приложения для создания клиента:

```
NYdb::NTopic::TTopicClient topicClient(driver);
```

Go

Для работы с топиками используется экземпляр драйвера YDB, созданный с помощью `ydb.Open`. Клиент топиков доступен через метод `db.Topic()`.

```
package main

import (
  "context"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/topic/topicoptions"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
```

```
  // db.Topic() — клиент для работы с топиками
  writer, err := db.Topic().StartWriter("topic-path")
  if err != nil {
    panic(err)
  }

  reader, err := db.Topic().StartReader("consumer-name",
    topicoptions.ReadTopic("topic-path"),
  )
  if err != nil {
    panic(err)
  }
  _ = writer
  _ = reader
}
```

Java

Для работы с топиками создаются экземпляры транспорта YDB и клиента.

Транспорт YDB отвечает за взаимодействие приложения и YDB на транспортном уровне. Он должен существовать на всем протяжении жизненного цикла работы с топиками и должен быть инициализирован перед созданием клиента.

Фрагмент кода приложения для инициализации транспорта YDB:

```
try (GrpcTransport transport = GrpcTransport.forConnectionString(connString)
        .withAuthProvider(CloudAuthHelper.getAuthProviderFromEnviron())
        .build()) {
    // Use YDB transport
}
```

В этом примере используется вспомогательный метод `CloudAuthHelper.getAuthProviderFromEnviron()`, получающий токен из переменных окружения.
Например, `YDB_ACCESS_TOKEN_CREDENTIALS`.
Подробнее про соединение с БД и аутентификацию.

Клиент сервиса топиков (исходный код) работает поверх транспорта YDB и отвечает как за управляющие операции с топиками, так и за создание писателей и читателей.

Фрагмент кода приложения для создания клиента:

```
try (TopicClient topicClient = TopicClient.newClient(transport)
              .setCompressionExecutor(compressionExecutor)
              .build()) {
  // Use topic client
}
```

В обоих примерах кода выше используется блок (try-with-resources).
Это позволяет автоматически закрывать клиент и транспорт при выходе из этого блока, т.к. оба являются наследниками `AutoCloseable`.

C#

Для работы с топиками достаточно передать строку подключения напрямую в конструктор нужного клиента.

В этом примере используется анонимная аутентификация. Подробнее про соединение с базой данных и аутентификацию.

Фрагмент кода приложения для создания различных клиентов к топикам:

```
const string connectionString = "Host=localhost;Port=2136;Database=/local";

await using var topicClient = new TopicClient(connectionString);

await using var writer = new WriterBuilder<string>(connectionString, topicName)
{
    ProducerId = "ProducerId_Example"
}.Build();

await using var reader = new ReaderBuilder<string>(connectionString)
{
    ConsumerName = "Consumer_Example",
    SubscribeSettings = { new SubscribeSettings(topicName) }
}.Build();
```

Python

Для работы с топиками создаётся экземпляр драйвера YDB. Клиент топиков доступен через атрибут `topic_client` и используется для управляющих операций с топиками, а также создания писателей и читателей.

Native SDK

```
import os
import ydb

driver_config = ydb.DriverConfig(
    endpoint=os.environ["YDB_ENDPOINT"],
    database=os.environ["YDB_DATABASE"],
)
driver = ydb.Driver(driver_config)
```

```
driver.wait(timeout=5)
# driver.topic_client — клиент для работы с топиками
writer = driver.topic_client.writer(topic_path)
reader = driver.topic_client.reader(topic=topic_path, consumer=consumer_name)
```

Native SDK (Asyncio)

```
import os
import ydb

driver_config = ydb.DriverConfig(
    endpoint=os.environ["YDB_ENDPOINT"],
    database=os.environ["YDB_DATABASE"],
)
async with ydb.aio.Driver(driver_config) as driver:
    await driver.wait(timeout=5)
    # driver.topic_client — клиент для работы с топиками
    writer = driver.topic_client.writer(topic_path)
    reader = driver.topic_client.reader(topic=topic_path, consumer=consumer_name)
```

Подробнее про соединение с БД и аутентификацию.

JavaScript

```
const t = topic(driver);

await using reader = t.createReader({
  topic: "/Root/demo-topic",
  consumer: "demo-consumer",
});

await using writer = t.createWriter({
  topic: "/Root/demo-topic",
  producer: "demo-producer",
});
```

Rust

```
use ydb::{ClientBuilder, YdbResult};

#[tokio::main]
async fn main() -> YdbResult<()> {
    let client = ClientBuilder::new_from_connection_string("grpc://localhost:2136/local")?.client()?;
    client.wait().await?;
    let mut topic_client = client.topic_client();
    // topic_client.create_reader(...), create_writer_with_params(...), ...
    Ok(())
}
```

PHP

Функциональность на данный момент не поддерживается.

Управление топиками

Создание топика

Единственный обязательный параметр для создания топика - это его путь, остальные параметры опциональны.

C++

Полный список настроек можно посмотреть в заголовочном файле.

Пример создания топика с тремя партициями и поддержкой кодека ZSTD:

```
auto settings = NYdb::NTopic::TCreateTopicSettings()
    .PartitioningSettings(3, 3)
    .AppendSupportedCodecs(NYdb::NTopic::ECodec::ZSTD);

auto status = topicClient
    .CreateTopic("my-topic", settings)  // returns TFuture<TStatus>
    .GetValueSync();
```

Go

Полный список поддерживаемых параметров можно посмотреть в документации SDK.

Пример создания топика со списком поддерживаемых кодеков и минимальным количеством партиций

```
err := db.Topic().Create(ctx, "topic-path",
  // optional
  topicoptions.CreateWithSupportedCodecs(topictypes.CodecRaw, topictypes.CodecGzip),

  // optional
  topicoptions.CreateWithMinActivePartitions(3),
)
```

Python

Пример создания топика со списком поддерживаемых кодеков и минимальным количеством партиций

Native SDK

```
driver.topic_client.create_topic(topic_path,
    supported_codecs=[ydb.TopicCodec.RAW, ydb.TopicCodec.GZIP], # optional
    min_active_partitions=3,                                    # optional
)
```

Native SDK (Asyncio)

```
await driver.topic_client.create_topic(topic_path,
    supported_codecs=[ydb.TopicCodec.RAW, ydb.TopicCodec.GZIP],  # optional
    min_active_partitions=3,                                     # optional
)
```

Java

Полный список настроек можно посмотреть в коде SDK.

Пример создания топика со списком поддерживаемых кодеков и минимальным количеством партиций

```
topicClient.createTopic(topicPath, CreateTopicSettings.newBuilder()
                // Optional
                .setSupportedCodecs(SupportedCodecs.newBuilder()
                        .addCodec(Codec.RAW)
                        .addCodec(Codec.GZIP)
                        .build())
                // Optional
                .setPartitioningSettings(PartitioningSettings.newBuilder()
                        .setMinActivePartitions(3)
                        .build())
                .build());
```

C#

Пример создания топика со списком поддерживаемых кодеков и минимальным количеством партиций:

```
await topicClient.CreateTopic(new CreateTopicSettings
{
    Path = topicName,
    Consumers = { new Consumer("Consumer_Example") },
    SupportedCodecs = { Codec.Raw, Codec.Gzip },
    PartitioningSettings = new PartitioningSettings
    {
        MinActivePartitions = 3
    }
});
```

JavaScript

```
const topicService = driver.createClient(TopicServiceDefinition);
await topicService.createTopic(
  create(CreateTopicRequestSchema, {
    path: "/path-to-my-topic",
    partitioningSettings: {
      minActivePartitions: 1n,
      maxActivePartitions: 100n,
    },
    consumers: [{ name: "my-consumer" }],
  }),
);
```

Rust

```
use ydb::{Codec, CreateTopicOptionsBuilder, YdbResult};

topic_client
    .create_topic(
        "/local/my-topic".into(),
        CreateTopicOptionsBuilder::default()
            .min_active_partitions(3)
            .supported_codecs(vec![Codec::Raw, Codec::Zstd])
            .build()?,
    )
    .await?;
```

PHP

Функциональность на данный момент не поддерживается.

Изменение топика

C++

При изменении топика в параметрах метода `AlterTopic` нужно указать путь топика и параметры, которые будут изменяться. Изменяемые параметры представлены структурой `TAlterTopicSettings`.

Полный список настроек можно посмотреть в заголовочном файле.

Пример добавления важного читателя к топику и установки времени хранения сообщений для топика в два дня:

```
auto alterSettings = NYdb::NTopic::TAlterTopicSettings()
    .BeginAddConsumer("my-consumer")
        .Important(true)
    .EndAddConsumer()
    .SetRetentionPeriod(TDuration::Days(2));

auto status = topicClient
    .AlterTopic("my-topic", alterSettings)  // returns TFuture<TStatus>
    .GetValueSync();
```

Go

При изменении топика в параметрах нужно указать путь топика и те параметры, которые будут изменяться.

Полный список поддерживаемых параметров можно посмотреть в документации SDK.

Пример добавления читателя к топику

```
err := db.Topic().Alter(ctx, "topic-path",
  topicoptions.AlterWithAddConsumers(topictypes.Consumer{
    Name:            "new-consumer",
    SupportedCodecs: []topictypes.Codec{topictypes.CodecRaw, topictypes.CodecGzip}, // optional
  }),
)
```

Python

Пример изменения списка поддерживаемых кодеков и минимального количества партиций у топика

Native SDK

```
driver.topic_client.alter_topic(topic_path,
    set_supported_codecs=[ydb.TopicCodec.RAW, ydb.TopicCodec.GZIP], # optional
    set_min_active_partitions=3,                                    # optional
)
```

Native SDK (Asyncio)

```
await driver.topic_client.alter_topic(topic_path,
    set_supported_codecs=[ydb.TopicCodec.RAW, ydb.TopicCodec.GZIP],  # optional
    set_min_active_partitions=3,                                     # optional
)
```

Java

При изменении топика в параметрах метода `alterTopic` нужно указать путь топика и параметры, которые будут изменяться.

Полный список настроек можно посмотреть в коде SDK.

```
topicClient.alterTopic(topicPath, AlterTopicSettings.newBuilder()
                .addAddConsumer(Consumer.newBuilder()
                        .setName("new-consumer")
                        .setSupportedCodecs(SupportedCodecs.newBuilder()
                                .addCodec(Codec.RAW)
                                .addCodec(Codec.GZIP)
                                .build())
                        .build())
                .build());
```

C#

Функциональность на данный момент не поддерживается.

JavaScript

```
const topicService = driver.createClient(TopicServiceDefinition);
await topicService.alterTopic(
  create(AlterTopicRequestSchema, {
    path: "/path-to-my-topic",
    addConsumers: [{ name: "my-consumer-2" }],
  }),
);
```

Rust

```
use ydb::{AlterTopicOptionsBuilder, YdbResult};

topic_client
    .alter_topic(
        "/local/my-topic".into(),
        AlterTopicOptionsBuilder::default()
            .set_min_active_partitions(Some(5))
            .build()?,
    )
    .await?;
```

PHP

Функциональность на данный момент не поддерживается.

Получение информации о топике

C++

Для получения информации о топике используется метод `DescribeTopic`.

Описание топика представлено структурой `TTopicDescription`.

Полный список полей описания смотри в заголовочном файле.

Получить доступ к этому описанию можно так:

```
auto result = topicClient.DescribeTopic("my-topic").GetValueSync();
if (result.IsSuccess()) {
    const auto& description = result.GetTopicDescription();
    std::cout << "Topic description: " << GetProto(description) << std::endl;
}
```

Существует отдельный метод для получения информации о читателе - `DescribeConsumer`.

Go

```
  descResult, err := db.Topic().Describe(ctx, "topic-path")
if err != nil {
  log.Fatalf("failed describe topic: %v", err)
  return
}
fmt.Printf("describe: %#v\n", descResult)
```

Python

Native SDK

```
info = driver.topic_client.describe_topic(topic_path)
print(info)
```

Native SDK (Asyncio)

```
info = await driver.topic_client.describe_topic(topic_path)
print(info)
```

Java

Для получения информации о топике используется метод `describeTopic`.

Полный список полей описания можно посмотреть в коде SDK.

```
Result<TopicDescription> topicDescriptionResult = topicClient.describeTopic(topicPath)
        .join();
TopicDescription description = topicDescriptionResult.getValue();
```

C#

Функциональность на данный момент не поддерживается.

JavaScript

```
const topicService = driver.createClient(TopicServiceDefinition);
await topicService.describeTopic(
  create(DescribeTopicRequestSchema, {
    path: "/path-to-my-topic",
  }),
);
```

Rust

```
use ydb::{DescribeTopicOptionsBuilder, YdbResult};
```

```
let description = topic_client
    .describe_topic(
        "/local/my-topic".into(),
        DescribeTopicOptionsBuilder::default().include_stats(true).build()?,
    )
    .await?;
```

PHP

Функциональность на данный момент не поддерживается.

Удаление топика

Для удаления топика достаточно указать путь к нему.

C++

```
auto status = topicClient.DropTopic("my-topic").GetValueSync();
```

Go

```
  err := db.Topic().Drop(ctx, "topic-path")
```

Python

Native SDK

```
driver.topic_client.drop_topic(topic_path)
```

Native SDK (Asyncio)

```
await driver.topic_client.drop_topic(topic_path)
```

Java

```
topicClient.dropTopic(topicPath);
```

C#

```
await topicClient.DropTopic(topicName);
```

JavaScript

```
const topicService = driver.createClient(TopicServiceDefinition);
await topicService.dropTopic(
  create(DropTopicRequestSchema, {
    path: "/path-to-my-topic",
  }),
);
```

Rust

```
topic_client.drop_topic("/local/my-topic".into()).await?;
```

PHP

Функциональность на данный момент не поддерживается.

Запись сообщений

Подключение к топику для записи сообщений

На данный момент поддерживается подключение только с совпадающими идентификаторами источника и группы сообщений (`producer_id` и `message_group_id`), в будущем это ограничение будет снято.

C++

В C++ SDK для записи в топик доступно три варианта API. Базовые настройки (буферизация, кодеки, ретраи) одинаковы у всех трёх и заданы структурой `TWriteSessionSettings`, поэтому ниже подсказки сосредоточены на отличиях и сценариях использования.

- `IWriteSession` — низкоуровневая сессия записи в одну партицию с полным набором возможностей: цикл событий (`TReadyToAcceptEvent`, `TAcksEvent`, `TSessionClosedEvent`), явное управление потоком отправки через `TContinuationToken`, которые система выдаёт перед приёмом каждого сообщения; независимые подтверждения для каждого сообщения; отправка предварительно сжатых данных через метод `WriteEncoded`, минуя повторное сжатие на сервере. Структура `TWriteSessionSettings` определена именно здесь — два других варианта её переиспользуют. Подходит, когда нужен статус каждого сообщения, кастомная асинхронная логика или ручное управление сжатием.
- `ISimpleBlockingWriteSession` — синхронный fire-and-forget API для записи в одну партицию топика, самый простой вариант. Метод `Write(message, blockTimeout)` кладёт сообщение во внутренний буфер, отправка на сервер идёт в фоне. В нормальном режиме вызов возвращается мгновенно и блокируется только при переполнении буфера (по `MaxMemoryUsage` / `MaxInflightCount`), не дольше `blockTimeout`. Возврат `false` означает, что сообщение **не попало в буфер** и потеряно. Подтверждений по отдельным сообщениям нет; убедиться, что весь буфер доставлен на сервер,

можно только вызвав `Close()` — он ждёт ack от сервера. Подходит, когда достаточно гарантии «всё или ничего» к моменту закрытия сессии и нужен простой синхронный код.

- `IProducer` — высокоуровневый API поверх нескольких сессий записи: прозрачно шардирует сообщения по партициям топика по ключу. Вдохновлён интерфейсом Producer из Apache Kafka, но учитывает особенности YDB и при работе с топиками с автопартиционированием даёт полные гарантии порядка и exactly-once. Подтверждения от сервера доступны через обработчик `AcksHandler`; дождаться доставки накопленного буфера — `Flush()`, корректно завершить работу — `Close()`. Подходит, когда нужно писать в **многопартиционный** топик с маршрутизацией по ключу.

IWriteSession

`IWriteSession` — базовая сессия записи, от которой наследуют настройки другие варианты записи. Настройки сессии записи представлены структурой `TWriteSessionSettings`, для варианта `ISimpleBlockingWriteSession` часть настроек не поддерживается.

Полный список настроек смотри в заголовочном файле.

```
std::string producerAndGroupID = "group-id";
auto settings = NYdb::NTopic::TWriteSessionSettings()
    .Path("my-topic")
    .ProducerId(producerAndGroupID)
    .MessageGroupId(producerAndGroupID);

auto session = topicClient.CreateWriteSession(settings);
```

ISimpleBlockingWriteSession

`ISimpleBlockingWriteSession` — простой синхронный вариант сессии записи `IWriteSession` для записи по одному сообщению без подтверждения по каждому сообщению. Метод `Write` блокируется при превышении числа inflight записей или размера буфера SDK. Настройки сессии записи представлены структурой `TWriteSessionSettings`, как и в случае `IWriteSession`, однако часть настроек не поддерживается.

Полный список настроек смотри в заголовочном файле.

```
std::string producerAndGroupID = "group-id";
auto settings = NYdb::NTopic::TWriteSessionSettings()
    .Path("my-topic")
    .ProducerId(producerAndGroupID)
    .MessageGroupId(producerAndGroupID);

auto session = topicClient.CreateSimpleBlockingWriteSession(settings);
```

IProducer

`IProducer` — высокоуровневый API поверх сессий записи: один объект скрывает управление несколькими сессиями и автоматически выбирает партицию по ключу сообщения. Настройки продюсера представлены структурой `TProducerSettings`, которая наследует `TWriteSessionSettings`, поэтому общие настройки записи совпадают с `IWriteSession`.

Настройки задаются через `TProducerSettings`:

- `ProducerIdPrefix` — префикс producer id для подсессий записи;
- `PartitionChooserStrategy` — стратегия выбора партиции по ключу сообщения:
  - `Bound` — ключ сопоставляется с диапазонами партиций топика (`FromBound` / `ToBound` из описания топика). По умолчанию перед сопоставлением ключ проходит через MurmurHash64. Рекомендуется для топиков с автопартиционированием: при разделении партиции SDK обновляет границы и продолжает направлять сообщения с тем же ключом в корректный диапазон.
  - `KafkaHash` — по аналогии с Kafka: от ключа считается MurmurHash, индекс партиции — остаток от деления хеша на число партиций. Удобно при миграции с Kafka. Не поддерживается при включённом автопартиционировании.
- `PartitioningKeyHasher` — функция преобразования ключа перед сопоставлением с диапазонами; используется только для стратегии `Bound`. Можно задать свою, например чтобы в сравнении участвовал исходный ключ без хеширования.

Полный список настроек смотри в заголовочном файле.

```
auto producerSettings = NYdb::NTopic::TProducerSettings()
    .Path("my-topic")
    .ProducerIdPrefix("my-producer")
    .PartitionChooserStrategy(NYdb::NTopic::EPartitionChooserStrategy::Bound);

auto producer = topicClient.CreateProducer(producerSettings);
```

Go

```
producerAndGroupID := "group-id"
writer, err := db.Topic().StartWriter(producerAndGroupID, "topicName",
  topicoptions.WithMessageGroupID(producerAndGroupID),
)
if err != nil {
    return err
}
```

Python

Native SDK

```
writer = driver.topic_client.writer(topic_path)
```

Native SDK (Asyncio)

```
writer = driver.topic_client.writer(topic_path)
```

Java

Синхронный API

Инициализация настроек писателя:

```
String producerAndGroupID = "group-id";
WriterSettings settings = WriterSettings.newBuilder()
      .setTopicPath(topicPath)
      .setProducerId(producerAndGroupID)
      .setMessageGroupId(producerAndGroupID)
      .build();
```

Создание синхронного писателя:

```
SyncWriter writer = topicClient.createSyncWriter(settings);
```

После создания писателя его необходимо инициализировать. Для этого есть два метода:

- `init()`: неблокирующий, запускает процесс инициализации в фоне и не ждёт его завершения.

  ```
  writer.init();
  ```

- `initAndWait()`: блокирующий, запускает процесс инициализации и ждёт его завершения. Если в процессе инициализации возникла ошибка, будет брошено исключение.

  ```
  try {
      writer.initAndWait();
      logger.info("Init finished successfully");
  } catch (Exception exception) {
      logger.error("Exception while initializing writer: ", exception);
      return;
  }
  ```

Асинхронный API

Инициализация настроек писателя:

```
String producerAndGroupID = "group-id";
WriterSettings settings = WriterSettings.newBuilder()
      .setTopicPath(topicPath)
      .setProducerId(producerAndGroupID)
      .setMessageGroupId(producerAndGroupID)
      .build();
```

Создание и инициализация асинхронного писателя:

```
AsyncWriter writer = topicClient.createAsyncWriter(settings);

// Init in background
writer.init()
        .thenRun(() -> logger.info("Init finished successfully"))
        .exceptionally(ex -> {
            logger.error("Init failed with ex: ", ex);
            return null;
        });
```

C#

```
await using var writer = new WriterBuilder<string>(connectionString, topicName)
{
    ProducerId = "ProducerId_Example"
}.Build();
```

JavaScript

```
await using writer = createTopicWriter(driver, {
  topic: topicName,
  producer: producerName,
});
```

Rust

```
use ydb::{TopicWriter, TopicWriterOptionsBuilder, YdbResult};

let writer: TopicWriter = topic_client
    .create_writer_with_params(
```

```
        TopicWriterOptionsBuilder::default()
            .topic_path("/local/my-topic".into())
            .producer_id("group-id".into())
            .message_group_id("group-id".into())
            .build()?,
    )
    .await?;
```

PHP

Функциональность на данный момент не поддерживается.

Запись сообщений

C++

IWriteSession

Работа с объектом `IWriteSession` устроена как обработка цикла событий с тремя типами событий: `TReadyToAcceptEvent`, `TAcksEvent` и `TSessionClosedEvent`.

Для каждого из типов событий можно установить обработчик этого события, а также можно установить общий обработчик. Обработчики устанавливаются в настройках сессии записи перед её созданием.

Если обработчик для некоторого события не установлен, его необходимо получить и обработать в методах `GetEvent` / `GetEvents`. Для неблокирующего ожидания очередного события есть метод `WaitEvent` с интерфейсом `TFuture<void>()`.

Для записи каждого сообщения пользователь должен "потратить" move-only объект `TContinuationToken`, который выдаёт SDK с событием `TReadyToAcceptEvent`. При записи сообщения можно установить пользовательские seqNo и временную метку создания, но по умолчанию их проставляет SDK автоматически.

По умолчанию `Write` выполняется асинхронно — данные из сообщений вычитываются и сохраняются во внутренний буфер, отправка происходит в фоне в соответствии с настройками `MaxMemoryUsage`, `MaxInflightCount`, `BatchFlushInterval`, `BatchFlushSizeBytes`. Сессия сама переподключается к YDB при обрывах связи и повторяет отправку сообщений пока это возможно, в соответствии с настройкой `RetryPolicy`. При получении ошибки, после которой невозможно продолжить работу, сессия записи отправляет пользователю `TSessionClosedEvent` с диагностической информацией.

Так может выглядеть запись нескольких сообщений в цикле событий без использования обработчиков:

```
// Event loop
while (true) {
    // Get event
    // May block for a while if write session is busy
    std::optional<NYdb::NTopic::TWriteSessionEvent::TEvent> event = session->GetEvent(/*block=*/true);

    if (auto* readyEvent = std::get_if<NYdb::NTopic::TWriteSessionEvent::TReadyToAcceptEvent>(&*event)) {
        session->Write(std::move(event.ContinuationToken), "This is yet another message.");

    } else if (auto* ackEvent = std::get_if<NYdb::NTopic::TWriteSessionEvent::TAcksEvent>(&*event)) {
        std::cout << ackEvent->DebugString() << std::endl;

    } else if (auto* closeSessionEvent = std::get_if<NYdb::NTopic::TSessionClosedEvent>(&*event)) {
        break;
    }
}
```

ISimpleBlockingWriteSession

Как упрощённый вариант `IWriteSession`, `ISimpleBlockingWriteSession` пишет через ту же внутреннюю буферизацию, но без цикла событий: не использует `ContinuationToken` и не возвращает подтверждения через `TAcksEvent`. Метод `Write` кладёт сообщение во внутренний буфер; если буфер переполнен, вызов блокируется до появления места. Параметр `blockTimeout` ограничивает время ожидания. Метод возвращает `true`, если сообщение принято в буфер, и `false`, если за отведённое время записать не удалось.

Отправка на сервер, как и у `IWriteSession`, выполняется в фоне. Чтобы дождаться завершения всех записей и закрыть сессию, вызовите `Close()`.

```
auto messageData = std::string("message");
NYdb::NTopic::TWriteMessage writeMessage(messageData);
session->Write(std::move(writeMessage));
```

IProducer

`TProducerSettings` наследует `TWriteSessionSettings`, поэтому буферизация, отправка и переподключение устроены так же, как у `IWriteSession`: `Write` кладёт сообщение во внутренний буфер, отправка на сервер идёт в фоне в соответствии с настройками `MaxMemoryUsage`, `MaxInflightCount`, `BatchFlushInterval`, `BatchFlushSizeBytes`. Продюсер переподключается к YDB при обрывах связи и повторяет отправку, пока это возможно, в соответствии с `RetryPolicy`. При неустранимой ошибке продюсер закрывается; статус и причину можно получить из результата `Write` или `Flush`.

`Flush` дожидается доставки накопленных данных на сервер; `Close` дожидается отправки оставшихся в буфере сообщений и завершает работу продюсера.

```
auto messageData = std::string("order-created");
// First argument is the partitioning key — the SDK chooses a partition by it.
NYdb::NTopic::TWriteMessage writeMessage("user-42", messageData);
producer->Write(std::move(writeMessage));
producer->Flush().GetValueSync();
```

Подробный пример см. в репозитории ydb-platform/ydb.

Go

Для отправки сообщения - достаточно в поле Data сохранить Reader, из которого можно будет прочитать данные. Можно рассчитывать на то что данные каждого сообщения читаются один раз (или до первой ошибки), к моменту возврата из Write данные будут уже прочитаны и сохранены во внутренний буфер.

SeqNo и дата создания сообщений по умолчанию проставляются автоматически.

По умолчанию Write выполняется асинхронно - данные из сообщений вычитываются и сохраняются во внутренний буфер, отправка происходит в фоне. Writer сам переподключается к YDB при обрывах связи и повторяет отправку сообщений пока это возможно. При получении ошибки, после которой невозможно продолжить работу, Writer останавливается и следующие вызовы Write будут завершаться с ошибкой.

```
err := writer.Write(ctx,
  topicwriter.Message{Data: strings.NewReader("1")},
  topicwriter.Message{Data: bytes.NewReader([]byte{1,2,3})},
  topicwriter.Message{Data: strings.NewReader("3")},
)
if err != nil {
  return err
}
```

Для записи по ключу в несколько партиций используйте `WithWriteToManyPartitions(...)` при создании писателя и заполняйте поле `Key` в `topicwriter.Message`.

Стратегии маршрутизации (задаются в `WithWriterPartitionByKey(...)` или `WithWriterPartitionByPartitionID()`):

- `BoundPartitionChooser` — ключ сопоставляется с диапазонами партиций топика (`FromBound` / `ToBound`). По умолчанию перед сопоставлением ключ проходит через MurmurHash64. Рекомендуется для топиков с автопартиционированием: SDK обновляет границы при разделении партиций.
- `KafkaHashPartitionChooser` — по аналогии с Kafka: MurmurHash ключа по модулю числа партиций. Удобно при миграции с Kafka. Не поддерживается при включённом автопартиционировании.
- `WithWriterPartitionByPartitionID` — запись в партицию, указанную в поле `PartitionID` сообщения. Не комбинируется с маршрутизацией по ключу; при split партиции писатель нужно пересоздавать вручную.

```
writer, err := db.Topic().StartWriter(topicPath,
  topicoptions.WithWriteToManyPartitions(
    topicoptions.WithProducerIDPrefix("orders-producer"),
    topicoptions.WithWriterPartitionByKey(topicoptions.BoundPartitionChooser()),
  ),
)
if err != nil {
  return err
}
defer func() { _ = writer.Close(context.Background()) }()

err = writer.Write(ctx, topicwriter.Message{
  Key:  "user-42",
  Data: bytes.NewReader([]byte("order-created")),
})
if err != nil {
  return err
}
```

Подробный пример с маршрутизацией по ключу, альтернативными стратегиями (`KafkaHash` и `PartitionID`) и транзакционным вариантом см. в репозитории ydb-go-sdk.

Python

Для отправки сообщений можно передавать как просто содержимое сообщения (bytes, str), так и вручную задавать некоторые свойства. Объекты можно передавать по одному или сразу в массиве (list). Метод `write` выполняется асинхронно. Возврат из метода происходит сразу после того как сообщения будут положены во внутренний буфер клиента, обычно это происходит быстро. Ожидание может возникнуть, если внутренний буфер уже заполнен и нужно подождать, пока часть данных будет отправлена на сервер.

Native SDK

```
# Простая отправка сообщений, без явного указания метаданных.
# Удобно начинать, удобно использовать пока важно только содержимое сообщения.
writer = driver.topic_client.writer(topic_path)
writer.write("mess")  # Строки будут переданы в кодировке utf-8, так удобно отправлять
                      # текстовые сообщения.
writer.write(bytes([1, 2, 3]))  # Эти байты будут отправлены "как есть", так удобно отправлять
                                # бинарные данные.
writer.write(["mess-1", "mess-2"])  # Здесь за один вызов отправляется несколько сообщений —
                                    # так снижаются накладные расходы на внутренние процессы SDK,
                                    # имеет смысл при большом потоке сообщений.

# Полная форма, используется, когда кроме содержимого сообщения нужно вручную задать и его свойства.
writer = driver.topic_client.writer(topic="topic-path", auto_seqno=False, auto_created_at=False)

writer.write(ydb.TopicWriterMessage("asd", seqno=123, created_at=datetime.datetime.now()))
writer.write(ydb.TopicWriterMessage(bytes([1, 2, 3]), seqno=124, created_at=datetime.datetime.now()))

# В полной форме так же можно отправлять несколько сообщений за один вызов функции.
# Это имеет смысл при большом потоке отправляемых сообщений — для снижения
# накладных расходов на внутренние вызовы SDK.
writer.write([
  ydb.TopicWriterMessage("asd", seqno=123, created_at=datetime.datetime.now()),
```

```
  ydb.TopicWriterMessage(bytes([1, 2, 3]), seqno=124, created_at=datetime.datetime.now(),
  ])
```

Native SDK (Asyncio)

```
writer = driver.topic_client.writer(topic_path)
await writer.write("mess")
await writer.write(bytes([1, 2, 3]))
await writer.write(["mess-1", "mess-2"])
```

Java

Синхронный API

Метод `send` блокирует управление, пока сообщение не будет помещено в очередь отправки.
Попадание сообщения в эту очередь означает, что писатель сделает всё возможное для доставки сообщения.
Например, если сессия записи по какой-то причине оборвётся, писатель переустановит соединение и попробует отправить это сообщение на новой сессии.
Но попадание сообщения в очередь отправки не гарантирует того, что сообщение в итоге будет записано.
Например, могут возникать ошибки, приводящие к завершению работы писателя до того, как сообщения из очереди будут отправлены.
Если нужно подтверждение успешной записи для каждого сообщения, используйте асинхронного писателя и проверяйте статус, возвращаемый методом `send`.

```
writer.send(Message.of("11".getBytes()));

long timeoutSeconds = 5; // How long should we wait for a message to be put into sending buffer
try {
  writer.send(
          Message.newBuilder()
                  .setData("22".getBytes())
                  .setCreateTimestamp(Instant.now().minusSeconds(5))
                  .build(),
          timeoutSeconds,
          TimeUnit.SECONDS
  );
} catch (TimeoutException exception) {
  logger.error("Send queue is full. Couldn't put message into sending queue within {} seconds", timeoutSecond
s);
} catch (InterruptedException | ExecutionException exception) {
  logger.error("Couldn't put the message into sending queue due to exception: ", exception);
}
```

Асинхронный API

Метод `send` в асинхронном клиенте неблокирующий. Помещает сообщение в очередь отправки.
Метод возвращает `CompletableFuture<WriteAck>`, позволяющую проверить, действительно ли сообщение было записано.
В случае, если очередь переполнена, будет брошено исключение QueueOverflowException.
Это способ сигнализировать пользователю о том, что поток записи следует притормозить.
В таком случае стоит или пропускать сообщения, или выполнять повторные попытки записи через exponential backoff.
Также можно увеличить размер клиентского буфера (`setMaxSendBufferMemorySize`), чтобы обрабатывать больший объем сообщений перед тем, как он заполнится.

```
try {
  // Non-blocking. Throws QueueOverflowException if send queue is full
  writer.send(Message.of("33".getBytes()));
} catch (QueueOverflowException exception) {
  // Send queue is full. Need to retry with backoff or skip
}
```

C#

Асинхронная запись сообщения в топик.

```
var asyncWriteTask = writer.WriteAsync("Hello, Example YDB Topics!"); // Task<WriteResult>
```

JavaScript

```
// Пишет сообщение во внутренний буфер
writer.write(Buffer.from("Hello, world!", "utf-8"));

// Для немедленной отправки нужно вызвать flush
await writer.flush();

// Или закрыть писатель
await writer.close();
```

Rust

```
use ydb::{TopicWriterMessageBuilder, YdbResult};

writer
    .write(
        TopicWriterMessageBuilder::default()
            .data(b"payload".to_vec())
```

```
            .build()?,
    )
    .await?;
writer.stop().await?;
```

PHP

Функциональность на данный момент не поддерживается.

Запись сообщений с подтверждением о сохранении на сервере

C++

IWriteSession

Ответы о записи сообщений на сервере приходят клиенту SDK в виде событий `TAcksEvent`. В одном событии могут содержаться ответы о нескольких отправленных ранее сообщениях. Варианты ответа: запись подтверждена (`EES_WRITTEN`), запись отброшена как дубликат ранее записанного сообщения (`EES_ALREADY_WRITTEN`) или запись отброшена по причине сбоя (`EES_DISCARDED`).

Пример установки обработчика `TAcksEvent` для сессии записи:

```
auto settings = NYdb::NTopic::TWriteSessionSettings()
  // other settings are set here
  .EventHandlers(
    NYdb::NTopic::TWriteSessionSettings::TEventHandlers()
      .AcksHandler(
        [&](NYdb::NTopic::TWriteSessionEvent::TAcksEvent& event) {
          for (const auto& ack : event.Acks) {
            if (ack.State == NYdb::NTopic::TWriteSessionEvent::TWriteAck::EEventState::EES_WRITTEN) {
              ackedSeqNo.insert(ack.SeqNo);
              std::cout << "Acknowledged message with seqNo " << ack.SeqNo << std::endl;
            }
          }
        }
      )
  );

auto session = topicClient.CreateWriteSession(settings);
```

В такой сессии записи события `TAcksEvent` не будут приходить пользователю в `GetEvent` / `GetEvents`, вместо этого SDK при получении подтверждений от сервера будет вызывать переданный обработчик. Аналогично можно настраивать обработчики на остальные типы событий.

ISimpleBlockingWriteSession

В отличие от `IWriteSession`, `ISimpleBlockingWriteSession` не возвращает подтверждения по отдельным сообщениям: события `TAcksEvent` и обработчики для них недоступны. Чтобы дождаться, пока все сообщения из буфера будут записаны на сервер, вызовите `Close()` — метод дожидается подтверждения от сервера и закрывает сессию.

IProducer

Подтверждения от сервера приходят так же, как у `IWriteSession`: через обработчик `AcksHandler` в `TProducerSettings::EventHandlers`. Чтобы дождаться доставки накопленного буфера на сервер, вызовите `Flush()`.

```
auto producerSettings = NYdb::NTopic::TProducerSettings()
    .Path("my-topic")
    .ProducerIdPrefix("my-producer")
    .EventHandlers(
        NYdb::NTopic::TWriteSessionSettings::TEventHandlers()
            .AcksHandler([](NYdb::NTopic::TWriteSessionEvent::TAcksEvent& event) {
            .AcksHandler([](NYdb::NTopic::TWriteSessionEvent::TAcksEvent& event) {
                // handle acknowledgements
            })
            })
    );

auto producer = topicClient.CreateProducer(producerSettings);
```

Go

При подключении можно указать опцию синхронной записи сообщений - topicoptions.WithSyncWrite(true). Тогда Write будет возвращаться только после того как получит подтверждение с сервера о сохранении всех, сообщений переданных в вызове. При этом SDK так же как и обычно будет при необходимости переподключаться и повторять отправку сообщений. В этом режиме контекст управляет только временем ожидания ответа из SDK, т.е. даже после отмены контекста SDK продолжит попытки отправить сообщения.

```
producerAndGroupID := "group-id"
writer, _ := db.Topic().StartWriter(producerAndGroupID, "topicName",
  topicoptions.WithMessageGroupID(producerAndGroupID),
  topicoptions.WithSyncWrite(true),
)

err = writer.Write(ctx,
  topicwriter.Message{Data: strings.NewReader("1")},
  topicwriter.Message{Data: bytes.NewReader([]byte{1,2,3})},
  topicwriter.Message{Data: strings.NewReader("3")},
)
if err != nil {
```

```
  return err
}
```

Python

Есть два способа получить подтверждение о записи сообщений на сервере:

- `flush()` — дожидается подтверждения для всех сообщений, записанных ранее во внутренний буфер.
- `write_with_ack(...)` — отправляет сообщение и ждет подтверждение его доставки от сервера. При отправке нескольких сообщений подряд это способ работает медленно.

Native SDK

```
# Положить несколько сообщений во внутренний буфер, затем дождаться,
# пока все они будут доставлены до сервера.
for mess in messages:
    writer.write(mess)

writer.flush()

# Можно отправить несколько сообщений и дождаться подтверждения на всю группу.
writer.write_with_ack(["mess-1", "mess-2"])

# Ожидание при отправке каждого сообщения — этот метод вернет результат только после получения
# подтверждения от сервера.
# Это самый медленный вариант отправки сообщений, используйте его только если такой режим
# действительно нужен.
writer.write_with_ack("message")
```

Native SDK (Asyncio)

```
for mess in messages:
    await writer.write(mess)

await writer.flush()

await writer.write_with_ack(["mess-1", "mess-2"])
await writer.write_with_ack("message")
```

Java

Метод `send` возвращает `CompletableFuture<WriteAck>`. Её успешное завершение означает подтверждение записи сервером.
В структуре `WriteAck` содержится информация о seqNo, offset и статусе записи:

```
writer.send(Message.of(message))
      .whenComplete((result, ex) -> {
          if (ex != null) {
              logger.error("Exception on writing message message: ", ex);
          } else {
              switch (result.getState()) {
                  case WRITTEN:
                      WriteAck.Details details = result.getDetails();
                      StringBuilder str = new StringBuilder("Message was written successfully");
                      if (details != null) {
                          str.append(", offset: ").append(details.getOffset());
                      }
                      logger.debug(str.toString());
                      break;
                  case ALREADY_WRITTEN:
                      logger.warn("Message has already been written");
                      break;
                  default:
                      break;
              }
          }
      });
```

C#

Асинхронная запись сообщения в топик. В случае переполнения внутреннего буфера будет ожидать, когда буфер освободится для повторной отправки.

```
await writer.WriteAsync("Hello, Example YDB Topics!");
```

В случае, если сервер недоступен, сообщения могут накапливаться в очереди в ожидании отправки. Для управления временем ожидания можно использовать токен отмены (`CancellationToken`). Однако, при таком подходе существует вероятность того, что пользователь может отменить отправку уже подтвержденного сообщения.

```
var writeCts = new CancellationTokenSource();
writeCts.CancelAfter(TimeSpan.FromSeconds(3));

await writer.WriteAsync("Hello, Example YDB Topics!", writeCts.Token);
```

JavaScript

Все сообщения записываются во внутренний буфер. Для отправки на сервер есть 3 механизма: два автоматических и один ручной. Ручной - это вызов метода `writer.flush` который возвращает последний seqno записанный на сервере. Автоматическая отправка происходит по условиям:

- Превышение размера внутреннего буфера `maxBufferBytes` (значение по умолчанию = 256MiB).
- По тику интервала периодической отправки `flushIntervalMs` (значение по умолчанию = 10ms).

```
await using writer = createTopicWriter(driver, {
  topic: topicName,
  producer: producerName,
  // Callback that is called when writer receives an acknowledgment for a message.
  onAck: (seqNo, status) => {
    console.log("ACK", seqNo, status);
  },
})

writer.write(Buffer.from("Hello, world!", "utf-8"));

// Чтобы получить последний записанный seqNo на сервере.
await writer.flush();
```

Rust

```
use ydb::{TopicWriterMessageBuilder, YdbResult};

writer
    .write_with_ack(
        TopicWriterMessageBuilder::default()
            .data(b"payload".to_vec())
            .build()?,
    )
    .await?;
```

PHP

Функциональность на данный момент не поддерживается.

Выбор кодека для сжатия сообщений

Подробнее о сжатии данных в топиках.

C++

IWriteSession

Сжатие, которое используется при отправке сообщений методом `Write`, задаётся при создании сессии записи настройками `Codec` и `CompressionLevel`. По умолчанию выбирается кодек GZIP.
Пример создания сессии записи без сжатия сообщений:

```
auto settings = NYdb::NTopic::TWriteSessionSettings()
  // other settings are set here
  .Codec(ECodec::RAW);

auto session = topicClient.CreateWriteSession(settings);
```

Если необходимо в рамках сессии записи отправить сообщение, сжатое другим кодеком, можно использовать метод `WriteEncoded` с указанием кодека и размера расжатого сообщения. Для успешной записи этим способом используемый кодек должен быть разрешён в настройках топика.

ISimpleBlockingWriteSession

Кодек задаётся при создании сессии записи в `TWriteSessionSettings` — те же настройки `Codec` и `CompressionLevel`, что и у `IWriteSession`. Метод `WriteEncoded` недоступен.

```
auto settings = NYdb::NTopic::TWriteSessionSettings()
    // other settings are set here
    .Codec(ECodec::RAW);

auto session = topicClient.CreateSimpleBlockingWriteSession(settings);
```

IProducer

Кодек задаётся в `TProducerSettings` при создании продюсера — те же настройки `Codec` и `CompressionLevel`, что и у `IWriteSession`.

```
auto producerSettings = NYdb::NTopic::TProducerSettings()
    // other settings are set here
    .Codec(NYdb::NTopic::ECodec::RAW);

auto producer = topicClient.CreateProducer(producerSettings);
```

Go

По умолчанию SDK выбирает кодек автоматически (с учетом настроек топика). В автоматическом режиме SDK сначала отправляет по одной группе сообщений каждым из разрешенных кодеков, затем иногда будет пробовать сжать сообщения всеми доступными кодеками и выбирать кодек, дающий наименьший размер сообщения. Если для топика список разрешенных кодеков пуст, то автовыбор производится между Raw и Gzip-кодеками.

При необходимости можно задать фиксированный кодек в опциях подключения. Тогда будет использоваться именно он и замеры проводиться не будут.

```
producerAndGroupID := "group-id"
writer, _ := db.Topic().StartWriter(producerAndGroupID, "topicName",
  topicoptions.WithMessageGroupID(producerAndGroupID),
  topicoptions.WithCodec(topictypes.CodecGzip),
)
```

Python

По умолчанию SDK выбирает кодек автоматически (с учетом настроек топика). В автоматическом режиме SDK сначала отправляет по одной группе сообщений каждым из разрешенных кодеков, затем иногда будет пробовать сжать сообщения всеми доступными кодеками и выбирать кодек, дающий наименьший размер сообщения. Если для топика список разрешенных кодеков пуст, то автовыбор производится между Raw и Gzip-кодеками.

При необходимости можно задать фиксированный кодек в опциях подключения. Тогда будет использоваться именно он и замеры проводиться не будут.

```
writer = driver.topic_client.writer(topic_path,
    codec=ydb.TopicCodec.GZIP,
)
```

Java

```
String producerAndGroupID = "group-id";
WriterSettings settings = WriterSettings.newBuilder()
        .setTopicPath(topicPath)
        .setProducerId(producerAndGroupID)
        .setMessageGroupId(producerAndGroupID)
        .setCodec(Codec.ZSTD)
        .build();
```

C#

Функциональность на данный момент не поддерживается.

JavaScript

```
await using writer = t.createWriter({
  codec: Codec.RAW,
});

await using writer = t.createWriter({
  codec: Codec.GZIP,
});

await using writer = t.createWriter({
  codec: Codec.LZOP,
});

await using writer = t.createWriter({
  codec: 10000, // CUSTOM (допустимый диапазон: 10000–19999)
});
```

Rust

Выбор кодека сжатия при записи в Rust SDK пока недоступен; сообщения отправляются с кодеком `Raw`.

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#341

PHP

Функциональность на данный момент не поддерживается.

Запись сообщений без дедупликации

Подробнее о записи без дедупликации — в соответствующем разделе концепций.

C++

IWriteSession

Если в настройках сессии записи не указывается опция `ProducerId`, будет создана сессия записи без дедупликации. Пример создания такой сессии записи:

```
auto settings = NYdb::NTopic::TWriteSessionSettings()
    .Path(myTopicPath);

auto session = topicClient.CreateWriteSession(settings);
```

Для включения дедупликации нужно в настройках сессии записи указать опцию `ProducerId` или явно включить дедупликацию, вызвав метод `DeduplicationEnabled()`, например, как в секции "Подключение к топику".

ISimpleBlockingWriteSession

Поведение такое же, как у `IWriteSession`: если не указывать `ProducerId`, сессия создаётся без дедупликации.

```
auto settings = NYdb::NTopic::TWriteSessionSettings()
    .Path(myTopicPath);

auto session = topicClient.CreateSimpleBlockingWriteSession(settings);
```

IProducer

`IProducer` всегда записывает с дедупликацией: идентификатор продюсера формируется из `ProducerIdPrefix` и id партиции. Для записи без дедупликации используйте `IWriteSession` или `ISimpleBlockingWriteSession`.

JavaScript

Функциональность на данный момент не поддерживается.

Go

В **ydb-go-sdk** при создании писателя, если не передавать `topicoptions.WithWriterProducerID`, SDK всё равно подставляет идентификатор производителя (генерирует его автоматически). Режим записи без дедупликации, эквивалентный отсутствию `ProducerId` в примере для C++ выше, в текущей версии SDK недоступен.

Java

Функциональность на данный момент не поддерживается.

C#

Функциональность на данный момент не поддерживается.

Rust

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#341

PHP

Функциональность на данный момент не поддерживается.

Запись метаданных на уровне сообщения

При записи сообщения можно дополнительно указать метаданные как список пар "ключ-значение". Эти данные будут доступны при вычитывании сообщения.
Ограничение на размер метаданных — не более 1000 ключей.

C++

IWriteSession

Метаданные задаются в объекте `TWriteMessage` и передаются в `Write()`:

```
auto settings = NYdb::NTopic::TWriteSessionSettings()
    .Path(myTopicPath)
// set all other settings;
;

auto session = topicClient.CreateWriteSession(settings);

std::optional<NYdb::NTopic::TWriteSessionEvent::TEvent> event = session->GetEvent(/*block=*/true);
NYdb::NTopic::TWriteMessage message("This is yet another message").MessageMeta({
    {"meta-key", "meta-value"},
    {"another-key", "value"}
});

if (auto* readyEvent = std::get_if<NYdb::NTopic::TWriteSessionEvent::TReadyToAcceptEvent>(&*event)) {
    session->Write(std::move(event.ContinuationToken), std::move(message));
}
```

ISimpleBlockingWriteSession

Используется тот же `TWriteMessage`, что и у `IWriteSession`: метаданные задаются через `MessageMeta()` и передаются в `Write()`:

```
auto messageData = std::string("message-data");
NYdb::NTopic::TWriteMessage writeMessage(messageData);
writeMessage.MessageMeta({
    {"meta-key", "meta-value"},
    {"another-key", "value"},
});
session->Write(std::move(writeMessage));
```

IProducer

Как и у сессий записи, метаданные задаются в `TWriteMessage` через `MessageMeta()`. Отличие `IProducer` в том, что сообщение также содержит ключ партиционирования, по которому продюсер выбирает партицию:

```
auto messageData = std::string("message-data");
NYdb::NTopic::TWriteMessage writeMessage("user-42", messageData);
writeMessage.MessageMeta({
    {"meta-key", "meta-value"},
    {"another-key", "value"},
});
producer->Write(std::move(writeMessage));
```

Go

Метаданные задаются в поле `Metadata` структуры `topicwriter.Message`:

```
err := writer.Write(ctx, topicwriter.Message{
  Data: strings.NewReader("message-data"),
  Metadata: map[string][]byte{
    "meta-key":    []byte("meta-value"),
    "another-key": []byte("value"),
  },
})
```

При чтении метаданные доступны в поле `Metadata` у сообщения:

```
msg, err := reader.ReadMessage(ctx)
if err != nil {
  return err
}
for k, v := range msg.Metadata {
  fmt.Printf("%s: %s\n", k, string(v))
}
```

Java

При конструировании сообщения для записи с помощью Builder'а, ему можно передать объекты типа `MetadataItem` с парой ключ типа `String` + значение типа `byte[]`.

Можно передать сразу `List` таких объектов:

```
List<MetadataItem> metadataItems = Arrays.asList(
        new MetadataItem("meta-key", "meta-value".getBytes()),
        new MetadataItem("another-key", "value".getBytes())
);
writer.send(
        Message.newBuilder()
                .setMetadataItems(metadataItems)
                .build()
);
```

Или добавлять каждый `MetadataItem` отдельно:

```
writer.send(
        Message.newBuilder()
                .addMetadataItem(new MetadataItem("meta-key", "meta-value".getBytes()))
                .addMetadataItem(new MetadataItem("another-key", "value".getBytes()))
                .build()
);
```

При чтении эти метаданные сообщения получить, вызвав на нём метод `getMetadataItems()`:

```
Message message = reader.receive();
List<MetadataItem> metadata = message.getMetadataItems();
```

Python

Для использования функции передачи метаданных создайте объект `TopicWriterMessage` с аргументом `metadata_items`, как показано ниже:

Native SDK

```
message = ydb.TopicWriterMessage(data=f"message-data", metadata_items={"meta-key": "meta-value"})
writer.write(message)
```

Native SDK (Asyncio)

```
message = ydb.TopicWriterMessage(data="message-data", metadata_items={"meta-key": "meta-value"})
await writer.write(message)
```

Во время чтения метаданные можно получить из поля `metadata_items` объекта `PublicMessage`:

```
message = reader.receive_message()
for meta_key, meta_value in message.metadata_items.items():
    print(f"{meta_key}: {meta_value}")
```

C#

```
await writer.WriteAsync(
    new Ydb.Sdk.Services.Topic.Writer.Message<string>("Hello, Example YDB Topics!")
        { Metadata = { new Metadata("meta-key", "meta-value"u8.ToArray()) } }
);
```

JavaScript

```
writer.write(Buffer.from("Hello, world!", "utf-8"), {
  metadataItems: {
    "meta-key": new TextEncoder().encode("meta-value"),
  },
});
```

Rust

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#341

PHP

Функциональность на данный момент не поддерживается.

Запись в транзакции

C++

IWriteSession

Для записи в топик в транзакции необходимо передать ссылку на объект транзакции в метод `Write` сессии записи.

Пример на GitHub

```
NYdb::NQuery::TQueryClient queryClient(driver);

NYdb::NStatusHelpers::ThrowOnError(queryClient.RetryQuerySync([](NYdb::NQuery::TSession session) -> NYdb::TSt
atus {
    auto beginTxResult = session.BeginTransaction().GetValueSync();
    if (!beginTxResult.IsSuccess()) {
        return beginTxResult;
    }
    auto tx = beginTxResult.GetTransaction();

    NYdb::NTopic::TWriteMessage writeMessage("message");

    topicSession->Write(std::move(writeMessage), tx);
    return tx.Commit().GetValueSync();
}));
```

ISimpleBlockingWriteSession

Как и `IWriteSession`, `ISimpleBlockingWriteSession` поддерживает запись в транзакции. Так как у простого варианта нет `ContinuationToken`, объект транзакции передаётся вторым аргументом в `Write()`.

Пример на GitHub

```
NYdb::NQuery::TQueryClient queryClient(driver);

NYdb::NStatusHelpers::ThrowOnError(queryClient.RetryQuerySync([](NYdb::NQuery::TSession session) -> NYdb::TSt
atus {
    auto beginTxResult = session.BeginTransaction().GetValueSync();
    if (!beginTxResult.IsSuccess()) {
        return beginTxResult;
    }
    auto tx = beginTxResult.GetTransaction();

    NYdb::NTopic::TWriteMessage writeMessage("message");

    topicSession->Write(std::move(writeMessage), &tx);
    return tx.Commit().GetValueSync();
}));
```

IProducer

У `IProducer` транзакция задаётся не аргументом `Write`, а в `TWriteMessage` через `Tx()`. После этого продюсер записывает сообщение так же, как обычное сообщение с ключом партиционирования:

Go

Для записи в топик в транзакции необходимо создать транзакционного писателя через вызов TopicClient.StartTransactionalWriter. После этого можно отправлять сообщения, как обычно. Закрывать транзакционного писателя не требуется — это происходит автоматически при завершении транзакции.

Пример на GitHub

```
err := db.Query().DoTx(ctx, func(ctx context.Context, tx query.TxActor) error {
  writer, err := db.Topic().StartTransactionalWriter(tx, topicName)
```

```
  if err != nil {
    return err
  }

  return writer.Write(ctx, topicwriter.Message{Data: strings.NewReader("asd")})
})
```

Python

Для записи в топик в транзакции необходимо создать транзакционного писателя через вызов `topic_client.tx_writer`.
После этого можно отправлять сообщения, как обычно. Закрывать транзакционного писателя не требуется — это происходит автоматически при завершении транзакции.

В примере ниже нет явного вызова `tx.commit()` — он происходит неявно при успешном завершении лямбды `callee`.

Пример на GitHub

Native SDK

```
with ydb.QuerySessionPool(driver) as session_pool:

    def callee(tx: ydb.QueryTxContext):
        tx_writer: ydb.TopicTxWriter = driver.topic_client.tx_writer(tx, topic)

        for i in range(message_count):
            result_stream = tx.execute(query=f"select {i} as res;")
            for result_set in result_stream:
                message = str(result_set.rows[0]["res"])
                tx_writer.write(ydb.TopicWriterMessage(message))
                print(f"Message {message} was written with tx.")

    session_pool.retry_tx_sync(callee)
```

Native SDK (Asyncio)

Пример на GitHub

```
async with ydb.aio.QuerySessionPool(driver) as session_pool:

    async def callee(tx: ydb.aio.QueryTxContext):
        tx_writer: ydb.TopicTxWriterAsyncIO = driver.topic_client.tx_writer(tx, topic)

        for i in range(message_count):
            async with await tx.execute(query=f"select {i} as res;") as result_stream:
                async for result_set in result_stream:
                    message = str(result_set.rows[0]["res"])
                    await tx_writer.write(ydb.TopicWriterMessage(message))
                    print(f"Message {result_set.rows[0]['res']} was written with tx.")

    await session_pool.retry_tx_async(callee)
```

Java

Синхронный API

Пример на GitHub

В настройках `SendSettings` метода `send` можно указать транзакцию.
Тогда сообщение будет записано вместе с коммитом этой транзакцией.

```
// creating a session in the table service
Result<Session> sessionResult = tableClient.createSession(Duration.ofSeconds(10)).join();
if (!sessionResult.isSuccess()) {
  logger.error("Couldn't get a session from the pool: {}", sessionResult);
  return; // retry or shutdown
}
Session session = sessionResult.getValue();
// creating a transaction in the table service
// this transaction is not yet active and has no id
TableTransaction transaction = session.createNewTransaction(TxMode.SERIALIZABLE_RW);

// get message text within the transaction
Result<DataQueryResult> dataQueryResult = transaction.executeDataQuery("SELECT \"Hello, world!\";")
      .join();
if (!dataQueryResult.isSuccess()) {
  logger.error("Couldn't execute DataQuery: {}", dataQueryResult);
  return; // retry or shutdown
}
// now the transaction is active and has an id

ResultSetReader rsReader = dataQueryResult.getValue().getResultSet(0);
byte[] message;
if (rsReader.next()) {
  message = rsReader.getColumn(0).getBytes();
} else {
  return; // retry or shutdown
}

writer.send(
      Message.of(message),
```

```
      SendSettings.newBuilder()
              .setTransaction(transaction)
              .build()
);

// flush to wait until all messages reach server before commit
writer.flush();

Status commitStatus = transaction.commit().join();
analyzeCommitStatus(commitStatus);
```

Асинхронный API

Пример на GitHub

В настройках `SendSettings` метода `send` можно указать транзакцию.
Тогда сообщение будет записано вместе с коммитом этой транзакцией.

```
// creating a session in the table service
Result<Session> sessionResult = tableClient.createSession(Duration.ofSeconds(10)).join();
if (!sessionResult.isSuccess()) {
  logger.error("Couldn't get a session from the pool: {}", sessionResult);
  return; // retry or shutdown
}
Session session = sessionResult.getValue();
// creating a transaction in the table service
// this transaction is not yet active and has no id
TableTransaction transaction = session.createNewTransaction(TxMode.SERIALIZABLE_RW);

// get message text within the transaction
Result<DataQueryResult> dataQueryResult = transaction.executeDataQuery("SELECT \"Hello, world!\";")
      .join();
if (!dataQueryResult.isSuccess()) {
  logger.error("Couldn't execute DataQuery: {}", dataQueryResult);
  return; // retry or shutdown
}
// now the transaction is active and has an id

ResultSetReader rsReader = dataQueryResult.getValue().getResultSet(0);
byte[] message;
if (rsReader.next()) {
  message = rsReader.getColumn(0).getBytes();
} else {
  return; // retry or shutdown
}

try {
  writer.send(Message.newBuilder()
                          .setData(message)
                          .build(),
                  SendSettings.newBuilder()
                          .setTransaction(transaction)
                          .build())
          .whenComplete((result, ex) -> {
              if (ex != null) {
                  logger.error("Exception while sending a message: ", ex);
              } else {
                  switch (result.getState()) {
                      case WRITTEN:
                          WriteAck.Details details = result.getDetails();
                          logger.info("Message was written successfully, offset: " + details.getOffset());
                          break;
                      case ALREADY_WRITTEN:
                          logger.info("Message has already been written");
                          break;
                      default:
                          break;
                  }
              }
          })
          // Waiting for the message to reach the server before committing the transaction
          .join();

  Status commitStatus = transaction.commit().join();
  analyzeCommitStatus(commitStatus);
} catch (QueueOverflowException exception) {
  logger.error("Queue overflow exception while sending a message{}: ", index, exception);
  // Send queue is full. Need to retry with backoff or skip
}
```

Примечание

Требования к транзакции:

- Это должна быть активная (имеющая идентификатор) транзакция в одном из сервисов YDB. Например, Table или Query.
- В Topic Service поддерживается только уровень изоляции транзакций `SERIALIZABLE_RW`.

C#

Функциональность на данный момент не поддерживается.

JavaScript

Функциональность на данный момент не поддерживается.

Rust

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#341

PHP

Функциональность на данный момент не поддерживается.

Чтение сообщений

Подключение к топику для чтения сообщений

Чтение сообщений из топика может выполнятся с указанием Consumer'а, связанного с этим топиком, а также без Consumer'а. Если Consumer не указан, то клиентское приложение должно самостоятельно рассчитывать offset для чтения сообщений. Более подробно пример чтения без Consumer'а рассмотрен в соответствующей секции.

Создать Consumer можно при создании или изменении топика.
У топика может быть несколько Consumer'ов и для каждого из них сервер хранит свой прогресс чтения.

C++

Подключение для чтения из одного или нескольких топиков представлено объектом сессии чтения с интерфейсом `IReadSession`. Настройки сессии чтения представлены структурой `TReadSessionSettings`.

Полный список настроек смотри в заголовочном файле.

Чтобы создать подключение к существующему топику `my-topic` через добавленного ранее читателя `my-consumer`, используйте следующий код:

```
auto settings = NYdb::NTopic::TReadSessionSettings()
    .ConsumerName("my-consumer")
    .AppendTopics("my-topic");

auto session = topicClient.CreateReadSession(settings);
```

Go

Чтобы создать подключение к существующему топику `my-topic` через добавленного ранее читателя `my-consumer`, используйте следующий код:

```
reader, err := db.Topic().StartReader("my-consumer", topicoptions.ReadTopic("my-topic"))
if err != nil {
    return err
}
```

Python

Чтобы создать подключение к существующему топику `my-topic` через добавленного ранее читателя `my-consumer`, используйте следующий код:

Native SDK

```
reader = driver.topic_client.reader(topic="my-topic", consumer="my-consumer")
```

Native SDK (Asyncio)

```
reader = driver.topic_client.reader(topic="my-topic", consumer="my-consumer")
```

Java

Синхронный API

Инициализация настроек читателя

```
ReaderSettings settings = ReaderSettings.newBuilder()
      .setConsumerName(consumerName)  // имя consumer'а, зарегистрированного на топике
      .addTopic(TopicReadSettings.newBuilder()
              .setPath(topicPath)
              .setReadFrom(Instant.now().minus(Duration.ofHours(24))) // читать с этой временной метки (опцио
нально)
              .setMaxLag(Duration.ofMinutes(30)) // максимальное отставание от конца очереди (опционально)
              .build())
      .build();
```

Создание синхронного читателя

```
SyncReader reader = topicClient.createSyncReader(settings);
```

После создания синхронного читателя необходимо инициализировать. Для этого следует воспользоваться одним их двух методов:

- `init()`: неблокирующий, запускает процесс инициализации в фоне и не ждёт его завершения.

```
reader.init();
```

- `initAndWait()`: блокирующий, запускает процесс инициализации и ждёт его завершения. Если в процессе инициализации возникла ошибка, будет брошено исключение.

```
try {
    reader.initAndWait();
    logger.info("Init finished successfully");
} catch (Exception exception) {
    logger.error("Exception while initializing reader: ", exception);
    return;
}
```

Асинхронный API

Инициализация настроек читателя

```
ReaderSettings settings = ReaderSettings.newBuilder()
      .setConsumerName(consumerName)  // имя consumer'a, зарегистрированного на топике
      .addTopic(TopicReadSettings.newBuilder()
              .setPath(topicPath)
              .setReadFrom(Instant.now().minus(Duration.ofHours(24))) // читать с этой временной метки (опцио
нально)
              .setMaxLag(Duration.ofMinutes(30)) // максимальное отставание от конца очереди (опционально)
              .build())
      .build();
```

Для асинхронного читателя, помимо общих настроек чтения `ReaderSettings`, понадобятся настройки обработчика событий `ReadEventHandlersSettings`, в которых необходимо передать экземпляр наследника `ReadEventHandler`.
Он будет описывать, как должна происходить обработка различных событий, происходящих во время чтения.

```
ReadEventHandlersSettings handlerSettings = ReadEventHandlersSettings.newBuilder()
      .setEventHandler(new Handler())
      .build();
```

Опционально, в `ReadEventHandlersSettings` можно указать executor'a, на котором будет происходить обработка сообщений; по умолчанию используется внутренний поток SDK.

Для реализации обработчика событий можно унаследоваться от `AbstractReadEventHandler` и переопределить метод `onMessages`.
Метод `onMessages` вызывается каждый раз, когда SDK получает очередной пакет сообщений от сервера. В рамках одного вызова приходит один или несколько сообщений, которые можно подтвердить (`commit`) как по отдельности, так и после обработки всего пакета. Пример реализации:

```
private class Handler extends AbstractReadEventHandler {
  @Override
  public void onMessages(DataReceivedEvent event) {
      for (Message message : event.getMessages()) {
          StringBuilder str = new StringBuilder();
          logger.info("Message received. SeqNo={}, offset={}", message.getSeqNo(), message.getOffset());

          process(message);

          message.commit().thenRun(() -> {
              logger.info("Message committed");
          });
      }
  }
}
```

Создание и инициализация асинхронного читателя:

```
AsyncReader reader = topicClient.createAsyncReader(readerSettings, handlerSettings);
// Init in background
reader.init()
      .thenRun(() -> logger.info("Init finished successfully"))
      .exceptionally(ex -> {
          logger.error("Init failed with ex: ", ex);
          return null;
      });
```

C#

```
await using var reader = new ReaderBuilder<string>(connectionString)
{
    ConsumerName = "Consumer_Example",
    SubscribeSettings = { new SubscribeSettings(topicName) }
}.Build();
```

JavaScript

```
await using reader = createTopicReader(driver, {
  topic: topicName,
  consumer: consumerName,
});
```

Rust

```
use ydb::YdbResult;

let mut reader = topic_client
    .create_reader("my-consumer", "/local/my-topic")
    .await?;
```

PHP

Функциональность на данный момент не поддерживается.

Вы также можете использовать расширенный вариант создания подключения, чтобы указать несколько топиков и задать параметры чтения. Следующий код создаст подключение к топикам `my-topic` и `my-specific-topic` через читателя `my-consumer`:

C++

```
auto settings = NYdb::NTopic::TReadSessionSettings()
    .ConsumerName("my-consumer")
    .AppendTopics("my-topic")
    .AppendTopics(
        NYdb::NTopic::TTopicReadSettings("my-specific-topic")
            .ReadFromTimestamp(someTimestamp)
    );

auto session = topicClient.CreateReadSession(settings);
```

Go

```
reader, err := db.Topic().StartReader("my-consumer", []topicoptions.ReadSelector{
    {
        Path: "my-topic",
    },
    {
        Path:       "my-specific-topic",
        ReadFrom:   time.Date(2022, 7, 1, 10, 15, 0, 0, time.UTC),
    },
    },
)
if err != nil {
    return err
}
```

Также в примере выше задаётся время, с которого следует начинать читать сообщения.

Python

Функциональность находится в разработке.

Java

```
ReaderSettings settings = ReaderSettings.newBuilder()
        .setConsumerName(consumerName)
        .addTopic(TopicReadSettings.newBuilder()
                .setPath("my-topic")
                .build())
        .addTopic(TopicReadSettings.newBuilder()
                .setPath("my-specific-topic")
                .setReadFrom(Instant.now().minus(Duration.ofHours(24))) // Optional
                .setMaxLag(Duration.ofMinutes(30)) // Optional
                .build())
        .build();
```

C#

```
await using var reader = new ReaderBuilder<string>(connectionString)
{
    ConsumerName = "Consumer_Example",
    SubscribeSettings =
    {
        new SubscribeSettings(topicName),
        new SubscribeSettings(topicName + "_another") { ReadFrom = DateTime.Now }
    }
}.Build();
```

JavaScript

```
await using reader = createTopicReader(driver, {
  topic: {
    path: topicPath,
    partitionIds: [1n, 2n, 3n],
  },
  consumer: consumerName,
});

await using reader = createTopicReader(driver, {
  topic: {
    path: topicPath,
    maxLag: "1s", // number, import('ms').StringValue, protobuff Duration
  },
  consumer: consumerName,
});

await using reader = createTopicReader(driver, {
  topic: {
    path: topicPath,
    readFrom: new Date(), // number, Date, protobuf Timestamp
  },
  consumer: consumerName,
});

await using reader = createTopicReader(driver, {
  topic: [
    {
      path: topicPath,
      partitionIds: [1n, 2n, 3n],
    },
    {
      path: topicPath2,
      maxLag: "1s",
    },
    {
      path: topicPath3,
      readFrom: new Date(),
    },
    // ...
  ],
  consumer: consumerName,
});
```

Rust

```
use std::time::{Duration, SystemTime};

use ydb::{TopicReaderOptionsBuilder, TopicSelector, TopicSelectors, YdbResult};

let mut reader = topic_client
    .create_reader_with_params(
        TopicReaderOptionsBuilder::default()
            .consumer("my-consumer".into())
            .topic(TopicSelectors(vec![
                TopicSelector::new("/local/my-topic"),
                TopicSelector {
                    path: "/local/my-specific-topic".into(),
                    partition_ids: None,
                    read_from: Some(SystemTime::now() - Duration::from_secs(3600)),
                },
            ]))
            .build()?,
    )
    .await?;
```

PHP

Функциональность на данный момент не поддерживается.

Чтение сообщений

Сервер хранит позицию чтения сообщений. После вычитывания очередного сообщения клиент может отправить на сервер подтверждение обработки. Позиция чтения изменится, а при новом подключении будут вычитаны только неподтвержденные сообщения.

Читать сообщения можно и без подтверждения обработки. В этом случае при новом подключении будут прочитаны все неподтвержденные сообщения, в том числе и уже обработанные.

Информацию о том, какие сообщения уже обработаны, можно сохранять на клиентской стороне, передавая на сервер стартовую позицию чтения при создании подключения. При этом позиция чтения сообщений на сервере не изменяется.

Можно использовать транзакции. В этом случае позиция чтения изменится при подтверждении транзакции. При новом подключении будут прочитаны все неподтверждённые сообщения.

C++

Работа пользователя с объектом `IReadSession` в общем устроена как обработка цикла событий со следующими типами событий: `TDataReceivedEvent`, `TCommitOffsetAcknowledgementEvent`, `TStartPartitionSessionEvent`,

`TEndPartitionSessionEvent`, `TStopPartitionSessionEvent`, `TPartitionSessionStatusEvent`, `TPartitionSessionClosedEvent` и `TSessionClosedEvent`.

Для каждого из типов событий можно установить обработчик этого события, а также можно установить общий обработчик. Обработчики устанавливаются в настройках сессии записи перед её созданием.

Если обработчик для некоторого события не установлен, его необходимо получить и обработать в методах `GetEvent` / `GetEvents`. Для неблокирующего ожидания очередного события есть метод `WaitEvent` с сигнатурой `TFuture<void>()`.

Go

SDK получает данные с сервера партиями и буферизирует их. В зависимости от задач клиентский код может читать сообщения из буфера по одному или пакетами.

Python

SDK получает данные с сервера партиями и буферизирует их. В зависимости от задач клиентский код может читать сообщения из буфера по одному или пакетами.

Java

SDK получает данные с сервера партиями и буферизирует их. В зависимости от задач клиентский код может читать сообщения из буфера по одному или пакетами.

C#

SDK получает данные с сервера партиями и буферизирует их. В зависимости от задач клиентский код может читать сообщения из буфера по одному или пакетами.

JavaScript

Функциональность на данный момент не поддерживается.

Rust

Полный пример чтения топика в транзакции с записью в таблицу: `topic-read-in-transaction-example.rs`.

```
let batch = reader.pop_batch_in_tx(&mut tx).await?;
// обработка batch.messages и commit транзакции
```

PHP

Функциональность на данный момент не поддерживается.

Чтение без подтверждения обработки сообщений

Чтение сообщений по одному

C++

Чтение сообщений по одному в C++ SDK не предусмотрено. Событие `TDataReceivedEvent` содержит пакет сообщений.

Go

```
func SimpleReadMessages(ctx context.Context, r *topicreader.Reader) error {
    for {
        mess, err := r.ReadMessage(ctx)
        if err != nil {
            return err
        }
        processMessage(mess)
    }
}
```

Python

Native SDK

```
while True:
    message = reader.receive_message()
    process(message)
```

Native SDK (Asyncio)

```
while True:
    message = await reader.receive_message()
    process(message)
```

Java

Синхронный API

Чтобы читать сообщения без подтверждения обработки, по одному, используйте следующий код:

```
while(true) {
  Message message = reader.receive();
  process(message);
}
```

Асинхронный API

В асинхронном клиенте нет возможности читать сообщения по одному.

C#

```
try
{
    while (!readerCts.IsCancellationRequested)
    {
        var message = await reader.ReadAsync(readerCts.Token);

        logger.LogInformation("Received message: [{MessageData}]", message.Data);
    }
}
catch (OperationCanceledException)
{
}
```

JavaScript

```
for await (let batch of reader.read()) {
  for await (let msg of batch) {
  }
}
```

Rust

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#330

PHP

Функциональность на данный момент не поддерживается.

Чтение сообщений пакетом

C++

При установке сессии чтения с настройкой `SimpleDataHandlers` достаточно передать обработчик для сообщений с данными. SDK будет вызывать этот обработчик на каждый принятый от сервера пакет сообщений. Подтверждения чтения по умолчанию отправляться не будут.

```
auto settings = NYdb::NTopic::TReadSessionSettings()
    .EventHandlers_.SimpleDataHandlers(
        [](NYdb::NTopic::TReadSessionEvent::TDataReceivedEvent& event) {
            std::cout << "Get data event " << NYdb::NTopic::DebugString(event);
        }
    );

auto session = topicClient.CreateReadSession(settings);

// Wait SessionClosed event.
session->GetEvent(/* block = */true);
```

В этом примере после создания сессии основной поток дожидается завершения сессии со стороны сервера в методе `GetEvent`, другие типы событий приходить не будут.

Go

```
func SimpleReadBatches(ctx context.Context, r *topicreader.Reader) error {
    for {
        batch, err := r.ReadMessageBatch(ctx)
        if err != nil {
            return err
        }
        processBatch(batch)
    }
}
```

Python

Native SDK

```
while True:
    batch = reader.receive_batch()
    process(batch)
```

Native SDK (Asyncio)

```
while True:
    batch = await reader.receive_batch()
    process(batch)
```

Java

Синхронный API

В синхронном клиенте нет возможности прочитать сразу пакет сообщений.

Асинхронный API

Чтобы прочитать пакет сообщений без подтверждения обработки, используйте следующий код:

```
private class Handler extends AbstractReadEventHandler {
  @Override
  public void onMessages(DataReceivedEvent event) {
      for (Message message : event.getMessages()) {
          process(message);
      }
  }
}
```

C#

```
try
{
    while (!readerCts.IsCancellationRequested)
    {
        var batchMessages = await reader.ReadBatchAsync(readerCts.Token);

        foreach (var message in batchMessages.Batch)
        {
            logger.LogInformation("Received message: [{MessageData}]", message.Data);
        }
    }
}
catch (OperationCanceledException)
{
}
```

JavaScript

```
for await (let batch of reader.read()) {
}
```

Rust

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#330

PHP

Функциональность на данный момент не поддерживается.

Чтение с подтверждением обработки сообщений

Подтверждение обработки сообщения (коммит) - сообщает серверу, что сообщение из топика обработано получателем и больше его отправлять не нужно. При использовании чтения с подтверждением нужно подтверждать все полученные сообщения без пропуска. Коммит сообщений на сервере происходит после подтверждения очередного интервала сообщений «без дырок», сами подтверждения при этом можно отправлять в любом порядке.

Например, с сервера пришли сообщения 1, 2, 3. Программа обрабатывает их параллельно и отправляет подтверждения в таком порядке: 1, 3, 2. В этом случае сначала будет закоммичено сообщение 1, а сообщения 2 и 3 будут закоммичены только после того как сервер получит подтверждение об обработке сообщения 2.

В случае ошибки на коммите сообщения можно написать эту ошибку в лог и продолжить работу. Состояние сообщения в этой точке неизвестно. Сообщение могло закоммититься, а потом возникла сетевая ошибка и клиент не получил подтверждения. Если сообщение не закоммитилось, то оно будет прочитано ещё раз и снова поступит в обработку (может быть на другом читателе). Ретраить именно коммит смысла нет, т.к. сессия чтения этого сообщения уже потеряна.

Чтение сообщений по одному с подтверждением

C++

Чтение сообщений по одному в C++ SDK не предусмотрено. Событие `TDataReceivedEvent` содержит пакет сообщений.

Go

```
func SimpleReadMessages(ctx context.Context, r *topicreader.Reader) error {
    for {
      mess, err := r.ReadMessage(ctx)
      if err != nil {
          return err
      }
      processMessage(mess)
      r.Commit(mess.Context(), mess)
    }
}
```

По умолчанию `Commit` — это быстрый вызов: сохраняет данные во внутреннем буфере и сразу возвращает управление, а реальная отправка происходит позже. Поэтому, чтобы не терять последние коммиты перед выходом из программы, читателя

нужно закрывать явно с помощью вызова `Reader.Close()`.

Python

`commit` - это быстрый вызов: сохраняет данные во внутреннем буфере и сразу возвращает управление, а реальная отправка происходит позже. Поэтому, чтобы не терять последние коммиты перед выходом из программы, читателя нужно закрывать явно.

Native SDK

```
while True:
    message = reader.receive_message()
    process(message)
    reader.commit(message)
```

Native SDK (Asyncio)

```
while True:
    message = await reader.receive_message()
    process(message)
    reader.commit(message)
```

Java

Для подтверждения обработки сообщения достаточно вызвать у сообщения метод `commit`.
Актуально как для синхронного, так и для асинхронного читателя.
В асинхронном читателе, при обработке пакета сообщений, можно вызвать `commit` или у всего пакета сразу, или у каждого сообщения отдельно.
Этот метод возвращает `CompletableFuture<Void>`, успешное выполнение которой означает подтверждение обработки сервером.
В случае ошибки коммита не следует пытаться его ретраить. Скорее всего, ошибка вызвана закрытием сессии.
Читатель (необязательно этот же) сам создаст новую сессию для этой партиции и сообщение будет прочитано снова.

```
message.commit()
       .whenComplete((result, ex) -> {
           if (ex != null) {
               // Read session was probably closed, there is nothing we can do here.
               // Do not retry this commit on the same event.
               logger.error("exception while committing message: ", ex);
           } else {
               logger.info("message committed successfully");
           }
       });
```

C#

```
try
{
    while (!readerCts.IsCancellationRequested)
    {
        var message = await reader.ReadAsync(readerCts.Token);

        logger.LogInformation("Received message: [{MessageData}]", message.Data);

        try
        {
            await message.CommitAsync();
        }
        catch (ReaderException e)
        {
            logger.LogError(e, "Failed to commit a message");
        }
    }
}
catch (OperationCanceledException)
{
}
```

JavaScript

```
for await (let batch of reader.read()) {
  for (let msg of batch) {
    await reader.commit(msg);
  }
}
```

Rust

```
let batch = reader.read_batch().await?;
reader.commit(batch.get_commit_marker())?;
// или с ожиданием ack от сервера:
reader.commit_with_ack(batch.get_commit_marker()).await?;
```

PHP

Функциональность на данный момент не поддерживается.

Чтение сообщений пакетом с подтверждением

C++

Аналогично примеру выше, при установке сессии чтения с настройкой `SimpleDataHandlers` достаточно передать обработчик для сообщений с данными. SDK будет вызывать этот обработчик на каждый принятый от сервера пакет сообщений. Передача параметра `commitDataAfterProcessing = true` означает, что SDK будет отправлять на сервер подтверждения чтения всех сообщений после выполнения обработчика.

```
auto settings = NYdb::NTopic::TReadSessionSettings()
    .EventHandlers_.SimpleDataHandlers(
        [](NYdb::NTopic::TReadSessionEvent::TDataReceivedEvent& event) {
            std::cout << "Get data event " << NYdb::NTopic::DebugString(event);
        }
        , /* commitDataAfterProcessing = */true
    );

auto session = topicClient.CreateReadSession(settings);

// Wait SessionClosed event.
session->GetEvent(/* block = */true);
```

Go

```
func SimpleReadMessageBatch(ctx context.Context, r *topicreader.Reader) error {
    for {
      batch, err := r.ReadMessageBatch(ctx)
      if err != nil {
          return err
      }
      processBatch(batch)
      r.Commit(batch.Context(), batch)
    }
}
```

По умолчанию `Commit` - это быстрый вызов: сохраняет данные во внутреннем буфере и сразу возвращает управление, а реальная отправка происходит позже. Поэтому, чтобы не терять последние коммиты перед выходом из программы, читателя нужно закрывать явно.

Python

Native SDK

```
while True:
    batch = reader.receive_batch()
    process(batch)
    reader.commit(batch)
```

Native SDK (Asyncio)

```
while True:
    batch = await reader.receive_batch()
    process(batch)
    reader.commit(batch)
```

`commit` - это быстрый вызов: сохраняет данные во внутреннем буфере и сразу возвращает управление, а реальная отправка происходит позже. Поэтому, чтобы не терять последние коммиты перед выходом из программы, читателя нужно закрывать явно.

Java

Синхронный API

Неактуально, т.к. в синхронном читателе нет возможности читать сообщения пакетами.

Асинхронный API

В обработчике `onMessages` можно закоммитить весь пакет сообщений, вызвав `commit` на событии.

```
@Override
public void onMessages(DataReceivedEvent event) {
  for (Message message : event.getMessages()) {
      process(message);
  }
  event.commit()
         .whenComplete((result, ex) -> {
             if (ex != null) {
                 // Read session was probably closed, there is nothing we can do here.
                 // Do not retry this commit on the same message.
                 logger.error("exception while committing message batch: ", ex);
             } else {
                 logger.info("message batch committed successfully");
             }
```

```
        });
}
```

C#

```
try
{
    while (!readerCts.IsCancellationRequested)
    {
        var batchMessages = await reader.ReadBatchAsync(readerCts.Token);

        foreach (var message in batchMessages.Batch)
        {
            logger.LogInformation("Received message: [{MessageData}]", message.Data);
        }

        try
        {
            await batchMessages.CommitBatchAsync();
        }
        catch (ReaderException e)
        {
            logger.LogError(e, "Failed to commit a message");
        }
    }
}
catch (OperationCanceledException)
{
}
```

JavaScript

```
for await (let batch of reader.read()) {
  await reader.commit(batch);
}
```

Rust

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#330

PHP

Функциональность на данный момент не поддерживается.

Чтение с хранением позиции на клиентской стороне

Вместо коммитов сообщений на сервер можно хранить прогресс чтения самостоятельно. В этом случае нужно передать в SDK обработчик, который будет вызываться при старте чтения каждой партиции. В этом обработчике нужно будет указать позицию, с которой нужно начинать чтение этой партиции.

C++

При обработке событий `TStartPartitionSessionEvent` можно при ответе серверу задать позицию, с которой следует начинать чтение.
Для этого в метод `Confirm` следует передать параметр `readOffset`.
Дополнительно можно передать параметр `commitOffset`, который укажет позицию, сообщения до которой следует считать закоммиченными.

Пример установки обработчика:

```
settings.EventHandlers_.StartPartitionSessionHandler(
    [](NYdb::NTopic::TReadSessionEvent::TStartPartitionSessionEvent& event) {
        auto readFromOffset = GetOffsetToReadFrom(event.GetPartitionId());
        event.Confirm(readFromOffset);
    }
);
```

Здесь `GetOffsetToReadFrom` - это часть примера, а не SDK. Используйте свой способ определить требуемую стартовую позицию чтения для партиции с данным partition id.

Также в `TReadSessionSettings` поддерживается настройка `ReadFromTimestamp` для чтения событий с отметками времени записи не меньше данной. Эта настройка предполагается не для точного позиционирования старта, а для пропуска объёма данных за большой интервал времени. Несколько первых полученных сообщений могут иметь отметки времени записи меньше указанной.

Go

> **Совет**
>
> В режиме читателя по умолчанию оффсеты до позиции, указанной через `res.StartFrom`, подтверждаются на сервере. После этого повторное чтение тех же сообщений путём сдвига позиции назад становится невозможным. Чтобы отключить автоматическое подтверждение, используйте режим без коммитов при создании читателя.
>
> ```
> reader, err := db.Topic().StartReader(
>   consumerName,
>   topicoptions.ReadTopic(topicName),
>   topicoptions.WithReaderCommitMode(topicoptions.CommitModeNone),
> )
> ```

```
func ReadWithExplicitPartitionStartStopHandlerAndOwnReadProgressStorage(ctx context.Context, db ydb.Connectio
n) error {
    readContext, stopReader := context.WithCancel(context.Background())
    defer stopReader()

    readStartPosition := func(
        ctx context.Context,
        req topicoptions.GetPartitionStartOffsetRequest,
    ) (res topicoptions.GetPartitionStartOffsetResponse, err error) {
        offset, err := readLastOffsetFromDB(ctx, req.Topic, req.PartitionID)
        res.StartFrom(offset)

        // Reader will stop if return err != nil
        return res, err
    }

    r, err := db.Topic().StartReader("my-consumer", topicoptions.ReadTopic("my-topic"),
        topicoptions.WithGetPartitionStartOffset(readStartPosition),
    )
    if err != nil {
        return err
    }

    go func() {
        <-readContext.Done()
        _ = r.Close(ctx)
    }()

    for {
        batch, err := r.ReadMessageBatch(readContext)
        if err != nil {
            return err
        }

        processBatch(batch)
        _ = externalSystemCommit(batch.Context(), batch.Topic(), batch.PartitionID(), batch.EndOffset())
    }
}
```

Python

Функциональность находится в разработке.

Java

Чтение с заданного оффсета в Java возможно только в асинхронном читателе.
В обработчике событий `StartPartitionSessionEvent` можно при ответе серверу задать позицию, с которой следует начинать чтение.
Для этого в метод `confirm` следует передать настройки `StartPartitionSessionSettings` с указанным оффсетом через `setReadOffset`.
Также вызовом `setCommitOffset` можно указать оффсет, который следует считать закоммиченным.

```
@Override
public void onStartPartitionSession(StartPartitionSessionEvent event) {
    event.confirm(StartPartitionSessionSettings.newBuilder()
            .setReadOffset(lastReadOffset) // Long
            .setCommitOffset(lastCommitOffset) // Long
            .build());
}
```

Также поддерживается настройка читателя `setReadFrom` для чтения событий с отметками времени записи не меньше данной.

JavaScript

```
await using reader = createTopicReader(driver, {
  topic: topicName,
  consumer: consumerName,
  onPartitionSessionStart: (evt) => {
    return {
      readOffset: 0n,
```

```
      commitOffset: 0n,
    };
  },
});
```

C#

Функциональность на данный момент не поддерживается.

Rust

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#330

PHP

Функциональность на данный момент не поддерживается.

Чтение без указания Consumer'а

Обычно прогресс чтения топика сохраняется на сервере в каждом `Consumer`е. Но можно не хранить такой прогресс на сервере и при создании читателя явно указать, что чтение будет происходить без `Consumer`а.

C++

В `NYdb::NTopic::TReadSessionSettings` вызовите `WithoutConsumer()`:

```
auto settings = NYdb::NTopic::TReadSessionSettings()
    .WithoutConsumer()
    .AppendTopics(
        NYdb::NTopic::TTopicReadSettings("topic-path")
            .AppendPartitionIds(0)
            .AppendPartitionIds(1)
            .AppendPartitionIds(2));

auto readSession = topicClient.CreateReadSession(settings);
```

При переподключении прогресс чтения на сервере не сохраняется. Чтобы не начинать с начала, при каждом старте сессии чтения партиции передавайте смещение в `TStartPartitionSessionEvent::Confirm` — см. хранение позиции на клиенте.

Go

Нужно передать пустую строку в качестве имени consumer и опцию `topicoptions.WithReaderWithoutConsumer(false)` (режим **экспериментальный**, см. VERSIONING в репозитории SDK). В селекторе чтения укажите путь топика и список партиций. Коммиты сообщений в этом режиме недоступны (`CommitModeNone`); при переподключениях прогресс нужно восстанавливать на стороне клиента — см. хранение позиции на клиенте.

```
reader, err := db.Topic().StartReader(
  "",
  topicoptions.ReadSelectors{{
    Path:       "topic-path",
    Partitions: []int64{0, 1, 2},
  }},
  topicoptions.WithReaderWithoutConsumer(false),
)
if err != nil {
  return err
}
```

Java

Для чтения без Consumer'а следует в настройках читателя `ReaderSettings` это явно указать, вызвав `withoutConsumer()`:

```
ReaderSettings settings = ReaderSettings.newBuilder()
        .withoutConsumer()
        .addTopic(TopicReadSettings.newBuilder()
                .setPath(TOPIC_NAME)
                .build())
        .build();
```

В таком случае нужно учитывать, что при переустановке соединения прогресс на сервере будет сброшен. Поэтому, чтобы не начинать чтение сначала, в SDK следует передавать offset начала чтения при каждом старте сессии чтения партиции:

```
@Override
public void onStartPartitionSession(StartPartitionSessionEvent event) {
    event.confirm(StartPartitionSessionSettings.newBuilder()
            .setReadOffset(lastReadOffset) // the last offset read by this client, Long
            .build());
}
```

Python

Для чтения без Consumer'а следует создать читателя с помощью метода `reader` с указанием следующих аргументов:

- `topic` - объект `ydb.TopicReaderSelector` с указанными `path` и списком `partitions`;
- `consumer` - должен быть `None`;
- `event_handler` - наследник `ydb.TopicReaderEvents.EventHandler`, который реализует функцию `on_partition_get_start_offset`. Эта функция отвечает за возвращение начального смещения (offset) для чтения

сообщений при старте читателя, а также во время переподключений. Клиентское приложение должно указать это смещение в параметре `ydb.TopicReaderEvents.OnPartitionGetStartOffsetResponse.start_offset`. Также функция может быть реализована как асинхронная.

Пример:

```
class CustomEventHandler(ydb.TopicReaderEvents.EventHandler):
    def on_partition_get_start_offset(self, event: ydb.TopicReaderEvents.OnPartitionGetStartOffsetRequest):
        return ydb.TopicReaderEvents.OnPartitionGetStartOffsetResponse(
            start_offset=0,
        )

reader = driver.topic_client.reader(
    topic=ydb.TopicReaderSelector(
        path="topic-path",
        partitions=[0, 1, 2],
    ),
    consumer=None,
    event_handler=CustomEventHandler(),
)
```

C#

Функциональность на данный момент не поддерживается.

JavaScript

Функциональность на данный момент не поддерживается.

Rust

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#330

PHP

Функциональность на данный момент не поддерживается.

Чтение в транзакции

C++

Перед чтением из топика клиентский код должен передать в настройки получения событий из сессии ссылку на объект транзакции.

Пример на GitHub

```
readSession->WaitEvent().Wait(TDuration::Seconds(1));

NYdb::NStatusHelpers::ThrowOnError(queryClient.RetryQuerySync([&readSession](NYdb::NQuery::TSession session)
-> NYdb::TStatus {
    auto beginTxResult = session.BeginTransaction(NYdb::Query::TTxSettings::SerializableRW()).GetValueSync();
    if (!beginTxResult.IsSuccess()) {
        return beginTxResult;
    }
    auto tx = beginTxResult.GetTransaction();

    auto topicSettings = NYdb::NTopic::TReadSessionGetEventSettings()
        .Block(false);
        .Tx(tx);

    auto events = readSession->GetEvents(topicSettings);

    for (auto& event : events) {
        // обработать событие и записать результаты в таблицу
    }

    return tx.Commit().GetValueSync();
}));
```

> **Важно**
>
> При обработке событий `events` не нужно явно подтверждать обработку для событий типа `TDataReceivedEvent`.

Подтверждение обработки события `TStopPartitionSessionEvent` надо делать после вызова `Commit`.

```
std::optional<NYdb::NTopic::TStopPartitionSessionEvent> stopPartitionSession;

auto events = readSession->GetEvents(topicSettings);

for (auto& event : events) {
    if (auto* e = std::get_if<NYdb::NTopic::TStopPartitionSessionEvent>(&event)) {
        stopPartitionSessionEvent = std::move(*e);
    } else {
        // обработать событие и записать результаты в таблицу
    }
}
```

```
auto commitResult = tx.Commit(commitSettings).GetValueSync();
if (!commitResult.IsSuccess()) {
    return commitResult;
}

if (stopPartitionSessionEvent) {
    stopPartitionSessionEvent->Commit();
}
```

Go

Для чтения сообщений в рамках транзакции следует использовать метод `Reader.PopMessagesBatchTx`. Он прочитает пакет сообщений и добавит их коммит в транзакцию, при этом отдельно коммитить эти сообщения не требуется. Читателя сообщений можно использовать повторно в разных транзакциях. При этом важно, чтобы порядок коммита транзакций соответствовал порядку получения сообщений от читателя, так как коммиты сообщений в топике должны выполняться строго по порядку. Проще всего это сделать если использовать читателя в цикле.

Пример на GitHub

```
for {
  err := db.Query().DoTx(ctx, func(ctx context.Context, tx query.TxActor) error {
    batch, err := reader.PopMessagesBatchTx(ctx, tx) // батч закоммитится при общем коммите транзакции
    if err != nil {
      return err
    }

    return processBatch(ctx, batch)
  })
  if err != nil {
    handleError(err)
  }
}
```

Python

Для чтения сообщений в рамках транзакции следует использовать метод `reader.receive_batch_with_tx`. Он прочитает пакет сообщений и добавит их коммит в транзакцию, при этом отдельно коммитить эти сообщения не требуется. Читателя сообщений можно использовать повторно в разных транзакциях. При этом важно, чтобы порядок коммита транзакций соответствовал порядку получения сообщений от читателя, так как коммиты сообщений в топике должны выполняться строго по порядку - в противном случае транзакция получит ошибку на попытке сделать коммит. Проще всего это сделать, если использовать читателя в цикле.

Native SDK

Пример на GitHub

```
with driver.topic_client.reader(topic, consumer) as reader:
    with ydb.QuerySessionPool(driver) as session_pool:
        for _ in range(message_count):

            def callee(tx: ydb.QueryTxContext):
                batch = reader.receive_batch_with_tx(tx, max_messages=1)
                print(f"Message {batch.messages[0].data.decode()} was read with tx.")

            session_pool.retry_tx_sync(callee)
```

Native SDK (Asyncio)

Пример на GitHub

```
async with driver.topic_client.reader(topic, consumer) as reader:
    async with ydb.aio.QuerySessionPool(driver) as session_pool:
        for _ in range(message_count):

            async def callee(tx: ydb.aio.QueryTxContext):
                batch = await reader.receive_batch_with_tx(tx, max_messages=1)
                print(f"Message {batch.messages[0].data.decode()} was read with tx.")

            await session_pool.retry_tx_async(callee)
```

Java

Синхронный API

Пример на GitHub

В настройках `ReceiveSettings` метода `receive` можно указать транзакцию:

```
Message message = reader.receive(ReceiveSettings.newBuilder()
      .setTransaction(transaction)
      .build());
```

Тогда полученное сообщение будет закоммичено вместе с транзакцией. Коммитить его отдельно не нужно.
Метод `receive` свяжет на сервере оффсеты сообщения с транзакцией вызовом `sendUpdateOffsetsInTransaction` и вернёт управление, когда получит ответ на него.

Асинхронный API

Пример на GitHub

После получения сообщения в обработчике `onMessages` можно связать одно или несколько сообщений с транзакцией. Для этого нужно вызвать отдельный метод `reader.updateOffsetsInTransaction` и дождаться его выполнения на сервере. Этот метод принимает параметром список оффсетов. Для удобства у `Message` и `DataReceivedEvent` есть метод `getPartitionOffsets()`, возвращающий такой список.

```
@Override
public void onMessages(DataReceivedEvent event) {
  for (Message message : event.getMessages()) {
      // creating a session in the table service
      Result<Session> sessionResult = tableClient.createSession(Duration.ofSeconds(10)).join();
      if (!sessionResult.isSuccess()) {
          logger.error("Couldn't get a session from the pool: {}", sessionResult);
          return; // retry or shutdown
      }
      Session session = sessionResult.getValue();
      // creating a transaction in the table service
      // this transaction is not yet active and has no id
      TableTransaction transaction = session.createNewTransaction(TxMode.SERIALIZABLE_RW);

      // do something else in the transaction
      transaction.executeDataQuery("SELECT 1").join();
      // now the transaction is active and has an id
      // analyzeQueryResultIfNeeded();

      Status updateStatus = reader.updateOffsetsInTransaction(transaction,
                      message.getPartitionOffsets(), new UpdateOffsetsInTransactionSettings.Builder().build
())
              // Do not commit a transaction without waiting for updateOffsetsInTransaction result to avoid a
race condition
              .join();
      if (!updateStatus.isSuccess()) {
          logger.error("Couldn't update offsets in a transaction: {}", updateStatus);
          return; // retry or shutdown
      }

      Status commitStatus = transaction.commit().join();
      analyzeCommitStatus(commitStatus);
  }
}
```

Примечание

Требования к транзакции:

- Это должна быть активная (имеющая идентификатор) транзакция в одном из сервисов YDB. Например, Table или Query.
- В Topic Service поддерживается только уровень изоляции транзакций `SERIALIZABLE_RW`.

C#

Функциональность на данный момент не поддерживается.

Rust

Полный пример чтения топика в транзакции с записью в таблицу: `topic-read-in-transaction-example.rs`.

```
let batch = reader.pop_batch_in_tx(&mut tx).await?;
// обработка batch.messages и commit транзакции
```

PHP

Функциональность на данный момент не поддерживается.

JavaScript

Функциональность на данный момент не поддерживается.

Обработка серверного прерывания чтения

В YDB используется серверная балансировка партиций между клиентами. Это означает, что сервер может прерывать чтение сообщений из произвольных партиций.

При *мягком прерывании* клиент получает уведомление, что сервер уже закончил отправку сообщений из партиции и больше сообщения читаться не будут. Клиент может завершить обработку сообщений и отправить подтверждение на сервер.

В случае *жесткого прерывания* клиент получает уведомление, что работать с сообщениями партиции больше нельзя. Клиент должен прекратить обработку прочитанных сообщений. Неподтвержденные сообщения будут переданы другому читателю.

Мягкое прерывание чтения

C++

Мягкое прерывание приходит в виде события `TStopPartitionSessionEvent` с методом `Confirm`. Клиент может завершить обработку сообщений и отправить подтверждение на сервер.

Фрагмент цикла событий может выглядеть так:

```
auto event = readSession->GetEvent(/*block=*/true);
if (auto* stopPartitionSessionEvent = std::get_if<NYdb::NTopic::TReadSessionEvent::TStopPartitionSessionEvent
>(&*event)) {
    stopPartitionSessionEvent->Confirm();
} else {
  // other event types
}
```

Go

Клиентский код сразу получает все имеющиеся в буфере (на стороне SDK) сообщения, даже если их не достаточно для формирования пакета при групповой обработке.

```
r, _ := db.Topic().StartReader("my-consumer", nil,
    topicoptions.WithBatchReadMinCount(1000),
)

for {
    batch, _ := r.ReadMessageBatch(ctx) // <- if partition soft stop batch can be less, then 1000
    processBatch(batch)
    _ = r.Commit(batch.Context(), batch)
}
```

Python

Специальной обработки не требуется.

Native SDK

```
while True:
    batch = reader.receive_batch()
    process(batch)
    reader.commit(batch)
```

Native SDK (Asyncio)

```
while True:
    batch = await reader.receive_batch()
    process(batch)
    reader.commit(batch)
```

Java

Синхронный API

Неактуально, т.к. в синхронном читателе нет возможности настраивать обработку подобных событий.
Клиент сразу ответит серверу подтверждением остановки.

Асинхронный API

Для возможности реагировать на такое событие следует переопределить метод `onStopPartitionSession(StopPartitionSessionEvent event)` в объекте-наследнике `ReadEventHandler` (см Подключение к топику для чтения сообщений).
`event.confirm()` обязательно должен быть вызван, т.к. сервер ожидает этого ответа для продолжения остановки.

```
@Override
public void onStopPartitionSession(StopPartitionSessionEvent event) {
  logger.info("Partition session {} stopped. Committed offset: {}", event.getPartitionSessionId(),
          event.getCommittedOffset());
  // This event means that no more messages will be received by server
  // Received messages still can be read from ReaderBuffer
  // Messages still can be committed, until confirm() method is called

  // Confirm that session can be closed
  event.confirm();
}
```

C#

Специальной обработки не требуется.

JavaScript

Функциональность на данный момент не поддерживается.

Rust

Rust SDK обрабатывает события остановки и закрытия partition session внутренне; публичного API для настройки мягкого или жёсткого прерывания чтения пока нет.

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#330

PHP

Функциональность на данный момент не поддерживается.

Жесткое прерывание чтения

C++

Жёсткое прерывание приходит в виде события `TPartitionSessionClosedEvent` либо в ответ на подтверждение мягкого прерывания, либо при потере соединения с партицией. Узнать причину можно, вызвав метод `GetReason`.

Фрагмент цикла событий может выглядеть так:

```
auto event = readSession->GetEvent(/*block=*/true);
if (auto* partitionSessionClosedEvent = std::get_if<NYdb::NTopic::TReadSessionEvent::TPartitionSessionClosedE
vent>(&*event)) {
    if (partitionSessionClosedEvent->GetReason() == NYdb::NTopic::TPartitionSessionClosedEvent::EReason::Conn
ectionLost) {
        std::cout << "Connection with partition was lost" << std::endl;
    }
} else {
  // other event types
}
```

Go

При прерывании чтения контекст сообщения или пакета сообщений будет отменен.

```
ctx := batch.Context() // batch.Context() will cancel if partition revoke by server or connection broke
if len(batch.Messages) == 0 {
    return
}

buf := &bytes.Buffer{}
for _, mess := range batch.Messages {
    buf.Reset()
    _, _ = buf.ReadFrom(mess)
    _, _ = io.Copy(buf, mess)
    writeMessagesToDB(ctx, buf.Bytes())
}
```

Python

В этом примере обработка сообщений в батче остановится, если в процессе работы партиция будет отобрана. Такая оптимизация требует дополнительного кода на клиенте. В простых случаях, когда обработка отобранных партиций не является проблемой, ее можно не применять.

Native SDK

```
def process_batch(batch):
    for message in batch.messages:
        if not batch.alive:
            return False
        process(message)
    return True

batch = reader.receive_batch()
if process_batch(batch):
    reader.commit(batch)
```

Native SDK (Asyncio)

```
def process_batch(batch):
    for message in batch.messages:
        if not batch.alive:
            return False
        process(message)
    return True

batch = await reader.receive_batch()
if process_batch(batch):
    reader.commit(batch)
```

Java

Синхронный API

Неактуально, т.к. в синхронном читателе нет возможности настраивать обработку подобных событий.

Асинхронный API

```
@Override
public void onPartitionSessionClosed(PartitionSessionClosedEvent event) {
  logger.info("Partition session {} is closed.", event.getPartitionSession().getPartitionId());
}
```

C#

Функциональность на данный момент не поддерживается.

JavaScript

Функциональность на данный момент не поддерживается.

Rust

Rust SDK обрабатывает события остановки и закрытия partition session внутренне; публичного API для настройки мягкого или жёсткого прерывания чтения пока нет.

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#330

PHP

Функциональность на данный момент не поддерживается.

Поддержка автомасштабирования топиков

> **Важно**
>
> Параметры `min_active_partitions` и `max_active_partitions` сами по себе не включают автомасштабирование топика. Если не передать `auto_partitioning_settings`, останется стратегия `DISABLED`, а число активных партиций будет фиксированным. В YDB 26.2 метод `describe_topic()` в этом случае возвращает эффективное значение `max_active_partitions`, равное `min_active_partitions`.
>
> Чтобы настроить разные минимальную и максимальную границы, дополнительно задайте стратегию `TopicAutoPartitioningStrategy.SCALE_UP`, как в примере ниже.

C++

SDK поддерживает два режима чтения топиков с включенным автомасштабированием: режим полной поддержки и режим совместимости. Режим чтения задаётся в параметрах создания сессии чтения. По умолчанию используется режим совместимости.

```
auto settings = NYdb::NTopic::TReadSessionSettings()
    .SetAutoscalingSupport(true); // full support is enabled

// or

auto settings = NYdb::NTopic::TReadSessionSettings()
    .SetAutoscalingSupport(false); // compatibility mode is enabled

auto readSession = topicClient.CreateReadSession(settings);
```

В режиме полной поддержки, когда все сообщения из партиции будут прочитаны, придёт событие `TEndPartitionSessionEvent`. После получения этого события в партиции больше не появится новых сообщений для чтения. Чтобы продолжить чтение из дочерних партиций, необходимо вызвать `Confirm()`, тем самым подтвердив, что приложение готово принимать сообщения из дочерних партиций. Если сообщения из всех партиций обрабатываются в одном потоке, то `Confirm()` можно вызвать сразу после получения `TEndPartitionSessionEvent`. Если обработка сообщений из разных партиций осуществляется в разных потоках, то следует завершить обработку сообщений, например, выполнить накопившийся батч, подтвердить их обработку (коммит) или сохранить позицию чтения в своей базе, и только после этого вызвать `Confirm()`.

После получения `TEndPartitionSessionEvent` и обработки всех сообщений рекомендуется всегда сразу подтверждать их обработку (коммит). Это позволит сбалансировать чтение дочерних партиций между разными сессиями чтения, что приведёт к равномерному распределению нагрузки по всем читателям.

Фрагмент цикла событий может выглядеть так:

```
auto settings = NYdb::NTopic::TReadSessionSettings()
    .SetAutoscalingSupport(true);

auto readSession = topicClient.CreateReadSession(settings);

auto event = readSession->GetEvent(/*block=*/true);
if (auto* endPartitionSessionEvent = std::get_if<NYdb::NTopic::TReadSessionEvent::TEndPartitionSessionEvent>
(&*event)) {
    endPartitionSessionEvent->Confirm();
} else {
  // other event types
}
```

В режиме совместимости отсутствует явный сигнал о завершении чтения из партиции, и сервер будет пытаться эвристически определить, что клиент обработал партицию до конца. Это может привести к задержке между завершением чтения из исходной партиции и началом чтения из её дочерних партиций.

Если клиент подтверждает обработку сообщений (коммит), то сигналом завершения обработки сообщений из партиции будет подтверждение обработки последнего сообщения этой партиции. В случае, если клиент не подтверждает обработку сообщений, сервер будет периодически прерывать чтение из партиции и переключаться на чтение в другой сессии (если существуют другие сессии, готовые обрабатывать партицию). Это будет продолжаться до тех пор, пока чтение не начнётся с конца партиции.

Рекомендуется проверять корректность обработки мягкого прерывания чтения: клиент должен обработать полученные сообщения, подтвердить их обработку (коммит) или сохранить позицию чтения в своей базе, и только после этого вызывать `Confirm()` для события `TStopPartitionSessionEvent`.

Go

Включение автомасштабирования топика во время его создания производится с помощью опции `topicoptions.CreateWithAutoPartitioningSettings`:

```
import (
  ...

  "github.com/ydb-platform/ydb-go-sdk/v3/topic/topicoptions"
  "github.com/ydb-platform/ydb-go-sdk/v3/topic/topictypes"
)

err := db.Topic().Create(ctx,
  "topic",
  topicoptions.CreateWithAutoPartitioningSettings(
    topictypes.AutoPartitioningSettings{
      AutoPartitioningStrategy: topictypes.AutoPartitioningStrategyScaleUp,
    },
  ),
)
```

При необходимости в AutoPartitioningSettings можно задать и другие параметры:

```
err := db.Topic().Create(ctx,
  "topic",
  topicoptions.CreateWithAutoPartitioningSettings(
    topictypes.AutoPartitioningSettings{
      AutoPartitioningStrategy: topictypes.AutoPartitioningStrategyScaleUp,
      AutoPartitioningWriteSpeedStrategy: topictypes.AutoPartitioningWriteSpeedStrategy{
        StabilizationWindow:    time.Minute,
        UpUtilizationPercent:   80,
      },
    },
  ),
)
```

Включение автомасштабирования у существующего топика производится с помощью опции `topicoptions.AlterWithAutoPartitioningStrategy` у `.Topic().Alter`:

```
import (
  ...

  "github.com/ydb-platform/ydb-go-sdk/v3/topic/topicoptions"
  "github.com/ydb-platform/ydb-go-sdk/v3/topic/topictypes"
)

err := db.Topic().Alter(
  ctx,
  "topic",
  topicoptions.AlterWithAutoPartitioningStrategy(
    topictypes.AutoPartitioningStrategyScaleUp,
  ),
)

// другие опции
err := db.Topic().Alter(
  ctx,
  "topic",
  topicoptions.AlterWithAutoPartitioningStrategy(
    topictypes.AutoPartitioningStrategyScaleUp,
  ),
  topicoptions.AlterWithAutoPartitioningWriteSpeedStabilizationWindow(time.Minute),
  topicoptions.AlterWithAutoPartitioningWriteSpeedUpUtilizationPercent(80),
)
```

SDK поддерживает два режима чтения топиков с включенным автомасштабированием: режим полной поддержки и режим совместимости. Режим чтения задаётся опцией `topicoptions.WithReaderSupportSplitMergePartitions` во время создания читателя. По умолчанию используется режим полной поддержки (`true`).

```
import (
  ...

  "github.com/ydb-platform/ydb-go-sdk/v3/topic/topicoptions"
  "github.com/ydb-platform/ydb-go-sdk/v3/topic/topictypes"
)

// режим полной поддержки (обработка автомасштабирования в SDK, по умолчанию)
reader, err := db.Topic().StartReader(
  "consumer",
  topicoptions.ReadTopic("topic"),
  topicoptions.WithReaderSupportSplitMergePartitions(true),
)

// режим совместимости (обработка автомасштабирования на сервере)
reader, err := db.Topic().StartReader(
  "consumer",
  topicoptions.ReadTopic("topic"),
```

```
  topicoptions.WithReaderSupportSplitMergePartitions(false),
)
```

Python

Включение автомасштабирования топика во время его создания производится с помощью аргумента `auto_partitioning_settings` у `create_topic`:

Native SDK

```
driver.topic_client.create_topic(
    topic,
    consumers=[consumer],
    min_active_partitions=10,
    max_active_partitions=100,
    auto_partitioning_settings=ydb.TopicAutoPartitioningSettings(
        strategy=ydb.TopicAutoPartitioningStrategy.SCALE_UP,
        up_utilization_percent=80,
        down_utilization_percent=20,
        stabilization_window=datetime.timedelta(seconds=300),
    ),
)
```

Native SDK (Asyncio)

```
await driver.topic_client.create_topic(
    topic,
    consumers=[consumer],
    min_active_partitions=10,
    max_active_partitions=100,
    auto_partitioning_settings=ydb.TopicAutoPartitioningSettings(
        strategy=ydb.TopicAutoPartitioningStrategy.SCALE_UP,
        up_utilization_percent=80,
        down_utilization_percent=20,
        stabilization_window=datetime.timedelta(seconds=300),
    ),
)
```

Внесение изменений в существующий топик производится с помощью аргумента `alter_auto_partitioning_settings` у `alter_topic`:

```
    driver.topic_client.alter_topic(
        topic_path,
        alter_auto_partitioning_settings=ydb.TopicAlterAutoPartitioningSettings(
            set_strategy=ydb.TopicAutoPartitioningStrategy.SCALE_UP,
            set_up_utilization_percent=80,
            set_down_utilization_percent=20,
            set_stabilization_window=datetime.timedelta(seconds=300),
        ),
    )
```

SDK поддерживает два режима чтения топиков с включенным автомасштабированием: режим полной поддержки и режим совместимости. Режим чтения задаётся аргументом `auto_partitioning_support` во время создания читателя. По умолчанию используется режим полной поддержки.

```
reader = driver.topic_client.reader(
    topic,
    consumer,
    auto_partitioning_support=True, # Full support is enabled
)

# or

reader = driver.topic_client.reader(
    topic,
    consumer,
    auto_partitioning_support=False, # Compatibility mode is enabled
)
```

С практической точки зрения для конечного пользователя режимы не отличаются. Режим полной поддержки отличается от режима совместимости тем, кто гарантирует порядок чтения — клиент или сервер. Режим совместимости достигается серверной обработкой и, как правило, работает медленнее.

JavaScript

Функциональность на данный момент не поддерживается.

Java

Функциональность на данный момент не поддерживается.

C#

Функциональность на данный момент не поддерживается.

Rust

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#311

PHP

Функциональность на данный момент не поддерживается.

Подтверждение обработки вне читателя

Чаще всего подтверждение обработки удобно выполнять в рамках читателя, получающего сообщения. Однако существуют сценарии, при которых подтверждение обработки должно производиться процессом, отличным от процесса чтения. В таком случае необходим способ подтверждения, находящийся вне читателя.

C++

Подтверждение обработки вне сессии чтения производится с помощью метода `NYdb::NTopic::TTopicClient::CommitOffset`:

```
#include <ydb-cpp-sdk/client/topic/client.h>

NYdb::NTopic::TTopicClient topicClient(driver);

NYdb::NStatusHelpers::ThrowOnError(topicClient.CommitOffset(
    topicPath,
    partitionId,
    consumerName,
    offset).GetValueSync());
```

Если в момент подтверждения существует активная сессия чтения (например через `CreateReadSession`), рекомендуется передать её идентификатор с помощью опции `ReadSessionId` в `NYdb::NTopic::TCommitOffsetSettings`. Это позволяет серверу не прерывать текущую сессию чтения:

```
// Получение идентификатора сессии чтения
std::string sessionId = readSession->GetSessionId();

NYdb::NStatusHelpers::ThrowOnError(topicClient.CommitOffset(
    topicPath,
    partitionId,
    consumerName,
    offset,
    NYdb::NTopic::TCommitOffsetSettings()
        .ReadSessionId(sessionId)
).GetValueSync());
```

Go

Подтверждение обработки вне читателя производится с помощью метода `db.Topic().CommitOffset`:

```
// Базовый способ — подтверждение оффсета без активной сессии чтения
err := db.Topic().CommitOffset(
  ctx,
  topicPath,
  partitionID,
  consumer,
  offset,
)
```

Если в момент подтверждения существует активная сессия чтения (через `StartReader` или `StartListener`), рекомендуется передать её идентификатор с помощью опции `WithCommitOffsetReadSessionID`. Это позволяет серверу не прерывать текущую сессию чтения:

```
import (
  // ...
  "github.com/ydb-platform/ydb-go-sdk/v3/topic/topicoptions"
)

// Получение идентификатора сессии чтения
sessionID := reader.ReadSessionID()
// или: sessionID := listener.ReadSessionID()

err = db.Topic().CommitOffset(
  ctx,
  topicPath,
  partitionID,
  consumer,
  offset,
  topicoptions.WithCommitOffsetReadSessionID(sessionID),
)
```

Python

Подтверждение обработки вне читателя производится с помощью метода `topic_client.commit_offset`:

Native SDK

```
driver.topic_client.commit_offset(
    topic_path,
    consumer_name,
    partition_id,
```

```
    offset,
    reader.read_session_id,  # опционально: не прерывает активную сессию чтения
)
```

Native SDK (Asyncio)

```
await driver.topic_client.commit_offset(
    topic_path,
    consumer_name,
    partition_id,
    offset,
    reader.read_session_id,  # опционально: не прерывает активную сессию чтения
)
```

JavaScript

Функциональность на данный момент не поддерживается.

Java

```
TopicClient client = ...;

String sessionID = reader.getSessionId();
// У AsyncReader идентификатор сессии можно получить при обработке события SessionStartedEvent

client.commitOffset(
    topicPath,
    CommitOffsetSettings.newBuilder()
        .setReadSessionId(sessionID)
        .setPartitionId(partitionID)
        .setConsumer(consumer)
        .setOffset(offset)
        .build()
).join().expectSuccess("Error commit!");
```

C#

Функциональность на данный момент не поддерживается.

Rust

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#330

PHP

Функциональность на данный момент не поддерживается.

Работа с узлами координации

Данная статья описывает как использовать YDB SDK для координации работы нескольких экземпляров клиентского приложения посредством использования узлов координации и находящихся в них семафоров.

Создание узла координации

Узлы координации создаются в базах данных YDB в том же пространстве имён, что и другие объекты схемы, такие как таблицы и топики.

C++

```
TClient client(driver);
auto status = client
    .CreateNode("/path/to/mynode")
    .ExtractValueSync();
Y_ABORT_UNLESS(status.IsSuccess());
```

При создании можно опционально указать `TNodeSettings` со следующими настройками:

- `ReadConsistencyMode` - по умолчанию `RELAXED`, что допускает чтение не самого свежего значения в случае смены лидера. Опционально можно включить `STRICT` режим чтения, при котором все чтения проходят через алгоритм консенсуса и гарантируют возврат самого свежего значения, но становятся существенно дороже.
- `AttachConsistencyMode` - по умолчанию `STRICT`, что означает обязательное использование алгоритма консенсуса при восстановлении сессии. Опционально можно включить `RELAXED` режим восстановления сессии в случае сбоев, который отключает это требование. Расслабленный режим может потребоваться при очень большом количестве клиентов, позволяя восстанавливать сессию без прохождения через консенсус, что не влияет на общую корректность, но может усугублять чтение не самого свежего значения во время смены лидера, а также устаревание сессий в случае проблем.
- `SelfCheckPeriod` (по умолчанию 1 секунда) - периодичность с которой сервис производит проверки собственной живости. Не рекомендуется менять за исключением особых случаев.
  - Чем больше указанное значение, тем меньше нагрузка на сервер, но тем дольше возможная задержка между сменой лидера и тем, насколько оперативно об этом узнает сам сервис.
  - Чем меньше указанное значение, тем больше нагрузка на сервер и большая оперативность в детектировании проблем, но возможна генерация false positive когда сервис ошибочно детектирует проблемы.
- `SessionGracePeriod` (по умолчанию 10 секунд) - период, в течение которого новый лидер не закрывает открытые сессии, продлевая их.
  - Чем меньше значение, тем меньше окно, когда сессии от несуществующих клиентов, которые не успели сообщить о пропаже при смене лидера, будут удерживать семафоры и мешать другим клиентам.
  - Чем меньше значение, тем выше вероятность ложных срабатываний, когда живой лидер может завершить работу для перестраховки, так как не будет уверен, что этот период не закончился у нового лидера.
  - Должен быть строго больше, чем `SelfCheckPeriod`.

Go

```
err := db.Coordination().CreateNode(ctx,
    "/path/to/mynode",
)
```

Java

Для работы с узлами координации подключите артефакт Maven `ydb-sdk-coordination` (модуль `tech.ydb.coordination.*`). Узлы координации нужны, когда несколько экземпляров приложения должны согласовывать доступ к ресурсам — подробнее в разделе Узел координации.

Ниже — полный пример: подключение через `GrpcTransport`, создание узла и проверка через `describeNode`.

```
import tech.ydb.coordination.CoordinationClient;
import tech.ydb.coordination.description.NodeConfig;
import tech.ydb.core.grpc.GrpcTransport;

public class CreateCoordinationNodeExample {

    private static final String NODE_PATH_SUFFIX = "/path/to/mynode";

    public static void main(String[] args) {
        // Строка подключения из переменной окружения или локальный YDB по умолчанию
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString).build();
             CoordinationClient client = CoordinationClient.newClient(transport).build()) {

            // Полный путь к узлу = путь базы + имя узла в пространстве имён
            String nodePath = client.getDatabase() + NODE_PATH_SUFFIX;

            // Создаём узел координации с настройками по умолчанию
            client.createNode(nodePath).join().expectSuccess("не удалось создать узел");

            // Проверяем, что узел создан: читаем его конфигурацию
            NodeConfig config = client.describeNode(nodePath).join().getValue();
            System.out.println("Узел создан: " + nodePath);
            System.out.println("SelfCheckPeriod: " + config.getSelfCheckPeriod());
            System.out.println("SessionGracePeriod: " + config.getSessionGracePeriod());
        }
    }
}
```

При необходимости задайте конфигурацию узла через NodeConfig, используя цепочку `NodeConfig.create().with…`. Доступные параметры: периоды `SelfCheckPeriod` и `SessionGracePeriod`, режимы согласованности чтения и подключения сессии (`readConsistencyMode`, `attachConsistencyMode`), режим счётчиков ограничителя скорости (`rateLimiterCountersMode`). Значения по умолчанию совпадают с описанием для C++ (см. выше). Готовый `NodeConfig` передаётся в `CoordinationNodeSettings` и в `createNode(nodePath, settings)`.

Дополнительно доступны `alterNode` (изменение конфигурации) и `dropNode` (удаление узла).

Python

Native SDK

```
import ydb

client = driver.coordination_client
client.create_node("/path/to/mynode")
```

Native SDK (Asyncio)

```
import ydb

client = driver.coordination_client
await client.create_node("/path/to/mynode")
```

C#

Функциональность на данный момент не поддерживается.

JavaScript

```
import { CoordinationClient } from "@ydbjs/coordination";

let client = new CoordinationClient(driver);
await client.createNode("/path/to/mynode", {});
```

Rust

Клиент координации возвращается из `Client::coordination_client`. Узел создаётся через `CoordinationClient::create_node` с путём и `NodeConfig` (через `NodeConfigBuilder`). Также доступны `alter_node`, `drop_node`, `describe_node`. Полный пример — `mutex.rs`.

```
use ydb::NodeConfigBuilder;

let mut coordination_client = client.coordination_client();

coordination_client
    .create_node(
        "/path/to/mynode".into(),
        NodeConfigBuilder::default().build()?,
    )
    .await?;
```

PHP

Функциональность на данный момент не поддерживается.

Работа с сессиями

Создание сессии

Для начала работы клиент должен установить сессию, в рамках которой он будет осуществлять все операции с узлом координации.

C++

```
TClient client(driver);
const TSession& session = client
   .StartSession("/path/to/mynode")
   .ExtractValueSync()
   .ExtractResult();
```

При установке сессии можно опционально передать структуру `TSessionSettings` со следующими настройками:

- `Description` - текстовое описание сессии, отображается во внутренних интерфейсах и может быть полезно при диагностике проблем.
- `OnStateChanged` - вызывается на важных изменениях в процессе жизни сессии, передавая соответствующее состояние:
  - `ATTACHED` - сессия подключена и работает в нормальном режиме;
  - `DETACHED` - сессия временно потеряла связь с сервисом, но ещё может быть восстановлена;
  - `EXPIRED` - сессия потеряла связь с сервисом и не может быть восстановлена.
- `OnStopped` - вызывается, когда сессия прекращает попытки восстановить связь с сервисом, что может быть полезно для установления нового соединения.
- `Timeout` - максимальный таймаут, в течение которого сессия может быть восстановлена после потери связи с сервисом.

Go

```
session, err := db.Coordination().CreateSession(ctx,
    "/path/to/mynode", // имя Coordination Node в базе
)
```

Java

Перед работой с семафорами клиент открывает сессию (см. CoordinationSession): вызов `createSession` создаёт объект сессии, а `connect()` устанавливает двунаправленный gRPC-поток с узлом. Параметры повторных попыток и таймаут подключения задаются в CoordinationSessionSettings (`withConnectTimeout`, `withRetryPolicy`, `withExecutor`).

Типичный сценарий: после успешного `connect()` выполняете операции с семафорами, затем закрываете сессию через `close()` (удобно — try-with-resources). Пока сессия активна, SDK при сбоях сети сам повторяет подключение согласно настройкам.

```
import tech.ydb.coordination.CoordinationClient;
import tech.ydb.coordination.CoordinationSession;
import tech.ydb.coordination.settings.CoordinationSessionSettings;
import tech.ydb.core.grpc.GrpcTransport;

public class CoordinationSessionExample {

    private static final String NODE_PATH_SUFFIX = "/path/to/mynode";

    public static void main(String[] args) {
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString).build();
             CoordinationClient client = CoordinationClient.newClient(transport).build()) {

            String nodePath = client.getDatabase() + NODE_PATH_SUFFIX;
            client.createNode(nodePath).join().expectSuccess("не удалось создать узел");

            // try-with-resources гарантирует вызов close() и остановку потока с узлом
            try (CoordinationSession session = client.createSession(
                    nodePath,
                    CoordinationSessionSettings.newBuilder().build())) {

                // Устанавливаем соединение с узлом координации
                session.connect().join().expectSuccess("не удалось подключить сессию");
                System.out.println("Сессия подключена, id=" + session.getId());

                // ... операции с семафорами (см. раздел «Работа с семафорами») ...

            } // session.close() — явное завершение сессии
        }
    }
}
```

Python

Native SDK

```
import ydb

client = driver.coordination_client
with client.session("/path/to/mynode") as session:
    # работа с сессией
    pass
```

Native SDK (Asyncio)

```
import ydb

client = driver.coordination_client
async with client.session("/path/to/mynode") as session:
    # работа с сессией
    pass
```

C#

Функциональность на данный момент не поддерживается.

JavaScript

```
import { CoordinationClient } from "@ydbjs/coordination";

let client = new CoordinationClient(driver);
await using session = await client.createSession("/path/to/mynode", {}, signal);
```

Rust

Сессию создаёт `CoordinationClient::create_session` с путём к узлу и `SessionOptions` (`SessionOptionsBuilder`: таймаут, описание и т.д.). Поток с узлом поднимается внутри конструктора сессии; отдельного вызова `connect`, как в Java, нет.

```rust
use ydb::SessionOptionsBuilder;

let session = coordination_client
    .create_session(
        "/path/to/mynode".into(),
        SessionOptionsBuilder::default().build()?,
    )
    .await?;
```

PHP

Функциональность на данный момент не поддерживается.

Контроль завершения сессии

Клиентскому приложению необходимо следить за состоянием сессии, так как оно может полагаться на состояние захваченных семафоров только пока сессия активна. Когда сессия завершается по инициативе клиента или сервера, клиент больше не может быть уверен, что другие клиенты в кластере не захватили его семафоры и не изменили их состояние.

C++

В C++ SDK установленная сессия в фоне поддерживает и автоматически восстанавливает связь с кластером YDB.

Go

В Go SDK для отслеживания таких ситуаций используется контекст сессии `session.Context()`, который завершается вместе с сессией. SDK самостоятельно обрабатывает ошибки транспортного уровня и восстанавливает соединение с сервисом, пытаясь восстановить сессию, если это возможно. Таким образом, клиенту достаточно следить только за контекстом сессии, чтобы своевременно отреагировать на её потерю.

Python

В Python SDK сессия автоматически восстанавливает связь с кластером YDB при сбоях. Рекомендуется использовать контекстный менеджер (`with` или `async with`) для гарантированного закрытия сессии при выходе из блока. При работе с семафорами через контекстный менеджер (`with session.semaphore(name)` или `async with session.semaphore(name)`) семафор автоматически освобождается при выходе из блока, а сессия — при закрытии контекста.

C#

Функциональность на данный момент не поддерживается.

JavaScript

В JS SDK для отслеживания таких ситуаций используется сигнал `session.signal`, который прерывается вместе с сессией. SDK самостоятельно обрабатывает ошибки транспортного уровня и восстанавливает соединение с сервисом, пытаясь восстановить сессию, если это возможно. Таким образом, клиенту достаточно следить за сигналом сессии, чтобы не совершать действий когда сессия была закрыта или просрочена.

Также в JavaScript SDK есть метод для получения новой сессии при утрате старой, и этот способ является рекомендованным для длительного использования `for await (session of client.openSession()) { session.signal }`.

Java

Завершите сессию (`close()`), когда ваш сценарий отработал: так вы явно освободите соединение с узлом. Пока сессия не закрыта, SDK при сбоях сети сам повторяет подключение согласно `CoordinationSessionSettings`. Семафор держите только на время решения пользовательской задачи и отпускайте через `SemaphoreLease.release()`, когда ресурс больше не нужен.

Rust

У `CoordinationSession` вызовите `alive`: вернётся `CancellationToken` — при завершении сессии он отменяется (аналог отслеживания контекста в Go). При отпускании `Lease` или при `Drop` сессии освобождение семафора уходит на сервер в фоне.

PHP

Функциональность на данный момент не поддерживается.

Работа с семафорами

Создание семафора

При создании семафора можно указать его лимит. Лимит определяет максимальное значение, на которое его можно увеличить. Вызовы, пытающиеся увеличить значение семафора выше этого лимита, начнут ждать, пока их запросы на увеличение смогут быть выполнены, так чтобы значение семафора не превышало его лимит.

C++

```rust
session
    .CreateSemaphore(
        "my-semaphore",  // semaphore name
        10               // semaphore limit
    )
    .ExtractValueSync()
    .ExtractResult();
```

Также при создании семафора можно передать строку, которая будет храниться вместе с семафором и возвращаться при его захвате:

```rust
session
    .CreateSemaphore(
        "my-semaphore",  // semaphore name
        10,              // semaphore limit
```

```
        "my-data"        // semaphore data
    )
    .ExtractValueSync()
    .ExtractResult();
```

Go

```
err := session.CreateSemaphore(ctx,
    "my-semaphore", // semaphore name
    10              // semaphore limit
)
```

Python

В Python SDK семафор создаётся неявно при первом вызове `acquire()` в методе `session.semaphore(name, limit)`. Лимит указывается при создании объекта семафора.

Native SDK

```
import ydb

client = driver.coordination_client
with client.session("/path/to/mynode") as session:
    # семафор будет создан при первом acquire() с лимитом 10
    semaphore = session.semaphore("my-semaphore", 10)
```

Native SDK (Asyncio)

```
import ydb

client = driver.coordination_client
async with client.session("/path/to/mynode") as session:
    # семафор будет создан при первом acquire() с лимитом 10
    semaphore = session.semaphore("my-semaphore", 10)
```

C#

Функциональность на данный момент не поддерживается.

JavaScript

```
const sem = session.semaphore("connections");
await sem.create({
  limit: 10,
  data: new Uint8Array(),
});
```

Java

Ниже — один полный пример жизненного цикла семафора: создание узла и сессии, создание семафора, захват, обновление и чтение данных, освобождение. Персистентный семафор нужно создать явно (`createSemaphore`); эфемерные семафоры создаются при первом захвате (см. раздел «Захват семафора»).

```
import java.nio.charset.StandardCharsets;
import java.time.Duration;

import tech.ydb.coordination.CoordinationClient;
import tech.ydb.coordination.CoordinationSession;
import tech.ydb.coordination.SemaphoreLease;
import tech.ydb.coordination.description.SemaphoreDescription;
import tech.ydb.coordination.settings.DescribeSemaphoreMode;
import tech.ydb.core.grpc.GrpcTransport;

public class CoordinationSemaphoreExample {

    private static final String NODE_PATH_SUFFIX = "/path/to/mynode";
    private static final String SEMAPHORE_NAME = "my-semaphore";

    public static void main(String[] args) {
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString).build();
             CoordinationClient client = CoordinationClient.newClient(transport).build()) {

            String nodePath = client.getDatabase() + NODE_PATH_SUFFIX;

            // 1. Создаём узел координации
            client.createNode(nodePath).join().expectSuccess("не удалось создать узел");

            try (CoordinationSession session = client.createSession(nodePath)) {
                session.connect().join().expectSuccess("не удалось подключить сессию");

                byte[] initialData = "my-data".getBytes(StandardCharsets.UTF_8);

                // 2. Создаём семафор с лимитом 10 и начальными данными
                session.createSemaphore(SEMAPHORE_NAME, 10, initialData)
```

```
                        .join().expectSuccess("не удалось создать семафор");

                // 3. Захватываем 5 токенов; ждём в очереди не более 30 секунд
                SemaphoreLease lease = session
                        .acquireSemaphore(SEMAPHORE_NAME, 5, Duration.ofSeconds(30))
                        .join().getValue();

                try {
                    // 4. Обновляем данные, прикреплённые к семафору
                    byte[] updatedData = "updated-data".getBytes(StandardCharsets.UTF_8);
                    session.updateSemaphore(SEMAPHORE_NAME, updatedData)
                            .join().expectSuccess("не удалось обновить данные семафора");

                    // 5. Читаем текущее состояние семафора
                    SemaphoreDescription description = session
                            .describeSemaphore(SEMAPHORE_NAME, DescribeSemaphoreMode.DATA_ONLY)
                            .join().getValue();

                    System.out.println("Имя: " + description.getName());
                    System.out.println("Лимит: " + description.getLimit());
                    System.out.println("Захвачено: " + description.getCount());
                    System.out.println("Данные: "
                            + new String(description.getData(), StandardCharsets.UTF_8));

                } finally {
                    // 6. Освобождаем захваченные токены
                    lease.release().join().expectSuccess("не удалось освободить семафор");
                }
            }
        }
    }
}
```

Если семафор с таким именем уже существует, `createSemaphore` вернёт статус «уже существует». Вариант без параметра `data` эквивалентен передаче `null`.

Rust

`CoordinationSession::create_semaphore` принимает имя, лимит и произвольные байты `data`, хранимые у семафора.

```
session.create_semaphore("my-semaphore", 10, vec![]).await?;

// или с пользовательскими данными, хранимыми у семафора:
session
    .create_semaphore("other-semaphore", 10, b"my-data".to_vec())
    .await?;
```

PHP

Функциональность на данный момент не поддерживается.

Захват семафора

Чтобы захватить семафор, клиент должен вызвать метод `AcquireSemaphore` и дождаться получения специального объекта `Lease`. Этот объект представляет собой подтверждение о том, что значение семафора было успешно увеличено и может считаться таковым до явного отпускания такого семафора или завершения сессии, в которой такое подтверждение было получено.

C++

```
session
    .AcquireSemaphore(
        "my-semaphore",                      // semaphore name
        TAcquireSemaphoreSettings().Count(5)  // value to increase semaphore by
    )
    .ExtractValueSync()
    .ExtractResult();
```

При захвате можно опционально передать структуру `TAcquireSemaphoreSettings` со следующими настройками:

- `Count` - значение, на которое увеличивается семафор при захвате.
- `Data` - дополнительные данные, которые можно положить в семафор.
- `OnAccepted` - вызывается, когда операция встаёт в очередь (например, если семафор невозможно было захватить сразу).
  - Не будет вызвано, если семафор захватывается сразу.
  - Важно учитывать, что вызов может произойти параллельно с результатом `TFuture`.
- `Timeout` - максимальное время, в течение которого операция может пролежать в очереди на сервере.
  - Операция вернёт `false`, если за время `Timeout` после добавления в очередь не удалось захватить семафор.
  - При `Timeout` установленном в 0 операция по смыслу работает как `TryAcquire`, т.е. семафор будет либо захвачен атомарно и операция вернёт `true`, либо операция вернёт `false` без использования очередей.
- `Ephemeral` - если `true`, то имя является эфемерным семафором, такие семафоры автоматически создаются при первом `Acquire` и автоматически удаляются с последним `Release`.
- `Shared()` - алиас для выставления `Count = 1`, захват семафора в shared режиме.
- `Exclusive()` - алиас для выставления `Count = max`, захват семафора в exclusive режиме (для семафоров, созданных с лимитом `Max<ui64>()`).

Go

```
lease, err := session.AcquireSemaphore(ctx,
    "my-semaphore",  // semaphore name
    5,              // value to increase semaphore by
)
```

Для отмены ожидания взятия семафора, достаточно отменить переданный в метод контекст `ctx`.

Python

Native SDK

```
import ydb

client = driver.coordination_client
with client.session("/path/to/mynode") as session:
    semaphore = session.semaphore("my-semaphore", 10)
    with semaphore:
        # семафор захвачен на 1 единицу (значение по умолчанию)
        pass
    # или вручную:
    semaphore = session.semaphore("my-semaphore", 10)
    semaphore.acquire(count=5)
    # работа с ресурсом
    semaphore.release()
```

Native SDK (Asyncio)

```
import ydb

client = driver.coordination_client
async with client.session("/path/to/mynode") as session:
    semaphore = session.semaphore("my-semaphore", 10)
    async with semaphore:
        # семафор захвачен на 1 единицу (значение по умолчанию)
        pass
    # или вручную:
    semaphore = session.semaphore("my-semaphore", 10)
    await semaphore.acquire(count=5)
    # работа с ресурсом
    await semaphore.release()
```

C#

Функциональность на данный момент не поддерживается.

JavaScript

```
{
  await using lease = await sem.acquire({ count: 1, data: new Uint8Array() });
  await doWork(lease.signal);
} // lease.release() called automatically
```

Java

Захват выполняется через `acquireSemaphore` (полный пример — в разделе «Создание семафора»). Метод принимает имя семафора, число токенов `count`, опциональные данные операции и таймаут ожидания в очереди java.time.Duration. Возвращает `CompletableFuture<Result<SemaphoreLease>>` (см. Result и SemaphoreLease). Если семафор с указанным именем не существует, операция завершится исключением.

Для **эфемерных** семафоров используйте `acquireEphemeralSemaphore` (флаг `exclusive` задаёт режим захвата); такие семафоры создаются при первом захвате и удаляются после последнего освобождения.

В один момент времени сессия может удерживать **только один** семафор; повторные вызовы для того же имени **заменяют** предыдущую операцию (например, чтобы уменьшить `count` или сменить таймаут).

Rust

`acquire_semaphore` возвращает `Lease`. Таймаут ожидания в очереди, эфемерность и данные операции задаются через `AcquireOptionsBuilder` и `acquire_semaphore_with_params`.

```
use std::time::Duration;
use ydb::AcquireOptionsBuilder;

let _lease = session.acquire_semaphore("my-semaphore", 5).await?;

let opts = AcquireOptionsBuilder::default()
    .timeout(Duration::from_secs(30))
    .build()?;
let _lease = session
    .acquire_semaphore_with_params("my-semaphore", 5, opts)
    .await?;
```

PHP

Функциональность на данный момент не поддерживается.

Взятое значение захваченного семафора можно снизить (но не увеличить), вновь вызвав для него метод `AcquireSemaphore` с меньшим значением.

Обновление данных семафора

С помощью метода `UpdateSemaphore` можно обновить (заменить) данные семафора, которые были привязаны при его создании.

C++

```
session
    .UpdateSemaphore(
        "my-semaphore",  // semaphore name
        "updated-data"   // new semaphore data
    )
    .ExtractValueSync()
    .ExtractResult();
```

Go

```
err := session.UpdateSemaphore(
    "my-semaphore",                                                    // semaphore name
    options.WithUpdateData([]byte("updated-data")),   // new semaphore data
)
```

Python

Native SDK

```
import ydb

client = driver.coordination_client
with client.session("/path/to/mynode") as session:
    semaphore = session.semaphore("my-semaphore", 10)
    semaphore.update(b"updated-data")
```

Native SDK (Asyncio)

```
import ydb

client = driver.coordination_client
async with client.session("/path/to/mynode") as session:
    semaphore = session.semaphore("my-semaphore", 10)
    await semaphore.update(b"updated-data")
```

C#

Функциональность на данный момент не поддерживается.

JavaScript

```
const sem = session.semaphore("connections");
await sem.update({
  limit: 5,
  data: new Uint8Array(),
});
```

Java

Обновление данных — метод `updateSemaphore` (шаг 4 в примере раздела «Создание семафора»). Вызов не требует захвата семафора и не приводит к нему.

Rust

```
session
    .update_semaphore("my-semaphore", b"updated-data".to_vec())
    .await?;
```

PHP

Функциональность на данный момент не поддерживается.

Этот вызов не требует захвата семафора и не приводит к нему. Если требуется, чтобы данные обновлял только один конкретный клиент, то это необходимо явным образом обеспечить, например, захватив семафор, обновив данные и отпустив семафор обратно.

Получение данных семафора

C++

```
session
    .DescribeSemaphore(
        "my-semaphore"  // semaphore name
    )
```

```
    .ExtractValueSync()
    .ExtractResult();
```

При получении информации о семафоре можно опционально передать структуру `TDescribeSemaphoreSettings` со следующими настройками:

- `OnChanged` - вызывается один раз после изменения данных на сервере. С параметром `bool`, если `true` - то вызов произошёл из-за каких-то изменений, если `false` - то это ложный вызов и необходимо повторить `DescribeSemaphore` для восстановления подписки.
- `WatchData` - вызывать `OnChanged` в случае изменения данных семафора.
- `WatchOwners` - вызывать `OnChanged` в случае изменения владельцев семафора.
- `IncludeOwners` - вернуть список владельцев в результатах.
- `IncludeWaiters` - вернуть список ожидающих в результатах.

Результат вызова представляет собой структуру со следующими полями:

- `Name` - имя семафора.
- `Data` - данные семафора.
- `Count` - текущее значение семафора.
- `Limit` - максимальное количество токенов, указанное при создании семафора.
- `Owners` - список владельцев семафора.
- `Waiters` - список ожидающих в очереди на семафоре.
- `Ephemeral` - является ли семафор эфемерным.

Поля `Owners` и `Waiters` в результате представляют собой список структур со следующими полями:

- `OrderId` - порядковый номер операции захвата на семафоре. Может использоваться для идентификации, например если `OrderId` изменился, значит сессия сделала `ReleaseSemaphore` и новый `AcquireSemaphore`.
- `SessionId` - идентификатор сессии, которая делала данный `AcquireSemaphore`.
- `Timeout` - таймаут, с которым вызывался `AcquireSemaphore` для операций в очереди.
- `Count` - запрошенное в `AcquireSemaphore` значение.
- `Data` - данные, которые были указаны в `AcquireSemaphore`.

Go

```
description, err := session.DescribeSemaphore(
    "my-semaphore"                              // semaphore name
    options.WithDescribeOwners(true), // to get list of owners
    options.WithDescribeWaiters(true), // to get list of waiters
)
```

Python

Native SDK

```
import ydb

client = driver.coordination_client
with client.session("/path/to/mynode") as session:
    semaphore = session.semaphore("my-semaphore", 10)
    description = semaphore.describe()
    # description содержит: name, data, count, limit, owners, waiters, ephemeral
```

Native SDK (Asyncio)

```
import ydb

client = driver.coordination_client
async with client.session("/path/to/mynode") as session:
    semaphore = session.semaphore("my-semaphore", 10)
    description = await semaphore.describe()
    # description содержит: name, data, count, limit, owners, waiters, ephemeral
```

C#

Функциональность на данный момент не поддерживается.

JavaScript

```
const sem = session.semaphore("connections");
await sem.describe({
  owners: true,
  waiters: true,
});
```

Java

Чтение состояния семафора — метод `describeSemaphore` (шаг 5 в примере раздела «Создание семафора»). Принимает имя семафора и режим DescribeSemaphoreMode: только данные, со списком владельцев, со списком ожидающих или оба списка.

У элементов списков владельцев и ожидающих (`getOwnersList`, `getWaitersList`) доступны идентификатор сессии, таймаут, запрошенный `count`, данные операции и `orderId` (см. вложенный тип `SemaphoreDescription.Session` в исходниках).

Для подписки на изменения используйте `watchSemaphore` с тем же режимом описания и WatchSemaphoreMode (данные, владельцы или оба). Объект SemaphoreWatcher содержит снимок `SemaphoreDescription` и `getChangedFuture()` — `CompletableFuture<Result<SemaphoreChangedEvent>>` (см. SemaphoreChangedEvent, поля `isDataChanged`, `isOwnersChanged`). Future завершится при следующем событии; после уведомления для продолжения наблюдения вызовите `watchSemaphore` снова (см. тесты).

Rust

`describe_semaphore` по умолчанию запрашивает владельцев и ожидающих. Набор флагов можно задать через `DescribeOptions` и `describe_semaphore_with_params`. Для подписки на изменения смотрите `WatchOptions` в документации крейта.

```rust
let description = session.describe_semaphore("my-semaphore").await?;
```

PHP

Функциональность на данный момент не поддерживается.

Освобождение семафора

C++

```
session
    .ReleaseSemaphore(
        "my-semaphore"  // semaphore name
    )
    .ExtractValueSync()
    .ExtractResult();
```

Go

Чтобы отпустить захваченный в сессии семафор, необходимо вызвать метод `Release` у объекта `Lease`.

```
err := lease.Release()
```

Python

В Python SDK семафор освобождается методом `release()` у объекта семафора. При использовании контекстного менеджера (`with` или `async with`) освобождение происходит автоматически при выходе из блока.

Native SDK

```
import ydb

client = driver.coordination_client
with client.session("/path/to/mynode") as session:
    semaphore = session.semaphore("my-semaphore", 10)
    semaphore.acquire(count=5)
    # работа с ресурсом
    semaphore.release()
```

Native SDK (Asyncio)

```
import ydb

client = driver.coordination_client
async with client.session("/path/to/mynode") as session:
    semaphore = session.semaphore("my-semaphore", 10)
    await semaphore.acquire(count=5)
    # работа с ресурсом
    await semaphore.release()
```

C#

Функциональность на данный момент не поддерживается.

JavaScript

Чтобы отпустить захваченный в сессии семафор, необходимо вызвать метод `Release` у объекта `Lease`. Если взятие семафора было с использованием конструкции using, то при выходе из скоупа, семафор будет освобожден автоматически.

```
await lease.release();
```

Java

Освобождение — через SemaphoreLease.release() (шаг 6 в примере раздела «Создание семафора»). Метод асинхронный, возвращает `CompletableFuture<Status>`.

Rust

Вызовите `Lease::release` или просто завершите владение `Lease` — при уничтожении значения также отправляется освобождение на сервер.

```
let lease = session.acquire_semaphore("my-semaphore", 1).await?;
// …
```

```
lease.release();
```

PHP

Функциональность на данный момент не поддерживается.

Важные особенности

Операции `AcquireSemaphore` и `ReleaseSemaphore` являются идемпотентными. Если на семафоре был вызван `AcquireSemaphore`, повторные вызовы `AcquireSemaphore` изменяют только параметры захвата. Например, вызов `AcquireSemaphore` с `count=10` может добавить операцию в очередь. До или после успешного захвата можно повторно вызвать `AcquireSemaphore` с `count=9`, уменьшая количество захваченных единиц; новая операция заменит старую (которая завершится с кодом `ABORTED`, если она ещё не была успешно завершена). Позиция в очереди при этом не изменяется, несмотря на замену одной операции `AcquireSemaphore` на другую.

Операции `AcquireSemaphore` и `ReleaseSemaphore` возвращают `bool`, указывающий, изменила ли операция состояние семафора. Например, `AcquireSemaphore` вернёт `false`, если захват семафора не удался в течение времени `Timeout`, так как он был захвачен другим. Операция `ReleaseSemaphore` может вернуть `false`, если семафор не захвачен в текущей сессии.

Операцию `AcquireSemaphore`, находящуюся в очереди, можно завершить досрочно, вызвав `ReleaseSemaphore`. Независимо от количества вызовов `AcquireSemaphore` для конкретного семафора в одной сессии, освобождение происходит одним вызовом `ReleaseSemaphore`, то есть операции `AcquireSemaphore` и `ReleaseSemaphore` нельзя использовать как аналог `Acquire` или `Release` на рекурсивном мьютексе.

Операция `DescribeSemaphore` с флагами `WatchData` или `WatchOwners` создаёт подписку на изменения семафора. Любая более старая подписка на тот же семафор в сессии отменяется, вызывая `OnChanged(false)`. Рекомендуется игнорировать `OnChanged` от предыдущих вызовов `DescribeSemaphore`, если выполняется новый замещающий вызов, например, запоминая текущий id вызова.

Вызов `OnChanged(false)` может происходить не только из-за отмены новым `DescribeSemaphore`, но и по другим причинам, например, при временном разрыве соединения между grpc клиентом и сервером, при временном разрыве соединения между grpc сервером и текущим лидером сервиса, при изменении лидера сервиса, то есть при малейшем подозрении, что нотификация могла быть потеряна. Для восстановления подписки клиентский код должен выполнить новый вызов `DescribeSemaphore`, правильно обрабатывая ситуацию, что результат нового вызова может быть другим (например, если нотификация действительно была потеряна).

Примеры

- Распределённая блокировка
- Выбор лидера
- Обнаружение сервисов
- Публикация конфигурации

Обработка ошибок

При использовании YDB SDK возникают ситуации, в которых необходимо обрабатывать ошибки.

Ошибки можно разделить на три категории:

- **Временные сбои** (retryable). Включают кратковременную потерю сетевого соединения, временную недоступность или перегруженность одной из подсистем YDB, а также неспособность YDB ответить на запрос в течение установленного времени ожидания. В случае возникновения таких ошибок повторный запрос через некоторый промежуток времени с высокой вероятностью будет выполнен успешно.
- **Ошибки, которые не могут быть исправлены с помощью повтора** (non-retryable). Включают некорректно сформированные запросы, внутренние ошибки YDB, а также запросы, не соответствующие схеме данных. В такой ситуации нет необходимости повторять запрос, требуется дополнительное вмешательство разработчика.
- **Ошибки, которые, предположительно, могут быть исправлены с помощью повтора после реакции клиентского приложения** (conditionally retryable). Возникают при идемпотентных операциях, таких как отсутствие ответа в течение отведённого времени или запрос аутентификации.

Обработка временных сбоев (retryable errors)

YDB SDK предоставляет встроенный механизм обработки временных сбоев. По умолчанию в SDK используется рекомендованная политика повторов, которую можно изменить в соответствии с требованиями клиентского приложения. YDB возвращает коды завершения, позволяющие определить, уместна ли повторная попытка и какой выбрать интервал.

Операции следует повторять только в том случае, если ошибки являются временными сбоями. Не пытайтесь повторно выполнять недопустимые операции, такие как вставка в таблицу строки с существующим значением первичного ключа или вставка данных, не соответствующих схеме таблицы.

Крайне важно оптимизировать число повторных попыток и интервал между ними. Чрезмерное количество повторных попыток и слишком короткий интервал создают избыточную нагрузку. Недостаточное количество повторных попыток не позволит завершить выполнение операции.

Встроенные в SDK YDB retryer'ы используют следующие стратегии задержек повторного выполнения запросов в зависимости от статуса завершения запроса:

- **Немедленный повтор.** Повторные попытки совершаются немедленно.
- **Короткая экспоненциальная задержка.** Первоначальный интервал составляет несколько миллисекунд. Для каждой последующей попытки интервал увеличивается по экспоненциальному закону.
- **Большая экспоненциальная задержка.** Первоначальный интервал составляет несколько секунд. Для каждой последующей попытки интервал увеличивается по экспоненциальному закону.

При выборе интервала вручную обычно используются следующие стратегии:

- **Экспоненциальная задержка.** Для каждой последующей попытки интервал увеличивается по экспоненциальному закону.
- **Интервалы с приращениями.** Для каждой последующей попытки интервал увеличивается с определёнными приращениями.
- **Постоянные интервалы.** Повторные попытки совершаются через одинаковые интервалы.
- **Немедленный повтор.** Повторные попытки совершаются немедленно.
- **Случайный выбор.** Повторные попытки совершаются через случайно выбранный интервал времени.

При выборе интервала и числа повторных попыток учитывайте статусы завершения YDB.

Не используйте бесконечные повторные попытки — это может привести к избыточной нагрузке.

Не выполняйте немедленный повтор более одного раза.

Примеры кода смотрите в разделе Выполнение повторных попыток.

Статусы завершения

Когда возникает ошибка, YDB SDK возвращает объект ошибки, который включает статусы завершения. Возвращаемый статус завершения может исходить от сервера YDB, транспорта gRPC или самого SDK.

Статусы завершения в диапазоне 400000–400999 — это статусы завершения, полученные от сервера YDB. Смотрите Статусы завершения сервера YDB.

Статусы завершения в диапазоне 401000–401999 относятся к кодам, специфичным для SDK. Для получения дополнительной информации об этих статусах, относящихся к конкретному SDK, обратитесь к соответствующей документации SDK.

Для получения дополнительной информации о статусах завершения gRPC см. документацию gRPC.

Журналирование ошибок

При использовании SDK рекомендуется записывать в журнал все ошибки и исключения:

- Журналируйте количество предпринимаемых повторных попыток. Увеличение числа регулярных повторных попыток часто указывает на наличие проблем.
- Журналируйте все ошибки, в том числе их типы, коды завершения и причины возникновения.
- Журналируйте общее время выполнения операций, включая операции, завершившиеся после повторного выполнения.

Смотрите также

- Каскадный эффект при частых повторных попытках
- Статусы завершения gRPC
- Статусы завершения сервера YDB
- Ошибки

Статусы завершения сервера YDB

Код	Статус	Возможность повтора	Стратегия задержек	Пересоздать сессию
400000	SUCCESS	–	–	–
400010	BAD_REQUEST	неповторяемый[2]	–	нет
400020	UNAUTHORIZED	неповторяемый[2]	–	нет
400030	INTERNAL_ERROR	неповторяемый[2]	–	нет
400040	ABORTED	повторяемый[3]	короткая[4]	нет
400050	UNAVAILABLE	повторяемый[3]	короткая[4]	нет
400060	OVERLOADED	повторяемый[3]	большая[5]	нет
400070	SCHEME_ERROR	неповторяемый[2]	–	нет
400080	GENERIC_ERROR	неповторяемый[2]	–	нет
400090	TIMEOUT	неповторяемый[2]	–	нет
400100	BAD_SESSION	повторяемый[3]	моментально[6]	да
400120	PRECONDITION_FAILED	неповторяемый[2]	–	нет
400130	ALREADY_EXISTS	неповторяемый[2]	–	нет
400140	NOT_FOUND	неповторяемый[2]	–	нет
400150	SESSION_EXPIRED	повторяемый[3]	моментально[6]	да
400160	CANCELLED	неповторяемый[2]	–	нет
400170	UNDETERMINED	условно повторяемый[7]	короткая[4]	нет
400180	UNSUPPORTED	неповторяемый[2]	–	нет
400190	SESSION_BUSY	повторяемый[3]	короткая[4]	да
400200	EXTERNAL_ERROR	неповторяемый[2]	–	нет

400000: SUCCESS

Не является ошибкой — запрос успешно обработан.

400010: BAD_REQUEST Неповторяемый

Ошибка в синтаксисе запроса, пропущены обязательные поля.

Проверьте запрос.

400020: UNAUTHORIZED Неповторяемый

Отсутствует доступ к запрашиваемому схемному объекту (таблица, директория).

Запросите доступ у администратора базы данных.

400030: INTERNAL_ERROR Неповторяемый

Неизвестная внутренняя ошибка.

Зарегистрируйте проблему на GitHub или обратитесь к технической поддержке YDB.

400040: ABORTED Повторяемый | Короткая задержка

Операция не выполнена (например, по причине инвалидации локов, `TRANSACTION_LOCKS_INVALIDATED` в подробных сообщениях об ошибке).

Повторите всю транзакцию.

400050: UNAVAILABLE Повторяемый | Короткая задержка

Часть системы недоступна.

Повторите последнее действие (запрос).

400060: OVERLOADED Повторяемый | Большая задержка

Часть системы перегружена.

Повторите последнее действие (запрос), снизьте интенсивность потока запросов.

400070: SCHEME_ERROR Неповторяемый

Запрос не соответствует схеме.

Исправьте запрос или схему.

400080: GENERIC_ERROR Неповторяемый

Неклассифицируемая ошибка, возможно, связанная с запросом.

Ознакомьтесь с подробным сообщением об ошибке. При необходимости зарегистрируйте проблему на GitHub или обратитесь к технической поддержке YDB.

400090: TIMEOUT Условно повторяемый | Моментально

Запрос не выполнен за отведённое время.

Можно повторить для идемпотентных запросов.

400100: BAD_SESSION Повторяемый | Моментально

Данная сессия больше недоступна.

Пересоздайте сессию.

400120: PRECONDITION_FAILED Неповторяемый

Запрос не может быть выполнен в текущем состоянии (например, вставка в таблицу с существующим ключом).

Исправьте состояние или запрос и повторите попытку.

400130: ALREADY_EXISTS Неповторяемый

Объект базы данных, который создаётся, уже существует в кластере YDB.

Ответ зависит от логики приложения.

400140: NOT_FOUND Неповторяемый

Объект базы данных не найден в YDB.

Ответ зависит от логики приложения.

400150: SESSION_EXPIRED Условно повторяемый | Моментально

Срок действия сессии уже истёк.

Пересоздать сессию.

400160: CANCELLED Неповторяемый

Запрос был отменён на сервере. Например, пользователь отменил запрос во YDB UI, который выполнялся слишком долго, запрос был сделан с опцией таймаута cancel_after, или сессия была закрыта во время выполнения запроса.

Если сессия была закрыта, создайте новую сессию перед отправкой следующего запроса. Если выполнение запроса заняло слишком много времени, попробуйте оптимизировать запрос. Если использовалась опция таймаута `cancel_after`, увеличьте её значение.

400170: UNDETERMINED Условно повторяемый | Короткая задержка

Состояние транзакции неизвестно. В результате выполнения запроса произошёл сбой, из-за которого невозможно определить состояние транзакции. На запросы, завершившиеся с таким статусом, распространяются гарантии целостности и атомарности транзакции. То есть либо все изменения зафиксированы, либо вся транзакция отменена.

Для идемпотентных транзакций можно повторить всю транзакцию с небольшой задержкой. В противном случае реакция зависит от логики приложения.

400180: UNSUPPORTED Неповторяемый

Запрос не поддерживается YDB, либо потому, что обработка таких запросов ещё не реализована в данной версии YDB, либо потому, что поддержка таких запросов не включена в конфигурации YDB.

Исправьте запрос или включите поддержку подобных запросов в конфигурации YDB.

400190: SESSION_BUSY Повторяемый | Короткая задержка

Сессия занята.

Пересоздайте сессию.

400200: EXTERNAL_ERROR Неповторяемый

Произошла ошибка во внешней системе, например, при обработке федеративного запроса или при импорте данных из внешнего источника.

Проанализируйте подробное сообщение об ошибке. При необходимости зарегистрируйте проблему на GitHub или обратитесь к технической поддержке YDB.

Смотрите также

Ошибки

---

[2] **Ошибки, которые не могут быть исправлены с помощью повтора** (non-retryable). Включают некорректно сформированные запросы, внутренние ошибки YDB, а также запросы, не соответствующие схеме данных. В такой ситуации нет необходимости повторять запрос, требуется дополнительное вмешательство разработчика.

[3] **Временные сбои** (retryable). Включают кратковременную потерю сетевого соединения, временную недоступность или перегруженность одной из подсистем YDB, а также неспособность YDB ответить на запрос в течение установленного времени ожидания. В случае возникновения таких ошибок повторный запрос через некоторый промежуток времени с высокой вероятностью будет выполнен успешно.

[4] **Короткая экспоненциальная задержка** – одна из стратегий задержек, применяемых в SDK при повторе запросов, завершившихся ошибкой.

Первоначальный интервал при такой задержке составляет несколько **миллисекунд**. Для каждой последующей попытки интервал увеличивается экспоненциально.

См. подробнее в статье Обработка временных сбоев (retryable errors).

[5] **Большая экспоненциальная задержка** – одна из стратегий задержек, применяемых в SDK при повторе запросов, завершившихся ошибкой.

Первоначальный интервал при такой задержке составляет несколько **секунд**. Для каждой последующей попытки интервал увеличивается по экспоненциальному закону.

См. подробнее в статье Обработка временных сбоев (retryable errors).

[6] **Немедленный повтор** – одна из стратегий задержек, применяемых в SDK при повторе запросов, завершившихся ошибкой.

Повторные попытки при такой стратегии совершаются немедленно.

См. подробнее в статье Обработка временных сбоев (retryable errors).

[7] **Ошибки, которые, предположительно, могут быть исправлены с помощью повтора после реакции клиентского приложения** (conditionally retryable). Возникают при идемпотентных операциях, таких как отсутствие ответа в течение отведённого времени или запрос аутентификации.

Статусы завершения gRPC

YDB предоставляет gRPC API, с помощью которого вы можете управлять ресурсами и данными БД. В следующей таблице описаны статусы завершения запросов gRPC:

Код	Статус	Возможность повтора	Стратегия задержек	Пересоздать сессию
0	OK	–	–	–
1	CANCELLED	условно повторяемый[8]	короткая[9]	да
2	UNKNOWN	неповторяемый[10]	–	да
3	INVALID_ARGUMENT	неповторяемый[10]	–	да
4	DEADLINE_EXCEEDED	условно повторяемый[8]	короткая[9]	да
5	NOT_FOUND	неповторяемый[10]	–	да
6	ALREADY_EXISTS	неповторяемый[10]	–	да
7	PERMISSION_DENIED	неповторяемый[10]	–	да
8	RESOURCE_EXHAUSTED	повторяемый[11]	большая[12]	нет
9	FAILED_PRECONDITION	неповторяемый[10]	–	да
10	ABORTED	повторяемый[11]	моментально[13]	да
11	OUT_OF_RANGE	неповторяемый[10]	–	нет
12	UNIMPLEMENTED	неповторяемый[10]	–	да
13	INTERNAL	условно повторяемый[8]	короткая[9]	да
14	UNAVAILABLE	условно повторяемый[8]	короткая[9]	да
15	DATA_LOSS	неповторяемый[10]	–	да
16	UNAUTHENTICATED	неповторяемый[10]	–	да

0: OK

Не является ошибкой. Возвращается при успешном выполнении.

1: CANCELLED Условно повторяемый | Короткая задержка

Операция была отменена, как правило, вызывающей стороной.

2: UNKNOWN Неповторяемый

Неизвестная ошибка. Например, эта ошибка может возникнуть, когда значение `Status`, полученное из другого адресного пространства, относится к неизвестному пространству ошибок. Также эта ошибка может быть преобразована из ошибок, вызванных API, которые не предоставляют достаточно информации об ошибке.

3: INVALID_ARGUMENT Неповторяемый

Клиент указал недопустимый аргумент. В отличие от `FAILED_PRECONDITION`, `INVALID_ARGUMENT` указывает на аргументы, которые являются проблемными независимо от состояния системы (например, неправильное имя файла).

4: DEADLINE_EXCEEDED Условно повторяемый | Короткая задержка

Запрос не был обработан за заданный клиентский таймаут или произошла иная сетевая проблема.

Проверьте корректность заданного таймаута, наличие сетевого доступа, правильность endpoint'а и других сетевых настроек. Также рекомендуется снизить интенсивность потока запросов и оптимизировать их.

5: NOT_FOUND Неповторяемый

Запрашиваемый схемный объект (например, таблица или папка) не найден.

6: ALREADY_EXISTS Неповторяемый

Схемный объект, который пытается создать клиент (например, таблица или папка), уже существует.

7: PERMISSION_DENIED Неповторяемый

У вызывающего нет разрешения на выполнение указанной операции.

8: RESOURCE_EXHAUSTED Повторяемый | Большая задержка

Недостаточно свободных ресурсов для обслуживания запроса.

Снизьте интенсивность потока запросов, проверьте клиентскую балансировку.

9: FAILED_PRECONDITION Неповторяемый

Запрос не может быть выполнен в текущем состоянии (например, вставка в таблицу с существующим ключом).

Исправьте состояние или запрос и повторите попытку.

10: ABORTED Повторяемый | Моментально

Операция не выполнена (например, из-за инвалидации локов, `TRANSACTION_LOCKS_INVALIDATE` в подробных сообщениях об ошибке).

Повторите всю транзакцию.

11: OUT_OF_RANGE Неповторяемый

Была предпринята попытка выполнить операцию за пределами допустимого диапазона. В отличие от `INVALID_ARGUMENT`, эта ошибка указывает на проблему, которая может быть исправлена при изменении состояния системы.

12: UNIMPLEMENTED Неповторяемый

Операция не реализована или не поддерживается (не активирована) в YDB.

13: INTERNAL Условно повторяемый | Короткая задержка

Внутренние ошибки. Это означает, что некоторые инварианты, ожидаемые базовой системой, были нарушены. Данный код ошибки предназначен для серьёзных ошибок.

14: UNAVAILABLE Условно повторяемый | Короткая задержка

Сервис в данный момент недоступен. Скорее всего, это временное состояние, которое можно обойти, повторив запрос после некоторой задержки. Обратите внимание, что повторение неидемпотентных операций не всегда безопасно.

15: DATA_LOSS Неповторяемый

Неустранимая потеря или повреждение данных.

16: UNAUTHENTICATED Неповторяемый

Запрос не содержит действительных учётных данных для аутентификации.

Повторите запрос с актуальными данными для аутентификации.

---

[8] **Ошибки, которые, предположительно, могут быть исправлены с помощью повтора после реакции клиентского приложения** (conditionally retryable). Возникают при идемпотентных операциях, таких как отсутствие ответа в течение отведённого времени или запрос аутентификации.

[9] **Короткая экспоненциальная задержка** – одна из стратегий задержек, применяемых в SDK при повторе запросов, завершившихся ошибкой.

Первоначальный интервал при такой задержке составляет несколько **миллисекунд**. Для каждой последующей попытки интервал увеличивается экспоненциально.

См. подробнее в статье Обработка временных сбоев (retryable errors).

[10] **Ошибки, которые не могут быть исправлены с помощью повтора** (non-retryable). Включают некорректно сформированные запросы, внутренние ошибки YDB, а также запросы, не соответствующие схеме данных. В такой ситуации нет необходимости повторять запрос, требуется дополнительное вмешательство разработчика.

[11] **Временные сбои** (retryable). Включают кратковременную потерю сетевого соединения, временную недоступность или перегруженность одной из подсистем YDB, а также неспособность YDB ответить на запрос в течение установленного времени ожидания. В случае возникновения таких ошибок повторный запрос через некоторый промежуток времени с высокой вероятностью будет выполнен успешно.

[12] **Большая экспоненциальная задержка** – одна из стратегий задержек, применяемых в SDK при повторе запросов, завершившихся ошибкой.

Первоначальный интервал при такой задержке составляет несколько **секунд**. Для каждой последующей попытки интервал увеличивается по экспоненциальному закону.

См. подробнее в статье Обработка временных сбоев (retryable errors).

[13] **Немедленный повтор** – одна из стратегий задержек, применяемых в SDK при повторе запросов, завершившихся ошибкой.

Повторные попытки при такой стратегии совершаются немедленно.

См. подробнее в статье Обработка временных сбоев (retryable errors).

Обзор gRPC API

YDB предоставляет gRPC API, с помощью которого вы можете управлять ресурсами и данными БД. Для описания методов и структур данных API используется Protocol Buffers (proto 3). Подробнее смотрите .proto-спецификации с комментариями. Также YDB использует специальные заголовки метаданных gRPC.

Доступны следующие сервисы:

- Health Check API.

Заголовки метаданных gRPC

YDB использует следующие gRPC metadata заголовки:

- gRPC заголовки, отправляемые клиентом в YDB :
  - `x-ydb-database` - база данных
  - `x-ydb-auth-ticket` - токен авторизации, полученный от провайдера авторизации
  - `x-ydb-sdk-build-info` - информация о YDB SDK
  - `x-ydb-trace-id` - идентификатор запроса, устанавливаемый пользователем. Если не задано пользователем - YDB SDK генерирует автоматически в формате UUID
  - `x-ydb-application-name` - опциональное имя приложения, устанавливаемое пользователем
  - `x-ydb-client-capabilities` - поддерживаемые клиентским SDK возможности (`session-balancer` и другие)
  - `x-ydb-client-pid` - идентификатор процесса клиентского приложения
  - `traceparent` - заголовок для передачи родительского идентификатора трассы OpenTelemetry (спецификация)
- gRPC заголовки, отправляемые клиенту YDB вместе с ответом на текущий запрос:
  - `x-ydb-server-hints` - уведомления YDB (`session-close` и другие)
  - `x-ydb-consumed-units` - потребленные ресурсов YDB на текущем запросе

Health Check API

YDB имеет встроенную систему самодиагностики, с помощью которой можно получить краткий отчет о состоянии базы данных и информацию об имеющихся проблемах.

Чтобы инициировать проверку, вызовите метод `SelfCheck` из пространства имен `NYdb::NMonitoring` в SDK. Также необходимо передать имя проверяемой БД стандартным способом.

C++

Пример кода приложения для создания клиента:

```
auto client = NYdb::NMonitoring::TMonitoringClient(driver);
```

Вызов метода `SelfCheck`:

```
auto settings =  NYdb::NMonitoring::TSelfCheckSettings();
settings.ReturnVerboseStatus(true);
auto result = client.SelfCheck(settings).GetValueSync();
```

Go

Функциональность на данный момент не поддерживается

Java

Функциональность на данный момент не поддерживается

Python

Функциональность на данный момент не поддерживается

C#

Функциональность на данный момент не поддерживается.

JavaScript

Функциональность на данный момент не поддерживается

Можно сделать клиент для мониторинга и вызывать методы проверки самостоятельно:

```
const monitoring = driver.createClient(MonitoringServiceDefinition);
await monitoring.selfCheck();
```

Rust

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#494

PHP

Функциональность на данный момент не поддерживается

Параметры вызова

`SelfCheck` возвращает информацию в форме списка проблем, каждая из которых может выглядеть так:

```
{
  "id": "RED-27c3-70fb",
  "status": "RED",
  "message": "Database has multiple issues",
  "location": {
    "database": {
      "name": "/slice"
    }
  },
  "reason": [
    "RED-27c3-4e47",
    "RED-27c3-53b5",
    "YELLOW-27c3-5321"
  ],
  "type": "DATABASE",
  "level": 1
}
```

Это короткое сообщение об одной из проблем. Все параметры вызова могут влиять на размер информации, которая содержится в ответе.

Полный список дополнительных параметров представлен ниже:

C++

```
struct TSelfCheckSettings : public TOperationRequestSettings<TSelfCheckSettings>{
    FLUENT_SETTING_OPTIONAL(bool, ReturnVerboseStatus);
    FLUENT_SETTING_OPTIONAL(EStatusFlag, MinimumStatus);
    FLUENT_SETTING_OPTIONAL(ui32, MaximumLevel);
};
```

Go

Функциональность на данный момент не поддерживается

Java

Функциональность на данный момент не поддерживается

Python

Функциональность на данный момент не поддерживается

C#

Функциональность на данный момент не поддерживается.

JavaScript

Функциональность на данный момент не поддерживается

Rust

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#494

PHP

Функциональность на данный момент не поддерживается

Поле	Тип	Описание
`ReturnVerboseStatus`	`bool`	Если задано, ответ также будет содержать сводку общего состояния базы данных в поле `database_status` (Пример). По умолчанию — `false`.
`MinimumStatus`	EStatusFlag	Каждая проблема содержит поле `status`. Если `minimum_status` определён, проблемы с менее серьёзным статусом будут отброшены. По умолчанию будут перечислены все проблемы.
`MaximumLevel`	`int32`	Каждая проблема содержит поле `level`. Если `maximum_level` определён, более серьёзные проблемы будут отброшены. По умолчанию будут перечислены все проблемы.

Структура ответа

Полную структуру ответа можно посмотреть в файле ydb_monitoring.proto в Git репозитории YDB.
В результате вызова этого метода будет возвращена следующая структура:

```
message SelfCheckResult {
    SelfCheck.Result self_check_result = 1;
    repeated IssueLog issue_log = 2;
    repeated DatabaseStatus database_status = 3;
    LocationNode location = 4;
}
```

Если обнаружены проблемы, поле `issue_log` будет содержать описания этих проблем со следующей структурой:

```
message IssueLog {
    string id = 1;
    StatusFlag.Status status = 2;
    string message = 3;
    Location location = 4;
    repeated string reason = 5;
    string type = 6;
    uint32 level = 7;
}
```

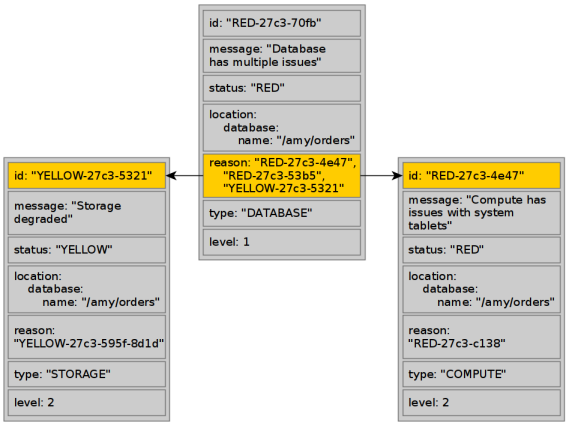
Описание полей в ответе

Поле	Описание
`self_check_result`	Перечисляемое поле, содержащее результат проверки базы данных.
`issue_log`	Набор элементов, каждый из которых описывает проблему в системе на определённом уровне.
`issue_log.id`	Уникальный идентификатор проблемы в этом ответе.
`issue_log.status`	Перечисляемое поле, содержащее статус проблемы.
`issue_log.message`	Текст, описывающий проблему.
`issue_log.location`	Местоположение проблемы. Это может быть физическое местоположение или контекст выполнения.
`issue_log.reason`	Набор элементов, каждый из которых описывает причину проблемы в системе на определённом уровне.
`issue_log.type`	Категория проблемы. Каждый тип находится на определённом уровне и связан с другими через жёсткую иерархию (как показано на изображении выше).
`issue_log.level`	Глубина вложенности проблемы.

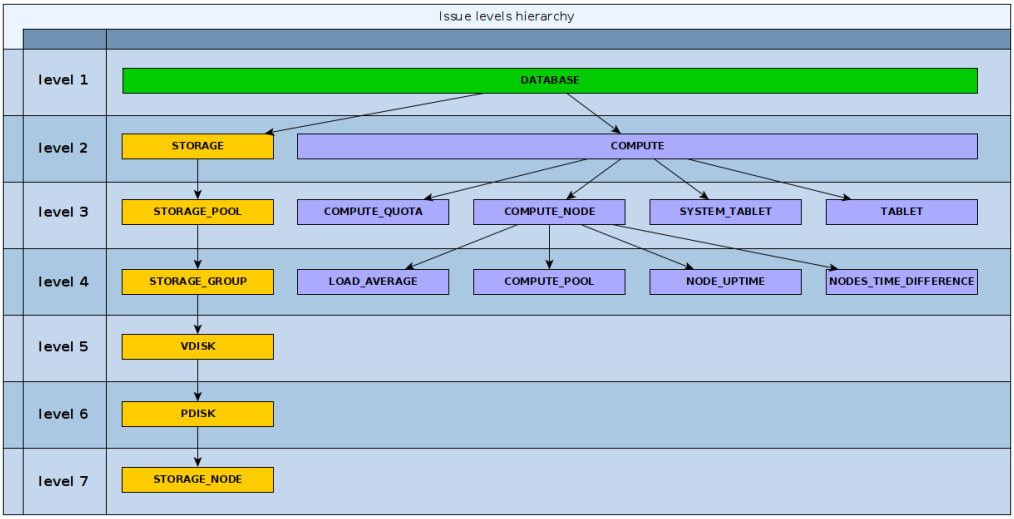
`database_status`	Если в настройках задан параметр `verbose`, то поле `database_status` будет заполнено. Оно предоставляет сводку общего состояния базы данных и используется для быстрой оценки состояния базы данных и выявления серьёзных проблем на высоком уровне. Пример. Полную структуру ответа можно посмотреть в файле ydb_monitoring.proto в Git репозитории YDB.
`location`	Содержит информацию о хосте, на котором был вызван сервис `HealthCheck`.

Иерархия проблем

Эти проблемы можно организовать в иерархию с помощью полей `id` и `reason`, что помогает визуализировать, как проблемы в отдельном модуле влияют на состояние системы в целом. Все проблемы организованы в иерархию, где верхние уровни могут зависеть от вложенных:



Каждая проблема имеет уровень вложенности `level` — чем выше `level`, тем глубже проблема находится в иерархии. Проблемы одного и того же типа (поле `type`) всегда имеют одинаковый `level`, и их можно представить в виде иерархии.



Результат проверки базы данных

Наиболее общий статус базы данных `self_check_result` может принимать следующие значения:

Значение	Описание
`GOOD`	Проблем не обнаружено.
`DEGRADED`	Обнаружена деградация одной из систем базы данных, но база данных всё ещё функционирует (например, допустимая потеря диска).
`MAINTENANCE_REQUIRED`	Обнаружена значительная деградация, есть риск потери доступности, требуется обслуживание.
`EMERGENCY`	Обнаружена серьёзная проблема в базе данных с полной или частичной потерей доступности.

Статус проблемы

Статус (серьёзность) текущей проблемы `issue_log.status`:

Значение	Описание
`GREY`	Обнаружена деградация одной из систем базы данных, но база данных всё ещё функционирует (например, допустимая потеря диска).
`GREEN`	Проблем не обнаружено.
`BLUE`	Временная небольшая деградация, не влияющая на доступность базы данных. Ожидается переход системы в `GREEN`.
`YELLOW`	Небольшая проблема, нет рисков для доступности. Рекомендуется продолжать мониторинг проблемы.
`ORANGE`	Серьёзная проблема, мы в шаге от потери доступности. Может потребоваться обслуживание.

`RED`	Компонент неисправен или недоступен.

Возможные проблемы

DATABASE

Database has multiple issues, Database has compute issues, Database has storage issues

Описание: Зависит от нижележащих слоёв `COMPUTE` и `STORAGE`. Это самый общий статус базы данных.

STORAGE

There are no storage pools

Описание: Нет информации о пулах хранения. Скорее всего, пулы хранения не настроены.

Storage degraded, Storage has no redundancy, Storage failed

Описание: Зависит от нижележащего слоя `STORAGE_POOLS`.

System tablet BSC didn't provide information

Описание: Информация о распределённом хранилище недоступна.

Storage usage over 75%, Storage usage over 85%, Storage usage over 90%

Описание: Необходимо увеличить дисковое пространство.

STORAGE_POOL

Pool degraded, Pool has no redundancy, Pool failed

Описание: Зависит от нижележащего слоя `STORAGE_GROUP`.

STORAGE_GROUP

Group has no vslots

Описание: Эта ошибка не ожидается. Внутренняя ошибка.

Group layout is incorrect

Описание: Группа хранилища была настроена некорректно.

Действия при срабатывании: В YDB UI перейти на страницу базы данных, выбрать вкладку `Storage`, по известному `id` группы проверить конфигурацию узлов и дисков.

Group degraded

Описание: В группе недоступно допустимое число дисков.
Логика работы: `HealthCheck` проверяет различные параметры (режим отказоустойчивости, количество отказавших дисков, статус дисков и т. д.) и в зависимости от этого устанавливает соответствующий статус группы.
Действия при срабатывании: В YDB YDB UI перейти на страницу базы данных, выбрать вкладку `Storage`, установить фильтры `Groups` и `Degraded`, по известному `id` группы проверить доступность нод и дисков на нодах.

Group has no redundancy

Описание: Группа хранения потеряла избыточность. Ещё один сбой в работе диска может привести к потере группы.
Логика работы: `HealthCheck` проверяет различные параметры (режим отказоустойчивости, количество отказавших дисков, статус дисков и т. д.) и в зависимости от этого устанавливает соответствующий статус группы.
Действия при срабатывании: В YDB YDB UI перейти на страницу базы данных, выбрать вкладку `Storage`, установить фильтры `Groups` и `Degraded`, по известному `id` группы проверить доступность нод и дисков на нодах.

Group failed

Описание: Группа хранения потеряла целостность и неработоспособна. Данные недоступны.
Логика работы: `HealthCheck` проверяет различные параметры (режим отказоустойчивости, количество отказавших дисков, статус дисков и т. д.) и в зависимости от этого устанавливает соответствующий статус группы.
Действия при срабатывании: В YDB YDB UI перейти на страницу базы данных, выбрать вкладку `Storage`, установить фильтры `Groups` и `Degraded`, по известному `id` группы проверить доступность нод и дисков на нодах.

VDISK

System tablet BSC didn't provide known status

Описание: Эта ошибка не ожидается. Внутренняя ошибка.

VDisk is not available

Описание: Виртуальный диск отсутствует.
Действия при срабатывании: В YDB YDB UI перейти на страницу базы данных, выбрать вкладку `Storage`, установить фильтры `Groups` и `Degraded`. По связанному issue `STORAGE_GROUP` можно узнать `id` группы. Навести на нужный vdisk — будет видно, на какой ноде проблема. Проверить доступность нод и дисков на нодах.

VDisk is being initialized

Описание: Инициализация виртуального диска в процессе.
Действия при срабатывании: В YDB UI перейти на страницу базы данных, выбрать вкладку `Storage`, установить фильтры `Groups` и `Degraded`. По связанному issue `STORAGE_GROUP` можно узнать `id` группы. Навести на нужный vdisk — будет видно, на какой ноде проблема. Проверить доступность нод и дисков на нодах.

Replication in progress

Описание: Диск в процессе репликации, но может принимать запросы.
Действия при срабатывании: В YDB UI перейти на страницу базы данных, выбрать вкладку `Storage`, установить фильтры `Groups` и `Degraded`. По связанному issue `STORAGE_GROUP` можно узнать `id` группы. Навести на нужный vdisk — будет видно, на какой ноде проблема. Проверить доступность нод и дисков на нодах.

VDisk have space issue

Описание: Зависит от нижележащего слоя `PDISK`.

PDISK

Unknown PDisk state

Описание: `HealthCheck` не может определить состояние PDisk. Внутренняя ошибка.

PDisk state is

Описание: Сообщает состояние физического диска.
Действия при срабатывании: В YDB UI перейти на страницу базы данных, выбрать вкладку `Storage`, установить фильтры `Nodes` и `Degraded`. По известным `id` ноды и pDisk проверить доступность нод и дисков на нодах.

Available size is less than 12%, Available size is less than 9%, Available size is less than 6%

Описание: На физическом диске заканчивается свободное место.
Действия при срабатывании: В YDB UI перейти на страницу базы данных, выбрать вкладку `Storage`, установить фильтры `Nodes` и `Out of Space`, по известным `id` ноды и PDisk просмотреть доступное место.

PDisk is not available

Описание: Физический диск отсутствует.
Действия при срабатывании: В YDB UI перейти на страницу базы данных, выбрать вкладку `Storage`, установить фильтры `Nodes` и `Degraded`, по известным `id` ноды и PDisk проверить доступность нод и дисков на нодах.

STORAGE_NODE

Storage node is not available

Описание: Стороджовая нода отсутствует. Эта информация помогает при диагностике верхнего слоя `PDISK`.

COMPUTE

There are no compute nodes

Описание: В базе данных отсутствуют ноды для запуска таблеток. Невозможно определить уровень `COMPUTE_NODE` ниже.

Compute has issues with system tablets

Описание: Зависит от нижележащего слоя `SYSTEM_TABLET`.

Some nodes are restarting too often

Описание: Зависит от нижележащего слоя `NODE_UPTIME`.

Compute is overloaded

Описание: Зависит от нижележащего слоя `COMPUTE_POOL`.

Compute quota usage

Описание: Зависит от нижележащего слоя `COMPUTE_QUOTA`.

Compute has issues with tablets

Описание: Зависит от нижележащего слоя `TABLET`.

COMPUTE_QUOTA

Paths quota usage is over than 90%, Paths quota usage is over than 99%, Paths quota exhausted, Shards quota usage is over than 90%, Shards quota usage is over than 99%, Shards quota exhausted

Описание: Квоты исчерпаны.
Действия при срабатывании: Проверить количество объектов (таблицы, топики) в базе данных, удалить лишние.

SYSTEM_TABLET

System tablet is unresponsive, System tablet response time over 1000ms, System tablet response time over 5000ms

Описание: Системная таблетка не отвечает или отвечает с задержкой.
Действия при срабатывании: В YDB UI на вкладке `Storage` установить фильтр `Nodes`. Проверить `Uptime` нод и их статус. Если `Uptime` небольшой, проверить по логам причины рестарта нод.

TABLET

Tablets are restarting too often

Описание: Таблетки слишком часто перезапускаются.
Действия при срабатывании: В YDB UI перейти на вкладку `Nodes`. Проверить `Uptime` нод и их статус. Если `Uptime` небольшой, необходимо проверить логи для определения причин частых рестартов нод.

Tablets are dead, Followers are dead

Описание: Таблетки не запущены (или не могут быть запущены).
Действия при срабатывании: В YDB UI перейти на вкладку `Nodes`. Проверить `Uptime` нод и их статус. Если `Uptime` небольшой, необходимо проверить логи для определения причин рестарта нод.

LOAD_AVERAGE

LoadAverage above 100%

Описание: Физический хост перегружен (Load) . Это указывает на то, что система работает на пределе, скорее всего, из-за большого количества процессов, ожидающих операций ввода-вывода.

Логика работы:
Информация о нагрузке:

- Источник: `/proc/loadavg`
- Информация о логических ядрах:
  - Основной источник: `/sys/fs/cgroup/cpu.max`
  - Дополнительный источник: `/sys/fs/cgroup/cpu/cpu.cfs_quota_us`, `/sys/fs/cgroup/cpu/cpu.cfs_period_us`
- Количество ядер вычисляется путём деления квоты на период (`quota / period`).

Действия при срабатывании: Проверить загруженность нод по CPU.

COMPUTE_POOL

Pool usage is over than 90%, Pool usage is over than 95%, Pool usage is over than 99%

Описание: Один из CPU-пулов перегружен.
Действия при срабатывании: Добавить ядра в конфигурацию акторной системы соответствующего CPU-пула.

NODE_UPTIME

The number of node restarts has increased

Описание: Количество рестартов ноды превысило порог. По умолчанию это 10 рестартов в час.
Действия при срабатывании: Проверить причины рестарта процесса по логам.

Node is restarting too often

Описание: Узлы слишком часто перезапускаются. По умолчанию это 30 рестартов в час.
Действия при срабатывании: Проверить причины рестарта процесса по логам.

NODES_TIME_DIFFERENCE

Node is ... ms behind peer [id], Node is ... ms ahead of peer [id]

Описание: Расхождение времени на узлах, что может приводить к проблемам с координацией распределённых транзакций. Проблема начинает проявляться при расхождении в 5 ms.
Действия при срабатывании: Проверить расхождение системного времени между нодами, перечисленными в алерте, и проверить работу процесса синхронизации времени.

Примеры ответа

Самый короткий ответ сервиса будет выглядеть следующим образом. Он возвращается, если с базой данных всё в порядке:

```
{
  "self_check_result": "GOOD"
}
```

Пример verbose

Ответ `GOOD` при использовании параметра `verbose`:

```
{
    "self_check_result": "GOOD",
    "database_status": [
        {
            "name": "/amy/db",
            "overall": "GREEN",
            "storage": {
                "overall": "GREEN",
                "pools": [
                    {
                        "id": "/amy/db:ssdencrypted",
                        "overall": "GREEN",
                        "groups": [
                            {
                                "id": "2181038132",
                                "overall": "GREEN",
                                "vdisks": [
                                    {
                                        "id": "9-1-1010",
                                        "overall": "GREEN",
                                        "pdisk": {
                                            "id": "9-1",
                                            "overall": "GREEN"
                                        }
                                    },
                                    {
                                        "id": "11-1004-1009",
                                        "overall": "GREEN",
                                        "pdisk": {
                                            "id": "11-1004",
                                            "overall": "GREEN"
                                        }
                                    },
                                    {
                                        "id": "10-1003-1011",
                                        "overall": "GREEN",
                                        "pdisk": {
                                            "id": "10-1003",
                                            "overall": "GREEN"
                                        }
```

```
                                },
                                {
                                    "id": "8-1005-1010",
                                    "overall": "GREEN",
                                    "pdisk": {
                                        "id": "8-1005",
                                        "overall": "GREEN"
                                    }
                                },
                                {
                                    "id": "7-1-1008",
                                    "overall": "GREEN",
                                    "pdisk": {
                                        "id": "7-1",
                                        "overall": "GREEN"
                                    }
                                },
                                {
                                    "id": "6-1-1007",
                                    "overall": "GREEN",
                                    "pdisk": {
                                        "id": "6-1",
                                        "overall": "GREEN"
                                    }
                                },
                                {
                                    "id": "4-1005-1010",
                                    "overall": "GREEN",
                                    "pdisk": {
                                        "id": "4-1005",
                                        "overall": "GREEN"
                                    }
                                },
                                {
                                    "id": "2-1003-1013",
                                    "overall": "GREEN",
                                    "pdisk": {
                                        "id": "2-1003",
                                        "overall": "GREEN"
                                    }
                                },
                                {
                                    "id": "1-1-1008",
                                    "overall": "GREEN",
                                    "pdisk": {
                                        "id": "1-1",
                                        "overall": "GREEN"
                                    }
                                }
                            ]
                        }
                    ]
                }
            ]
        },
        "compute": {
            "overall": "GREEN",
            "nodes": [
                {
                    "id": "50073",
                    "overall": "GREEN",
                    "pools": [
                        {
                            "overall": "GREEN",
                            "name": "System",
                            "usage": 0.000405479
                        },
                        {
                            "overall": "GREEN",
                            "name": "User",
                            "usage": 0.00265229
                        },
                        {
                            "overall": "GREEN",
                            "name": "Batch",
                            "usage": 0.000347933
                        },
                        {
                            "overall": "GREEN",
                            "name": "IO",
                            "usage": 0.000312022
                        },
                        {
                            "overall": "GREEN",
                            "name": "IC",
                            "usage": 0.000945925
                        }
                    ],
                    "load": {
                        "overall": "GREEN",
                        "load": 0.2,
```

```
                    "cores": 4
                }
            },
            {
                "id": "50074",
                "overall": "GREEN",
                "pools": [
                    {
                        "overall": "GREEN",
                        "name": "System",
                        "usage": 0.000619053
                    },
                    {
                        "overall": "GREEN",
                        "name": "User",
                        "usage": 0.00463859
                    },
                    {
                        "overall": "GREEN",
                        "name": "Batch",
                        "usage": 0.000596071
                    },
                    {
                        "overall": "GREEN",
                        "name": "IO",
                        "usage": 0.0006241
                    },
                    {
                        "overall": "GREEN",
                        "name": "IC",
                        "usage": 0.00218465
                    }
                ],
                "load": {
                    "overall": "GREEN",
                    "load": 0.08,
                    "cores": 4
                }
            },
            {
                "id": "50075",
                "overall": "GREEN",
                "pools": [
                    {
                        "overall": "GREEN",
                        "name": "System",
                        "usage": 0.000579126
                    },
                    {
                        "overall": "GREEN",
                        "name": "User",
                        "usage": 0.00344293
                    },
                    {
                        "overall": "GREEN",
                        "name": "Batch",
                        "usage": 0.000592347
                    },
                    {
                        "overall": "GREEN",
                        "name": "IO",
                        "usage": 0.000525747
                    },
                    {
                        "overall": "GREEN",
                        "name": "IC",
                        "usage": 0.00174265
                    }
                ],
                "load": {
                    "overall": "GREEN",
                    "load": 0.26,
                    "cores": 4
                }
            }
        ],
        "tablets": [
            {
                "overall": "GREEN",
                "type": "SchemeShard",
                "state": "GOOD",
                "count": 1
            },
            {
                "overall": "GREEN",
                "type": "SysViewProcessor",
                "state": "GOOD",
                "count": 1
            },
            {
                "overall": "GREEN",
```

```
                        "type": "Coordinator",
                        "state": "GOOD",
                        "count": 3
                    },
                    {
                        "overall": "GREEN",
                        "type": "Mediator",
                        "state": "GOOD",
                        "count": 3
                    },
                    {
                        "overall": "GREEN",
                        "type": "Hive",
                        "state": "GOOD",
                        "count": 1
                    }
                ]
            }
        }
    ]
}
```


Пример EMERGENCY

Ответ в случаи проблем может выглядеть так:


```
{
  "self_check_result": "EMERGENCY",
  "issue_log": [
    {
      "id": "RED-27c3-70fb",
      "status": "RED",
      "message": "Database has multiple issues",
      "location": {
        "database": {
          "name": "/slice"
        }
      },
      "reason": [
        "RED-27c3-4e47",
        "RED-27c3-53b5",
        "YELLOW-27c3-5321"
      ],
      "type": "DATABASE",
      "level": 1
    },
    {
      "id": "RED-27c3-4e47",
      "status": "RED",
      "message": "Compute has issues with system tablets",
      "location": {
        "database": {
          "name": "/slice"
        }
      },
      "reason": [
        "RED-27c3-c138-BSController"
      ],
      "type": "COMPUTE",
      "level": 2
    },
    {
      "id": "RED-27c3-c138-BSController",
      "status": "RED",
      "message": "System tablet is unresponsive",
      "location": {
        "compute": {
          "tablet": {
            "type": "BSController",
            "id": [
              "72057594037989391"
            ]
          }
        },
        "database": {
          "name": "/slice"
        }
      },
      "type": "SYSTEM_TABLET",
      "level": 3
    },
    {
      "id": "RED-27c3-53b5",
      "status": "RED",
      "message": "System tablet BSC didn't provide information",
      "location": {
        "database": {
          "name": "/slice"
        }
      },
      "type": "STORAGE",
```

```
      "level": 2
    },
    {
      "id": "YELLOW-27c3-5321",
      "status": "YELLOW",
      "message": "Storage degraded",
      "location": {
        "database": {
          "name": "/slice"
        }
      },
      "reason": [
        "YELLOW-27c3-595f-8d1d"
      ],
      "type": "STORAGE",
      "level": 2
    },
    {
      "id": "YELLOW-27c3-595f-8d1d",
      "status": "YELLOW",
      "message": "Pool degraded",
      "location": {
        "storage": {
          "pool": {
            "name": "static"
          }
        },
        "database": {
          "name": "/slice"
        }
      },
      "reason": [
        "YELLOW-27c3-ef3e-0"
      ],
      "type": "STORAGE_POOL",
      "level": 3
    },
    {
      "id": "RED-84d8-3-3-1",
      "status": "RED",
      "message": "PDisk is not available",
      "location": {
        "storage": {
          "node": {
            "id": 3,
            "host": "man0-0026.ydb-dev.nemax.nebiuscloud.net",
            "port": 19001
          },
          "pool": {
            "group": {
              "vdisk": {
                "pdisk": [
                  {
                    "id": "3-1",
                    "path": "/dev/disk/by-partlabel/NVMEKIKIMR01"
                  }
                ]
              }
            }
          }
        }
      },
      "type": "PDISK",
      "level": 6
    },
    {
      "id": "RED-27c3-4847-3-0-1-0-2-0",
      "status": "RED",
      "message": "VDisk is not available",
      "location": {
        "storage": {
          "node": {
            "id": 3,
            "host": "man0-0026.ydb-dev.nemax.nebiuscloud.net",
            "port": 19001
          },
          "pool": {
            "name": "static",
            "group": {
              "vdisk": {
                "id": [
                  "0-1-0-2-0"
                ]
              }
            }
          }
        }
      },
      "database": {
        "name": "/slice"
      }
    },
```

```
      "reason": [
        "RED-84d8-3-3-1"
      ],
      "type": "VDISK",
      "level": 5
    },
    {
      "id": "YELLOW-27c3-ef3e-0",
      "status": "YELLOW",
      "message": "Group degraded",
      "location": {
        "storage": {
          "pool": {
            "name": "static",
            "group": {
              "id": [
                "0"
              ]
            }
          }
        },
        "database": {
          "name": "/slice"
        }
      },
      "reason": [
        "RED-27c3-4847-3-0-1-0-2-0"
      ],
      "type": "STORAGE_GROUP",
      "level": 4
    }
  ],
  "location": {
    "id": 5,
    "host": "man0-0028.ydb-dev.nemax.nebiuscloud.net",
    "port": 19001
  }
}
```

Наблюдаемость в YDB SDK

В этом разделе описано подключение средств диагностики YDB SDK — логирования, метрик и распределённой трассировки. Рекомендуется подключать их заранее, до возникновения проблем, чтобы при расследовании инцидента видеть полную картину состояния системы до, во время и после сбоя.

Логирование:

- Включение логирования
- Экспорт логов в OpenTelemetry

Метрики:

- Метрики с OpenTelemetry
- Включение метрик в Prometheus

Трассировка:

- Трассировка с OpenTelemetry
- Включение трассировки в Jaeger

Серверная наблюдаемость YDB, не зависящая от SDK, описана в разделе Справка по наблюдаемости YDB.

Включение логирования

Ниже приведены примеры кода включения логирования в разных YDB SDK.

C++

Функциональность на данный момент не поддерживается.

Go

Native SDK

Есть несколько способов включить логи в приложении, использующем `ydb-go-sdk`:

Через переменную окружения `YDB_LOG_SEVERITY_LEVEL`

Данная переменная окружения включает встроенный в `ydb-go-sdk` логгер (синхронный, неблочный) с выводом в стандартный поток вывода.
Выставить переменную окружения можно так:

```
export YDB_LOG_SEVERITY_LEVEL=info
```

(доступные значения `trace`, `debug`, `info`, `warn`, `error`, `fatal`, `quiet`, по умолчанию `quiet`).

Подключить сторонний логгер `go.uber.org/zap`

```
package main

import (
  "context"
  "os"

  "go.uber.org/zap"

  ydbZap "github.com/ydb-platform/ydb-go-sdk-zap"
  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/trace"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  var log *zap.Logger // zap-logger with init out of this scope
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydbZap.WithTraces(
      log,
      trace.DetailsAll,
    ),
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
  ...
}
```

Подключить сторонний логгер `github.com/rs/zerolog`

```
package main

import (
  "context"
  "os"

  "github.com/rs/zerolog"

  ydbZerolog "github.com/ydb-platform/ydb-go-sdk-zerolog"
  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/trace"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  var log zerolog.Logger // zap-logger with init out of this scope
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydbZerolog.WithTraces(
      &log,
      trace.DetailsAll,
    ),
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
```

```
  ...
}
```

Подключить собственную имплементацию логгера `github.com/ydb-platform/ydb-go-sdk/v3/log.Logger`

```
package main

import (
  "context"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/log"
  "github.com/ydb-platform/ydb-go-sdk/v3/trace"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  var logger log.Logger // logger implementation with init out of this scope
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithLogger(
      logger,
      trace.DetailsAll,
    ),
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
  ...
}
```

Реализовать собственный пакет логирования

Реализовать собственный пакет логирования можно на основе событий драйвера в пакете трассировки `github.com/ydb-platform/ydb-go-sdk/v3/trace`. Пакет трассировки `github.com/ydb-platform/ydb-go-sdk/v3/trace` содержит описание всех протоколируемых событий драйвера.

Реализовать получение информации о серверных ошибках `IterateByIssues`

При работе с YDB через Go SDK вы можете не только включать логирование запросов и ответов, но и программно получать детализированные сведения о серверных ошибках (issues) — дополнительных сообщениях, которые YDB возвращает в составе ответа при ошибках выполнения операций. Для итерации по списку issues, содержащихся в ответе от сервера, используйте метод IterateByIssues.

database/sql

Есть несколько способов включить логи в приложении, использующем `ydb-go-sdk`:

Через переменную окружения `YDB_LOG_SEVERITY_LEVEL`

Данная переменная окружения включает встроенный в `ydb-go-sdk` логгер (синхронный, неблочный) с выводом в стандартный поток вывода.
Выставить переменную окружения можно так:

```
export YDB_LOG_SEVERITY_LEVEL=info
```

(доступные значения `trace`, `debug`, `info`, `warn`, `error`, `fatal`, `quiet`, по умолчанию `quiet`).

Подключить сторонний логгер `go.uber.org/zap`

```
package main

import (
  "context"
  "database/sql"
  "os"

  "go.uber.org/zap"

  ydbZap "github.com/ydb-platform/ydb-go-sdk-zap"
  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/trace"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  var log *zap.Logger // zap-logger with init out of this scope
  nativeDriver, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
```

```
    ydbZap.WithTraces(
      log,
      trace.DetailsAll,
    ),
  )
  if err != nil {
    panic(err)
  }
  defer nativeDriver.Close(ctx)

  connector, err := ydb.Connector(nativeDriver)
  if err != nil {
    panic(err)
  }
  defer connector.Close()

  db := sql.OpenDB(connector)
  defer db.Close()
  ...
}
```

Подключить сторонний логгер `github.com/rs/zerolog`

```
package main

import (
  "context"
  "database/sql"
  "os"

  "github.com/rs/zerolog"

  ydbZerolog "github.com/ydb-platform/ydb-go-sdk-zerolog"
  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/trace"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  var log zerolog.Logger // zap-logger with init out of this scope
  nativeDriver, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydbZerolog.WithTraces(
      &log,
      trace.DetailsAll,
    ),
  )
  if err != nil {
    panic(err)
  }
  defer nativeDriver.Close(ctx)

  connector, err := ydb.Connector(nativeDriver)
  if err != nil {
    panic(err)
  }
  defer connector.Close()

  db := sql.OpenDB(connector)
  defer db.Close()
  ...
}
```

Подключить собственную имплементацию логгера `github.com/ydb-platform/ydb-go-sdk/v3/log.Logger`

```
package main

import (
  "context"
  "database/sql"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/log"
  "github.com/ydb-platform/ydb-go-sdk/v3/trace"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  var logger log.Logger // logger implementation with init out of this scope
  nativeDriver, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithLogger(
      logger,
      trace.DetailsAll,
```

```
    ),
  )
  if err != nil {
    panic(err)
  }
  defer nativeDriver.Close(ctx)

  connector, err := ydb.Connector(nativeDriver)
  if err != nil {
    panic(err)
  }
  defer connector.Close()

  db := sql.OpenDB(connector)
  defer db.Close()
  ...
}
```

Реализовать собственный пакет логирования

> Реализовать собственный пакет логирования можно на основе событий драйвера в пакете трассировки `github.com/ydb-platform/ydb-go-sdk/v3/trace`. Пакет трассировки `github.com/ydb-platform/ydb-go-sdk/v3/trace` содержит описание всех протоколируемых событий драйвера.

Java

Native SDK

YDB Java SDK использует SLF4J как фасад логирования: SDK пишет сообщения через API SLF4J, а конкретный вывод (log4j2, logback и т.д.) подключается зависимостями приложения. Конфигурацию log4j2 разместите в `src/main/resources/log4j2.xml`.

Зависимости Maven (Native SDK + log4j2):

```
<dependencies>
    <dependency>
        <groupId>tech.ydb</groupId>
        <artifactId>ydb-sdk-core</artifactId>
        <version><!-- актуальная версия --></version>
    </dependency>
    <dependency>
        <groupId>tech.ydb</groupId>
        <artifactId>ydb-sdk-query</artifactId>
        <version><!-- актуальная версия --></version>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-slf4j2-impl</artifactId>
        <version>2.24.3</version>
    </dependency>
</dependencies>
```

В YDB Java SDK доступны следующие логгеры:

- `tech.ydb.core.grpc` — gRPC-транспорт (`info` по умолчанию, `debug` — все RPC)
- `tech.ydb.table.impl` — внутреннее состояние драйвера, в том числе пул сессий
- `tech.ydb.table.SessionRetryContext` — повторы, длительность попыток
- `tech.ydb.table.Session` — текст запроса, статус и время выполнения

Пример `src/main/resources/log4j2.xml`:

```
<Configuration status="WARN">
  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
    </Console>
  </Appenders>

  <Loggers>
    <Logger name="io.netty" level="warn" additivity="false">
      <AppenderRef ref="Console"/>
    </Logger>
    <Logger name="io.grpc.netty" level="warn" additivity="false">
      <AppenderRef ref="Console"/>
    </Logger>
    <Logger name="tech.ydb.core.grpc" level="info" additivity="false">
      <AppenderRef ref="Console"/>
    </Logger>
    <Logger name="tech.ydb.table.impl" level="info" additivity="false">
      <AppenderRef ref="Console"/>
    </Logger>
    <Logger name="tech.ydb.table.SessionRetryContext" level="debug" additivity="false">
      <AppenderRef ref="Console"/>
    </Logger>
    <Logger name="tech.ydb.table.Session" level="debug" additivity="false">
      <AppenderRef ref="Console"/>
    </Logger>
```

```
    <Root level="debug">
      <AppenderRef ref="Console"/>
    </Root>
  </Loggers>
</Configuration>
```

Запускаемый пример — подключение и `SELECT 1` при включённом логировании:

```
import tech.ydb.common.transaction.TxMode;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;

public class DebugLogsNativeExample {

    public static void main(String[] args) {
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString).build();
             QueryClient queryClient = QueryClient.newClient(transport).build()) {

            SessionRetryContext retryCtx = SessionRetryContext.create(queryClient).build();

            QueryReader reader = retryCtx.supplyResult(session -> QueryReader.readFrom(
                    session.createQuery("SELECT 1 AS value", TxMode.NONE, Params.empty())
            )).join().getValue();

            ResultSetReader rs = reader.getResultSet(0);
            if (rs.next()) {
                System.out.println("SELECT 1 = " + rs.getColumn("value").getInt32());
            }
        }
    }
}
```

JDBC

JDBC-драйвер использует тот же стек SLF4J; настройте логгеры `tech.ydb.*` так же, как в нативном SDK. Конфигурацию log4j2 разместите в `src/main/resources/log4j2.xml` (см. XML выше). Зависимости Maven:

```
<dependencies>
    <dependency>
        <groupId>tech.ydb.jdbc</groupId>
        <artifactId>ydb-jdbc-driver</artifactId>
        <version><!-- актуальная версия --></version>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-slf4j2-impl</artifactId>
        <version>2.24.3</version>
    </dependency>
</dependencies>
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DebugLogsJdbcExample {

    public static void main(String[] args) throws SQLException {
        String url = System.getenv().getOrDefault(
                "YDB_JDBC_URL", "jdbc:ydb:grpc://localhost:2136/local");

        try (Connection connection = DriverManager.getConnection(url);
             Statement statement = connection.createStatement();
             ResultSet rs = statement.executeQuery("SELECT 1 AS value")) {

            rs.next();
            System.out.println("SELECT 1 = " + rs.getInt("value"));
        }
    }
}
```

Те же дебаг-логи доступны во всех остальных фреймворках вокруг JDBC (Spring Boot, ORM, пулы соединений и т.д.): они ходят в YDB через этот драйвер, достаточно подключить ту же конфигурацию `slf4j` / log4j2 / logback в приложении.

Python

Python SDK использует стандартную библиотеку для логирования - `logging`. Для включения определенного режима логирования:

```
import logging

logging.getLogger('ydb').setLevel(logging.DEBUG)
```

JavaScript

Для логирования событий внутри sdk используется библиотека debug.
Для включения логов необходимо задать переменную окружения `DEBUG` со значением фильтра по событиям sdk - `DEBUG=ydbjs:*`.

Rust

Внутри крейта `ydb` сообщения идут через стандартную для Rust экосистемы библиотеку `tracing` (это имя крейта; сюда же относятся обычные текстовые логи уровня debug/trace, не только «распределённая трассировка»). Чтобы видеть вывод в консоль, до создания клиента подключите подписчика, например `tracing_subscriber::fmt` с нужным уровнем (`TRACE` для максимальной детализации). Пример: `basic-logs.rs`.

```
tracing_subscriber::fmt()
    .with_max_level(tracing::Level::TRACE)
    .init();

let client = ydb::ClientBuilder::new_from_connection_string("grpc://localhost:2136?database=local")?
    .client()?;
```

C#

В YDB C# SDK логирование подключается через стандартный интерфейс `ILoggerFactory` из `Microsoft.Extensions.Logging`. Можно передать любую реализацию — консольный логгер, Serilog, NLog и другие:

```
using Microsoft.Extensions.Logging;
using Ydb.Sdk.Ado;

var loggerFactory = LoggerFactory.Create(builder =>
    builder.AddConsole().SetMinimumLevel(LogLevel.Debug));

var ydbBuilder = new YdbConnectionStringBuilder
{
    Host = "localhost",
    Port = 2136,
    Database = "/local",
    LoggerFactory = loggerFactory
};

await using var dataSource = new YdbDataSource(ydbBuilder);
await using var connection = await dataSource.OpenConnectionAsync();
```

Так как `LoggerFactory` принимает стандартный `ILoggerFactory`, подключить Serilog или NLog можно без дополнительных адаптеров:

```
// Serilog
var loggerFactory = new SerilogLoggerFactory(Log.Logger);

// NLog
var loggerFactory = LoggerFactory.Create(builder => builder.AddNLog());
```

PHP

В YDB PHP SDK для логирования вам нужно использовать класс, который реализует `\Psr\Log\LoggerInterface`.
В YDB-PHP-SDK встроены логгеры в пространстве имен `YdbPlatform\Ydb\Logger`:

- `NullLogger` - по умолчанию, который ничего не выводит
- `SimpleStdLogger($level)` - логгер, который выводит логи в stderr.

Пример использования:

```
$config = [
  'logger' => new \YdbPlatform\Ydb\Logger\SimpleStdLogger(\YdbPlatform\Ydb\Logger\SimpleStdLogger::INFO)
]
$ydb = new \YdbPlatform\Ydb\Ydb($config);
```

Экспорт логов в OpenTelemetry

Каждый YDB SDK пишет свои внутренние логи (инициализация драйвера, пул сессий, выполнение запросов, повторные попытки и т.д.) через стандартное средство логирования своего языка. Вместо вывода логов только в консоль (см. Включить логирование) их можно перенаправить в OpenTelemetry Logs SDK и экспортировать по стандартному протоколу OTLP в коллектор. Коллектор затем пересылает записи в выбранный бэкенд для хранения и просмотра логов.

Принцип одинаков во всех SDK:

1. Создать OpenTelemetry `LoggerProvider` с OTLP-экспортёром логов и атрибутом ресурса `service.name`.
2. Перенаправить логгер SDK в этот провайдер через адаптер (log appender / bridge), превращающий каждую запись лога SDK в OTel log record. Состав адаптеров и статус поддержки логов в разных языках см. в документации OpenTelemetry по логам.
3. Запустить рабочую нагрузку — все внутренние логи SDK теперь отправляются в коллектор как OTLP log records.

> **Примечание**
>
> ### Принципы
>
> Способ сбора логов зависит от языка и инфраструктуры — единого формата нет:
>
> - **Программные мосты (log appender / bridge).** Приложение само отправляет логи в коллектор по OTLP прямо из процесса (workflow «direct-to-Collector»). Именно этот подход показан в примерах ниже.
> - **Агенты-коллекторы.** Приложение пишет логи в файл или `stdout`, а отдельный агент (например, filelog receiver в OpenTelemetry Collector) читает, парсит и пересылает их в бэкенд — без изменения кода приложения.

Подключение к SDK

C++

YDB C++ SDK пишет логи через `TLogBackend` из `util`. Реализуйте бэкенд, который превращает каждую запись в OTel log record, и передайте его в драйвер через `TDriverConfig::SetLog`. Сборку OpenTelemetry C++ SDK см. в Getting Started.

```cpp
#include <ydb-cpp-sdk/client/driver/driver.h>
#include <opentelemetry/exporters/otlp/otlp_grpc_log_record_exporter_factory.h>
#include <opentelemetry/exporters/otlp/otlp_grpc_log_record_exporter_options.h>
#include <opentelemetry/logs/provider.h>
#include <opentelemetry/sdk/logs/batch_log_record_processor_factory.h>
#include <opentelemetry/sdk/logs/logger_provider_factory.h>
#include <opentelemetry/sdk/resource/resource.h>
#include <library/cpp/logger/backend.h>
#include <library/cpp/logger/record.h>
#include <memory>

namespace otlp     = opentelemetry::exporter::otlp;
namespace logs_sdk = opentelemetry::sdk::logs;
namespace logs_api = opentelemetry::logs;
namespace resource = opentelemetry::sdk::resource;

namespace {

class TOtelLogBackend final : public TLogBackend {
public:
  explicit TOtelLogBackend(opentelemetry::nostd::shared_ptr<logs_api::Logger> logger)
      : Logger_(std::move(logger)) {}

  void WriteData(const TLogRecord& rec) override {
      Logger_->EmitLogRecord(MapSeverity(rec.Priority),
                             opentelemetry::nostd::string_view(rec.Data, rec.Len));
  }

  void ReopenLog() override {}

private:
  static logs_api::Severity MapSeverity(ELogPriority p) {
      switch (p) {
          case TLOG_EMERG:
          case TLOG_ALERT:
          case TLOG_CRIT:    return logs_api::Severity::kFatal;
          case TLOG_ERR:     return logs_api::Severity::kError;
          case TLOG_WARNING: return logs_api::Severity::kWarn;
          case TLOG_NOTICE:
          case TLOG_INFO:    return logs_api::Severity::kInfo;
          case TLOG_DEBUG:   return logs_api::Severity::kDebug;
          default:           return logs_api::Severity::kTrace;
      }
  }

  opentelemetry::nostd::shared_ptr<logs_api::Logger> Logger_;
};
} // namespace

int main() {
  // 1. Настраиваем провайдер логов OTel с OTLP-экспортёром
  otlp::OtlpGrpcLogRecordExporterOptions exporterOpts;
  exporterOpts.endpoint = "localhost:4317";
  auto exporter  = otlp::OtlpGrpcLogRecordExporterFactory::Create(exporterOpts);
  auto processor = logs_sdk::BatchLogRecordProcessorFactory::Create(std::move(exporter));
```

```
  auto res       = resource::Resource::Create({{"service.name", "ydb-cpp-sdk-otel-logs-sample"}});

  std::shared_ptr<logs_api::LoggerProvider> provider(
      logs_sdk::LoggerProviderFactory::Create(std::move(processor), res));
  logs_api::Provider::SetLoggerProvider(provider);

  auto logger = provider->GetLogger("ydb-cpp-sdk");

  // 2. Передаём мост в драйвер YDB
  auto config = NYdb::TDriverConfig()
      .SetEndpoint("localhost:2136")
      .SetDatabase("/local")
      .SetLog(std::make_unique<TOtelLogBackend>(logger));

  NYdb::TDriver driver(config);
  // ... используйте driver ...
  driver.Stop(true);
}
```

Go

Для YDB Go SDK есть готовый адаптер ydb-go-sdk-otel, который превращает события SDK в сигналы OpenTelemetry: трассы (`WithTracer`), метрики (`WithMetrics`) и логи (`WithLogger`). Адаптер не настраивает экспортёры сам — вы создаёте `LoggerProvider` с OTLP-экспортёром логов, получаете из него логгер и передаёте его в опцию `ydbOtel.WithLogger` при вызове `ydb.Open` . Каждая внутренняя запись лога SDK (инициализация драйвера, пул сессий, выполнение запросов, ретраи и т.д.) при этом отправляется в коллектор как OTLP log record.

```
go get github.com/ydb-platform/ydb-go-sdk-otel
go get go.opentelemetry.io/otel/sdk/log
go get go.opentelemetry.io/otel/exporters/otlp/otlplog/otlploggrpc
```

```
package main

import (
    "context"
    "os"

    "github.com/ydb-platform/ydb-go-sdk/v3"
    "github.com/ydb-platform/ydb-go-sdk/v3/trace"

    "go.opentelemetry.io/otel/attribute"
    "go.opentelemetry.io/otel/exporters/otlp/otlplog/otlploggrpc"
    sdklog "go.opentelemetry.io/otel/sdk/log"
    "go.opentelemetry.io/otel/sdk/resource"

    ydbOtel "github.com/ydb-platform/ydb-go-sdk-otel"
)

func main() {
    ctx := context.Background()

    // 1. Настраиваем провайдер логов OTel с OTLP-экспортёром.
    exporter, err := otlploggrpc.New(ctx,
        otlploggrpc.WithEndpoint("localhost:4317"),
        otlploggrpc.WithInsecure(),
    )
    if err != nil {
        panic(err)
    }
    res, _ := resource.Merge(resource.Default(), resource.NewSchemaless(
        attribute.String("service.name", "ydb-go-sdk-otel-logs-sample"),
    ))
    lp := sdklog.NewLoggerProvider(
        sdklog.WithResource(res),
        sdklog.WithProcessor(sdklog.NewBatchProcessor(exporter)),
    )
    defer lp.Shutdown(ctx)

    // 2. Открываем драйвер YDB с адаптером ydb-go-sdk-otel.
    // WithLogger пересылает события логов SDK в OTel log records.
    logger := lp.Logger("ydb-go-sdk")
    db, err := ydb.Open(ctx,
        os.Getenv("YDB_CONNECTION_STRING"),
        ydbOtel.WithLogger(logger, ydbOtel.WithDetailer(trace.DetailsAll)),
    )
    if err != nil {
        panic(err)
    }
    defer db.Close(ctx)
    // ... используйте db ...
}
```

Java

YDB Java SDK пишет логи через `slf4j` . Если в качестве реализации `slf4j` используется logback, подключите готовый аппендер `opentelemetry-logback-appender-1.0` , который пересылает каждое событие в OpenTelemetry Logs SDK:

```
<dependency>
    <groupId>io.opentelemetry</groupId>
```

```
    <artifactId>opentelemetry-sdk</artifactId>
    <version>${otel.version}</version>
</dependency>
<dependency>
    <groupId>io.opentelemetry</groupId>
    <artifactId>opentelemetry-exporter-otlp</artifactId>
    <version>${otel.version}</version>
</dependency>
<dependency>
    <groupId>io.opentelemetry.instrumentation</groupId>
    <artifactId>opentelemetry-logback-appender-1.0</artifactId>
    <version>${otel.instrumentation.version}</version>
</dependency>
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>${logback.version}</version>
</dependency>
```

Подключите аппендер в `logback.xml`:

```
<configuration>
    <appender name="OTEL" class="io.opentelemetry.instrumentation.logback.appender.v1_0.OpenTelemetryAppende
r">
    </appender>

    <root level="INFO">
        <appender-ref ref="OTEL"/>
    </root>
</configuration>
```

Соберите экземпляр OpenTelemetry SDK с OTLP-экспортёром логов и установите его в аппендер вызовом `OpenTelemetryAppender.install(...)` перед запуском рабочей нагрузки:

```
import java.util.concurrent.TimeUnit;

import io.opentelemetry.api.OpenTelemetry;
import io.opentelemetry.api.common.Attributes;
import io.opentelemetry.exporter.otlp.logs.OtlpGrpcLogRecordExporter;
import io.opentelemetry.instrumentation.logback.appender.v1_0.OpenTelemetryAppender;
import io.opentelemetry.sdk.OpenTelemetrySdk;
import io.opentelemetry.sdk.logs.SdkLoggerProvider;
import io.opentelemetry.sdk.logs.export.BatchLogRecordProcessor;
import io.opentelemetry.sdk.resources.Resource;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;

String serviceName = "ydb-java-sdk-otel-logs-sample";
Resource resource = Resource.getDefault().merge(
        Resource.create(Attributes.builder().put("service.name", serviceName).build()));

SdkLoggerProvider loggerProvider = SdkLoggerProvider.builder()
        .setResource(resource)
        .addLogRecordProcessor(BatchLogRecordProcessor.builder(
                OtlpGrpcLogRecordExporter.builder().setEndpoint("http://localhost:4317").build()
        ).build())
        .build();

OpenTelemetry openTelemetry = OpenTelemetrySdk.builder()
        .setLoggerProvider(loggerProvider)
        .build();

// Передаём OpenTelemetry SDK в аппендер, объявленный в logback.xml.
OpenTelemetryAppender.install(openTelemetry);

try (GrpcTransport transport = GrpcTransport.forConnectionString("grpc://localhost:2136/local").build();
     QueryClient queryClient = QueryClient.newClient(transport).build()) {
    // ... используйте queryClient ...
} finally {
    loggerProvider.forceFlush().join(10, TimeUnit.SECONDS);
    loggerProvider.shutdown().join(10, TimeUnit.SECONDS);
}
```

Python

YDB Python SDK пишет логи через стандартный модуль `logging` (логгеры с именами `ydb.*`). Используйте встроенный в OpenTelemetry `LoggingHandler`, чтобы перенаправить эти записи в `LoggerProvider` и экспортировать по OTLP:

```
pip install opentelemetry-sdk opentelemetry-exporter-otlp-proto-grpc
```

```
import logging

import ydb
from opentelemetry._logs import set_logger_provider
from opentelemetry.exporter.otlp.proto.grpc._log_exporter import OTLPLogExporter
from opentelemetry.sdk._logs import LoggerProvider, LoggingHandler
from opentelemetry.sdk._logs.export import BatchLogRecordProcessor
```

```
from opentelemetry.sdk.resources import Resource

resource = Resource(attributes={"service.name": "ydb-otel-logs-example"})
logger_provider = LoggerProvider(resource=resource)
logger_provider.add_log_record_processor(
    BatchLogRecordProcessor(OTLPLogExporter(endpoint="http://localhost:4317"))
)
set_logger_provider(logger_provider)

# Мост stdlib logging -> OpenTelemetry. Привязка обработчика к корневому логгеру
# перехватывает всё, что SDK пишет через logging.getLogger("ydb...").
otel_handler = LoggingHandler(level=logging.NOTSET, logger_provider=logger_provider)
logging.basicConfig(level=logging.INFO, handlers=[otel_handler])
logging.getLogger("ydb").setLevel(logging.INFO)

with ydb.Driver(endpoint="grpc://localhost:2136", database="/local") as driver:
    driver.wait(timeout=5)
    with ydb.QuerySessionPool(driver) as pool:
        pool.execute_with_retries("SELECT 1")

logger_provider.shutdown()
```

C#

YDB C# SDK пишет логи через переданный ему `ILoggerFactory`. Создайте фабрику на основе провайдера логирования OpenTelemetry с OTLP-экспортёром и передайте её в источник данных:

```
dotnet add package OpenTelemetry
dotnet add package OpenTelemetry.Exporter.OpenTelemetryProtocol
```

```
using Microsoft.Extensions.Logging;
using OpenTelemetry.Logs;
using OpenTelemetry.Resources;
using Ydb.Sdk.Ado;

var resourceBuilder = ResourceBuilder.CreateDefault()
    .AddService("ydb-sdk-otel-logs-sample");

using var loggerFactory = LoggerFactory.Create(builder =>
{
    builder.AddOpenTelemetry(options =>
    {
        options.SetResourceBuilder(resourceBuilder);
        options.AddOtlpExporter(o => o.Endpoint = new Uri("http://localhost:4317"));
    });
});

// Передаём фабрику в SDK: каждый внутренний лог (инициализация драйвера, пул сессий,
// выполнение запросов, ретраи, ...) экспортируется в коллектор как OTLP log record.
await using var dataSource = new YdbDataSource(
    new YdbConnectionStringBuilder("Host=localhost;Port=2136;Database=/local")
    {
        LoggerFactory = loggerFactory
    });

await using var connection = await dataSource.OpenConnectionAsync();
await new YdbCommand("SELECT 1", connection).ExecuteNonQueryAsync();
```

JavaScript

Функциональность на данный момент не поддерживается.

Rust

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#268

PHP

Функциональность на данный момент не поддерживается.

Метрики с OpenTelemetry

YDB SDK инструментируют операции Query Service метриками OpenTelemetry, позволяя наблюдать состояние клиента — длительность и количество операций, состояние пула сессий — от кода приложения до gRPC-вызовов к YDB. Метрики экспортируются по стандартному протоколу OTLP и совместимы с Prometheus, Grafana, VictoriaMetrics и любым другим бэкендом, поддерживающим OpenTelemetry.

Список метрик

Метрики операций

Имя	Тип	Единица	Описание
`ydb.client.operation.duration`	Histogram	`s`	Длительность одной попытки клиентской операции (`ExecuteQuery`, `Commit`, `Rollback`, `CreateSession`).
`ydb.client.operation.failed`	Counter	`{operation}`	Количество неуспешных клиентских операций.

Метрики пула сессий

Имя	Тип	Единица	Описание
`ydb.query.session.create_time`	Histogram	`s`	Длительность создания новой сессии.
`ydb.query.session.closed`	Counter	`{session}`	Монотонный счётчик закрытых сессий, разделённый по причинам закрытия, с момента создания пула.
`ydb.query.session.pending_requests`	Counter	`{request}`	Монотонный счётчик запросов на получение сессии, попавших в очередь ожидания, с момента создания пула.
`ydb.query.session.timeouts`	Counter	`{timeout}`	Монотонный счётчик таймаутов при ожидании свободной сессии, с момента создания пула.
`ydb.query.session.count`	Gauge	`{session}`	Текущее количество сессий в пуле, разделённое по состояниям.
`ydb.query.session.min`	Gauge	`{session}`	Настроенный минимальный размер пула сессий.
`ydb.query.session.max`	Gauge	`{session}`	Настроенный максимальный размер пула сессий.

Метрики повторных попыток

Имя	Тип	Единица	Описание
`ydb.client.retry.duration`	Histogram	`s`	Полная видимая клиенту длительность логической операции, выполненной через политику повторов, включая все попытки и задержки backoff.
`ydb.client.retry.attempts`	Histogram	`{attempt}`	Распределение числа попыток на одну логическую операцию. Значение `1` означает успех с первой попытки.

Атрибуты

Имя	Применяется к	Значение
`database`	`ydb.client.operation.duration`, `ydb.client.operation.failed`	Имя базы данных YDB.
`endpoint`	`ydb.client.operation.duration`, `ydb.client.operation.failed`	Discovery-endpoint в формате `host:port`.
`operation.name`	`ydb.client.operation.duration`, `ydb.client.operation.failed`, `ydb.client.retry.duration`, `ydb.client.retry.attempts`	Имя клиентской операции: `ExecuteQuery`, `Commit`, `Rollback`, `CreateSession`.
`status_code`	`ydb.client.operation.failed`	Код статуса YDB (например, `BAD_REQUEST`, `SCHEME_ERROR`).
`ydb.query.session.pool.name`	Все метрики `ydb.query.session.*`	Имя пула сессий. По умолчанию формируется как `<endpoint>/<database>`; настраивается через API конкретного SDK.
`ydb.query.session.state`	`ydb.query.session.count`	Состояние сессии: `idle` или `used`.
`reason`	`ydb.query.session.closed`	Причина закрытия сессии. Допустимые значения перечислены ниже.

Допустимые значения атрибута `reason`:

- `pool_idle_timeout` — пул удалил неактивную сессию по таймауту;
- `pool_graceful_shutdown` — пул штатно завершил работу и закрыл свои сессии;
- `client_timeout` — поток результатов запроса превысил клиентский транспортный таймаут;
- `client_cancelled` — клиент закрыл незавершённый поток результатов запроса;
- `attach_closed` — сервер закрыл работающий поток `AttachSession`;

- `transport_error` — сессия выведена из работы из-за транспортной ошибки, в том числе ошибки `Unavailable` при выполнении запроса или ошибки потока `AttachSession`;
- `node_shutdown` — сервер сообщил клиенту о штатном завершении работы узла;
- `session_shutdown` — сервер сообщил клиенту о штатном завершении сессии;
- `bad_session` — сервер вернул статус `BadSession` или `SessionExpired`;
- `session_busy` — сервер вернул статус `SessionBusy`.

Поддерживаемые значения `reason` зависят от SDK: каждый SDK передаёт только те причины закрытия сессии, которые распознаёт его реализация.

Подключение к SDK

C++

Подключите заголовок метрик OpenTelemetry из YDB C++ SDK и зарегистрируйте `MetricRegistry` в `TDriverConfig`:

```
#include <ydb-cpp-sdk/client/driver/driver.h>
#include <ydb-cpp-sdk/open_telemetry/metrics.h>

#include <opentelemetry/exporters/otlp/otlp_http_metric_exporter_factory.h>
#include <opentelemetry/exporters/otlp/otlp_http_metric_exporter_options.h>
#include <opentelemetry/sdk/metrics/meter_provider.h>
#include <opentelemetry/sdk/metrics/view/view_registry.h>
#include <opentelemetry/sdk/metrics/export/periodic_exporting_metric_reader_factory.h>
#include <opentelemetry/sdk/resource/resource.h>
#include <opentelemetry/metrics/provider.h>

namespace sdkmetrics = opentelemetry::sdk::metrics;
namespace otlp      = opentelemetry::exporter::otlp;
namespace resource  = opentelemetry::sdk::resource;
using namespace NYdb;

// 1. Инициализируем провайдер метрик OTel
otlp::OtlpHttpMetricExporterOptions opts;
opts.url = "http://localhost:4318/v1/metrics";

auto exporter = otlp::OtlpHttpMetricExporterFactory::Create(opts);

sdkmetrics::PeriodicExportingMetricReaderOptions readerOpts;
readerOpts.export_interval_millis = std::chrono::milliseconds(5000);
auto reader = sdkmetrics::PeriodicExportingMetricReaderFactory::Create(
    std::move(exporter), readerOpts);

auto res = resource::Resource::Create({{"service.name", "my-service"}});
auto rawProvider = std::make_shared<sdkmetrics::MeterProvider>(
    std::unique_ptr<sdkmetrics::ViewRegistry>(new sdkmetrics::ViewRegistry()), res);
rawProvider->AddMetricReader(std::move(reader));

std::shared_ptr<opentelemetry::metrics::MeterProvider> meterProvider = rawProvider;
opentelemetry::metrics::Provider::SetMeterProvider(meterProvider);

// 2. Оборачиваем в регистратор метрик YDB
auto ydbMetricRegistry = NMetrics::CreateOtelMetricRegistry(meterProvider);

// 3. Создаём драйвер YDB с включёнными метриками
auto driverConfig = TDriverConfig()
    .SetEndpoint("localhost:2136")
    .SetDatabase("/local")
    .SetMetricRegistry(ydbMetricRegistry);

TDriver driver(driverConfig);
```

Метрики и трассировку можно подключить вместе, зарегистрировав в `TDriverConfig` одновременно `MetricRegistry` и `TraceProvider`.

Go

Установите адаптер OpenTelemetry для YDB Go SDK:

```
go get github.com/ydb-platform/ydb-go-sdk-otel
```

Настройте `MeterProvider`, получите из него `Meter` и передайте его в адаптер `ydbOtel.WithMetrics`, который подключается к драйверу через опцию `ydb.Open`. Детализацию собираемых метрик можно настроить через `ydbOtel.WithDetailer`:

```
package main

import (
    "context"
    "os"

    "go.opentelemetry.io/otel"
    "go.opentelemetry.io/otel/attribute"
    "go.opentelemetry.io/otel/exporters/otlp/otlpmetric/otlpmetricgrpc"
    "go.opentelemetry.io/otel/sdk/metric"
    "go.opentelemetry.io/otel/sdk/resource"

    "github.com/ydb-platform/ydb-go-sdk/v3"
```

```
    "github.com/ydb-platform/ydb-go-sdk/v3/trace"
    ydbOtel "github.com/ydb-platform/ydb-go-sdk-otel"
)

func main() {
    ctx := context.Background()

    exporter, err := otlpmetricgrpc.New(ctx,
        otlpmetricgrpc.WithEndpoint("localhost:4317"),
        otlpmetricgrpc.WithInsecure(),
    )
    if err != nil {
        panic(err)
    }
    res, _ := resource.Merge(resource.Default(), resource.NewSchemaless(
        attribute.String("service.name", "my-service"),
    ))
    mp := metric.NewMeterProvider(
        metric.WithReader(metric.NewPeriodicReader(exporter)),
        metric.WithResource(res),
    )
    defer mp.Shutdown(ctx)
    otel.SetMeterProvider(mp)

    db, err := ydb.Open(ctx,
        os.Getenv("YDB_CONNECTION_STRING"),
        ydbOtel.WithMetrics(
            mp.Meter("ydb-go-sdk"),
            ydbOtel.WithDetailer(trace.DetailsAll),
        ),
    )
    if err != nil {
        panic(err)
    }
    defer db.Close(ctx)
}
```

Java

> **Примечание**
>
> В Java SDK на данный момент поддерживаются только метрики пула сессий.

Передайте свой `OpenTelemetry` в SDK через адаптер `OpenTelemetryMeter` и метод `QueryClient.Builder#withMeter`:

```
import io.opentelemetry.api.OpenTelemetry;
import io.opentelemetry.exporter.otlp.metrics.OtlpGrpcMetricExporter;
import io.opentelemetry.sdk.OpenTelemetrySdk;
import io.opentelemetry.sdk.metrics.SdkMeterProvider;
import io.opentelemetry.sdk.metrics.export.PeriodicMetricReader;

import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.core.metrics.OpenTelemetryMeter;
import tech.ydb.query.QueryClient;

SdkMeterProvider meterProvider = SdkMeterProvider.builder()
    .registerMetricReader(PeriodicMetricReader
        .builder(OtlpGrpcMetricExporter.getDefault())
        .build())
    .build();

OpenTelemetry openTelemetry = OpenTelemetrySdk.builder()
    .setMeterProvider(meterProvider)
    .build();

try (GrpcTransport transport = GrpcTransport
        .forConnectionString(System.getenv("YDB_CONNECTION_STRING"))
        .build();
     QueryClient queryClient = QueryClient.newClient(transport)
        .withMeter(OpenTelemetryMeter.fromOpenTelemetry(openTelemetry))
        .sessionPoolName("my-app")
        .build()) {
    // Используйте queryClient
}
meterProvider.close();
```

Если уже настроен глобальный `GlobalOpenTelemetry`, можно использовать `OpenTelemetryMeter.createGlobal()`. `TableClient` также поддерживает `withMeter(...)`.

Python

Функциональность на данный момент не поддерживается.

C#

Добавьте NuGet-пакет:

```
dotnet add package Ydb.Sdk.OpenTelemetry
```

Зарегистрируйте инструментацию YDB в конвейере OpenTelemetry-метрик:

```
services.AddOpenTelemetry()
    .WithMetrics(builder => builder
        .AddYdb()
        .AddOtlpExporter());
```

Или с использованием standalone `MeterProvider`:

```
using var meterProvider = Sdk.CreateMeterProviderBuilder()
    .SetResourceBuilder(ResourceBuilder.CreateDefault().AddService("my-service"))
    .AddYdb()
    .AddOtlpExporter()
    .Build();
```

Имя пула сессий задаётся параметром `PoolName=` в строке подключения `YdbDataSource`. Полный пример с нагрузкой и Grafana — в репозитории SDK (Ydb.Sdk.AdoNet.OpenTelemetry/Metrics).

JavaScript

Установите пакет телеметрии YDB JavaScript SDK и OpenTelemetry SDK:

```
npm install @ydbjs/telemetry @opentelemetry/sdk-node @opentelemetry/exporter-metrics-otlp-http
```

Инициализируйте `NodeSDK` до создания `Driver` и зарегистрируйте инструментацию `@ydbjs/telemetry`:

```
import { NodeSDK } from '@opentelemetry/sdk-node'
import { OTLPMetricExporter } from '@opentelemetry/exporter-metrics-otlp-http'
import { PeriodicExportingMetricReader } from '@opentelemetry/sdk-metrics'
import { Driver } from '@ydbjs/core'
import { query } from '@ydbjs/query'
import { register } from '@ydbjs/telemetry'

const sdk = new NodeSDK({
    serviceName: 'my-service',
    metricReader: new PeriodicExportingMetricReader({
        exporter: new OTLPMetricExporter({
            url: 'http://localhost:4318/v1/metrics',
        }),
    }),
})
sdk.start()

// Должно быть вызвано ДО создания Driver: пакет подписывается на
// diagnostics_channel события SDK и преобразует их в OTel metrics.
const instrumentation = register()

const driver = new Driver(process.env.YDB_CONNECTION_STRING)
await driver.ready()

const sql = query(driver)
await sql`SELECT 1`

instrumentation.disable()
await driver.close()
await sdk.shutdown()
```

Пакет `@ydbjs/telemetry` подписывается на события `node:diagnostics_channel` из `@ydbjs/core`, `@ydbjs/query`, `@ydbjs/auth` и `@ydbjs/retry` и публикует метрики из списка выше. Длительность операций экспортируется как `db.client.operation.duration` (семантические конвенции OpenTelemetry), а не `ydb.client.operation.duration`. Текущее число ожидающих сессию запросов — gauge `ydb.query.session.acquire.pending`, а не counter `ydb.query.session.pending_requests`. Подробнее см. репозиторий JavaScript SDK.

Кроме того, JavaScript SDK публикует собственные метрики повторных попыток с именами и семантикой, отличными от `ydb.client.retry.*`:

Имя	Тип	Единица	Описание
`ydb.retry.attempts`	Counter	`{attempt}`	Монотонный счётчик попыток повторного выполнения с тегом исхода `ydb.retry.outcome` (например, `success`, `retried`). В отличие от гистограммы `ydb.client.retry.attempts`, не показывает распределение числа попыток на один запрос.
`ydb.retry.duration`	Histogram	`s`	Суммарная длительность цикла повторных попыток, включая backoff (аналог `ydb.client.retry.duration`).

Rust

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#268

PHP

Функциональность на данный момент не поддерживается.

Включение метрик в Prometheus

Ниже приведены примеры кода включения метрик в Prometheus в разных YDB SDK.

C++

Функциональность на данный момент не поддерживается.

Go

Native SDK

```
package main

import (
    "context"

    "github.com/prometheus/client_golang/prometheus"
    metrics "github.com/ydb-platform/ydb-go-sdk-prometheus/v2"
    "github.com/ydb-platform/ydb-go-sdk/v3"
    "github.com/ydb-platform/ydb-go-sdk/v3/trace"
)

func main() {
    ctx := context.Background()
    registry := prometheus.NewRegistry()
    db, err := ydb.Open(ctx,
        os.Getenv("YDB_CONNECTION_STRING"),
        metrics.WithTraces(
            registry,
            metrics.WithDetails(trace.DetailsAll),
            metrics.WithSeparator("_"),
        ),
    )
    if err != nil {
        panic(err)
    }
    defer db.Close(ctx)
    ...
}
```

database/sql

```
package main

import (
    "context"
    "database/sql"

    "github.com/prometheus/client_golang/prometheus"
    metrics "github.com/ydb-platform/ydb-go-sdk-prometheus/v2"
    "github.com/ydb-platform/ydb-go-sdk/v3"
    "github.com/ydb-platform/ydb-go-sdk/v3/trace"
)

func main() {
    ctx := context.Background()
    registry := prometheus.NewRegistry()
    nativeDriver, err := ydb.Open(ctx,
        os.Getenv("YDB_CONNECTION_STRING"),
        metrics.WithTraces(
            registry,
            metrics.WithDetails(trace.DetailsAll),
            metrics.WithSeparator("_"),
        ),
    )
    if err != nil {
        panic(err)
    }
    defer nativeDriver.Close(ctx)

    connector, err := ydb.Connector(nativeDriver)
    if err != nil {
        panic(err)
    }

    db := sql.OpenDB(connector)
    defer db.Close()
    ...
}
```

Java

Функциональность на данный момент не поддерживается.

Python

Функциональность на данный момент не поддерживается.

C#

Функциональность на данный момент не поддерживается.

JavaScript

Функциональность на данный момент не поддерживается.

Rust

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#267

PHP

Функциональность на данный момент не поддерживается.

Трассировка с OpenTelemetry

YDB SDK инструментируют операции Query Service спанами OpenTelemetry, обеспечивая распределённую трассировку от кода приложения до каждого gRPC-вызова к YDB. Спаны экспортируются по стандартному протоколу OTLP и совместимы с Jaeger, Grafana Tempo, Zipkin и любым другим бэкендом, поддерживающим OpenTelemetry.

Создаваемые спаны

В настоящее время спаны поддерживаются для операций Query Service. Поддержка топиков и других сервисов планируется в будущих версиях SDK.

Создаются следующие спаны:

Спан	Тип	Описание
`ydb.RunWithRetry`	`Internal`	Охватывает весь цикл повторных попыток для одной операции
`ydb.Try`	`Internal`	Один спан на каждую попытку, включая первую; дочерние RPC-спаны прикрепляются к нему
`ydb.CreateSession`	`Client`	Создание сессии через gRPC `CreateSession` и `AttachStream`
`ydb.ExecuteQuery`	`Client`	Выполнение одного YQL-запроса
`ydb.BeginTransaction`	`Client`	Явный вызов начала транзакции
`ydb.Commit`	`Client`	Фиксация транзакции
`ydb.Rollback`	`Client`	Откат транзакции
`ydb.Driver.Initialize`	`Internal`	Первичная инициализация драйвера: обнаружение кластера и аутентификация

Типичное дерево спанов для транзакционной операции с повторной попыткой выглядит следующим образом:

```
ydb.RunWithRetry  (Internal)
├─ ydb.Try        (Internal)   ← 1-я попытка: ERROR
│  ├─ ydb.ExecuteQuery (Client)
│  ├─ ydb.ExecuteQuery (Client)
│  └─ ydb.Commit       (Client) ← ERROR: Transaction Lock Invalidated
└─ ydb.Try        (Internal)   ← 2-я попытка: SUCCESS, ydb.retry.backoff_ms=50
   ├─ ydb.ExecuteQuery (Client)
   ├─ ydb.ExecuteQuery (Client)
   └─ ydb.Commit       (Client)
```

`ydb.RunWithRetry` является родительским спаном для всей операции с ретраями. На каждую попытку выполнения создаётся отдельный дочерний спан `ydb.Try`: первый `ydb.Try` соответствует первой попытке, второй — первой **повторной** попытке и так далее. RPC-спаны конкретной попытки, например `ydb.ExecuteQuery` и `ydb.Commit`, создаются внутри соответствующего `ydb.Try`.

> **Примечание**
>
> Если попытка завершилась ошибкой, её спан `ydb.Try` завершается со статусом ошибки. При повторной попытке создаётся новый `ydb.Try`; начиная со второй попытки на нём указывается атрибут `ydb.retry.backoff_ms` — время ожидания перед этой попыткой в миллисекундах. Это ожидание входит в длительность следующего спана `ydb.Try`: спан начинается перед backoff-паузой, затем после паузы выполняются RPC-вызовы этой попытки.

Атрибуты спанов

SDK использует как стандартные атрибуты OpenTelemetry semantic conventions, так и YDB-специфичные расширения.

Стандартные атрибуты OpenTelemetry

Следующие атрибуты относятся к стабильным OpenTelemetry semantic conventions и могут обрабатываться бэкендами трассировки как стандартные:

Атрибут	Где устанавливается	Описание
`db.system.name`	RPC-спаны	Всегда `"ydb"`
`db.namespace`	RPC-спаны	Путь к базе данных YDB
`server.address`	RPC-спаны	Основной хост из строки подключения
`server.port`	RPC-спаны	Основной порт из строки подключения
`network.peer.address`	RPC-спаны	Фактический gRPC-эндпоинт, использованный для вызова
`network.peer.port`	RPC-спаны	Фактический порт gRPC-эндпоинта, использованный для вызова
`error.type`	Спаны, завершившиеся ошибкой	Тип ошибки. Например: `"transport_error"`, `"ydb_error"` или полное имя класса исключения
`db.response.status_code`	RPC-спаны при `YdbException`	Текстовое название статуса YDB из ошибки, например `ABORTED`, `UNAVAILABLE`, `OVERLOADED`

YDB-специфичные атрибуты

Следующие атрибуты являются расширениями YDB поверх стандартных semantic conventions:

Атрибут	Где устанавливается	Описание
`ydb.node.id`	RPC-спаны	Идентификатор узла YDB, обработавшего запрос
`ydb.node.dc`	RPC-спаны	Датацентр узла YDB, обработавшего запрос
`ydb.retry.backoff_ms`	Спаны `ydb.Try`, начиная со второй попытки	Время ожидания перед повторной попыткой в миллисекундах

Контекст трассировки W3C

SDK автоматически прокидывает заголовок W3C `traceparent` в каждый исходящий gRPC-вызов. Это позволяет серверу YDB трассировать внутренние операции в рамках той же трассы — без дополнительной настройки. Подробнее о серверной трассировке — в разделе Передача внешнего trace-id в YDB.

Подключение к SDK

C++

Подключите заголовок трассировки OpenTelemetry из YDB C++ SDK и добавьте зависимость на OTel C++ SDK:

```
#include <ydb-cpp-sdk/client/driver/driver.h>
#include <ydb-cpp-sdk/open_telemetry/trace.h>

#include <opentelemetry/exporters/otlp/otlp_http_exporter_factory.h>
#include <opentelemetry/exporters/otlp/otlp_http_exporter_options.h>
#include <opentelemetry/sdk/trace/tracer_provider.h>
#include <opentelemetry/sdk/trace/simple_processor_factory.h>
#include <opentelemetry/sdk/resource/resource.h>
#include <opentelemetry/trace/provider.h>

namespace sdktrace = opentelemetry::sdk::trace;
namespace otlp     = opentelemetry::exporter::otlp;
namespace resource = opentelemetry::sdk::resource;
using namespace NYdb;

// 1. Инициализируем провайдер трассировки OTel
otlp::OtlpHttpExporterOptions opts;
opts.url = "http://localhost:4318/v1/traces";
auto exporter  = otlp::OtlpHttpExporterFactory::Create(opts);
auto processor = sdktrace::SimpleSpanProcessorFactory::Create(std::move(exporter));
auto res       = resource::Resource::Create({{"service.name", "my-service"}});
auto otelProvider = std::make_shared<sdktrace::TracerProvider>(
    std::move(processor), res);
opentelemetry::trace::Provider::SetTracerProvider(otelProvider);

// 2. Оборачиваем в провайдер трассировки YDB
auto ydbTraceProvider = NTrace::CreateOtelTraceProvider(otelProvider);

// 3. Создаём драйвер YDB с включённой трассировкой
auto driverConfig = TDriverConfig()
    .SetEndpoint("localhost:2136")
    .SetDatabase("/local")
    .SetTraceProvider(ydbTraceProvider);

TDriver driver(driverConfig);
```

Go

Установите адаптер OpenTelemetry для YDB Go SDK:

```
go get github.com/ydb-platform/ydb-go-sdk-otel
```

Настройте `TracerProvider` и передайте адаптер в `ydb.Open`:

```
package main

import (
    "context"
    "os"

    "go.opentelemetry.io/otel"
    "go.opentelemetry.io/otel/attribute"
    "go.opentelemetry.io/otel/exporters/otlp/otlptrace/otlptracegrpc"
    "go.opentelemetry.io/otel/sdk/resource"
    sdktrace "go.opentelemetry.io/otel/sdk/trace"

    "github.com/ydb-platform/ydb-go-sdk/v3"
    ydbOtel "github.com/ydb-platform/ydb-go-sdk-otel"
)

func main() {
    ctx := context.Background()

    exporter, err := otlptracegrpc.New(ctx,
        otlptracegrpc.WithEndpoint("localhost:4317"),
        otlptracegrpc.WithInsecure(),
    )
    if err != nil {
```

```
        panic(err)
    }
    res, _ := resource.Merge(resource.Default(), resource.NewSchemaless(
        attribute.String("service.name", "my-service"),
    ))
    tp := sdktrace.NewTracerProvider(
        sdktrace.WithBatcher(exporter),
        sdktrace.WithResource(res),
    )
    defer tp.Shutdown(ctx)
    otel.SetTracerProvider(tp)

    db, err := ydb.Open(ctx,
        os.Getenv("YDB_CONNECTION_STRING"),
        ydbOtel.WithTraces(
            ydbOtel.WithTracer(tp.Tracer("ydb-go-sdk")),
        ),
    )
    if err != nil {
        panic(err)
    }
    defer db.Close(ctx)
}
```

Java

Добавьте зависимости YDB SDK и OpenTelemetry (пример для Maven):

```
<dependency>
    <groupId>tech.ydb</groupId>
    <artifactId>ydb-sdk-core</artifactId>
    <version>${ydb.sdk.version}</version>
</dependency>
<dependency>
    <groupId>io.opentelemetry</groupId>
    <artifactId>opentelemetry-sdk</artifactId>
    <version>${otel.version}</version>
</dependency>
<dependency>
    <groupId>io.opentelemetry</groupId>
    <artifactId>opentelemetry-exporter-otlp</artifactId>
    <version>${otel.version}</version>
</dependency>
```

Создайте экземпляр OpenTelemetry SDK и передайте его в транспорт через `OpenTelemetryTracer`:

```
import io.opentelemetry.api.OpenTelemetry;
import io.opentelemetry.exporter.otlp.trace.OtlpGrpcSpanExporter;
import io.opentelemetry.sdk.OpenTelemetrySdk;
import io.opentelemetry.sdk.resources.Resource;
import io.opentelemetry.sdk.trace.SdkTracerProvider;
import io.opentelemetry.sdk.trace.export.BatchSpanProcessor;
import io.opentelemetry.semconv.resource.attributes.ResourceAttributes;
import tech.ydb.core.auth.CloudAuthHelper;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.core.opentelemetry.OpenTelemetryTracer;
import tech.ydb.query.QueryClient;

Resource resource = Resource.getDefault().toBuilder()
    .put(ResourceAttributes.SERVICE_NAME, "my-service")
    .build();

SdkTracerProvider tracerProvider = SdkTracerProvider.builder()
    .setResource(resource)
    .addSpanProcessor(BatchSpanProcessor.builder(
        OtlpGrpcSpanExporter.builder()
            .setEndpoint("http://localhost:4317")
            .build()
    ).build())
    .build();

OpenTelemetry openTelemetry = OpenTelemetrySdk.builder()
    .setTracerProvider(tracerProvider)
    .build();

try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString)
        .withAuthProvider(CloudAuthHelper.getAuthProviderFromEnviron())
        .withTracer(OpenTelemetryTracer.fromOpenTelemetry(openTelemetry))
        .build();
     QueryClient queryClient = QueryClient.newClient(transport).build()) {
    // Используйте queryClient здесь
}
```

При использовании JDBC-драйвера достаточно добавить параметр `enableOpenTelemetryTracer=true` в строку подключения — драйвер подхватит глобальный OTel-провайдер автоматически:

```
jdbc:ydb://<host>:<port>/<database>?enableOpenTelemetryTracer=true
```

Python

Установите дополнительные зависимости `opentelemetry` и экспортёр OTLP:

```
pip install ydb[opentelemetry]
pip install opentelemetry-exporter-otlp-proto-grpc
```

Вызовите `enable_tracing()` после настройки глобального `TracerProvider`:

```
from opentelemetry import trace
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchSpanProcessor
from opentelemetry.exporter.otlp.proto.grpc.trace_exporter import OTLPSpanExporter
from opentelemetry.sdk.resources import Resource

import ydb
from ydb.opentelemetry import enable_tracing

resource = Resource(attributes={"service.name": "my-service"})
provider = TracerProvider(resource=resource)
provider.add_span_processor(
    BatchSpanProcessor(OTLPSpanExporter(endpoint="http://localhost:4317"))
)
trace.set_tracer_provider(provider)

enable_tracing()

with ydb.Driver(endpoint="grpc://localhost:2136", database="/local") as driver:
    driver.wait(timeout=5)
    with ydb.QuerySessionPool(driver) as pool:
        pool.execute_with_retries("SELECT 1")

provider.shutdown()
```

C#

Добавьте NuGet-пакет:

```
dotnet add package Ydb.Sdk.OpenTelemetry
```

Зарегистрируйте инструментацию YDB при настройке OpenTelemetry в вашем сервисе:

```
services.AddOpenTelemetry()
    .WithTracing(builder => builder
        .AddYdb()
        .AddOtlpExporter());
```

JavaScript

Установите `@ydbjs/telemetry` вместе с OpenTelemetry Node SDK и OTLP-экспортёром:

```
npm install @ydbjs/telemetry @opentelemetry/sdk-node @opentelemetry/exporter-trace-otlp-http
```

Инициализируйте `NodeSDK` до создания драйвера и вызовите `register()` из `@ydbjs/telemetry`:

```
import { NodeSDK } from '@opentelemetry/sdk-node'
import { OTLPTraceExporter } from '@opentelemetry/exporter-trace-otlp-http'
import { Driver } from '@ydbjs/core'
import { query } from '@ydbjs/query'
import { register } from '@ydbjs/telemetry'

const sdk = new NodeSDK({
    serviceName: 'my-service',
    traceExporter: new OTLPTraceExporter({ url: 'http://localhost:4318/v1/traces' }),
})
sdk.start()

// Должно быть вызвано ДО создания Driver — middleware пропагации W3C
// trace context устанавливается один раз при construction'е драйвера.
const instrumentation = register()

using driver = new Driver(process.env.YDB_CONNECTION_STRING)
await driver.ready()
await using sql = query(driver)
// ...

instrumentation.disable()
await sdk.shutdown()
```

Альтернативно, через `--import` для автозагрузки до старта приложения:

```
node --import @opentelemetry/sdk-node/register --import @ydbjs/telemetry/register your-app.js
```

Rust

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#268

PHP

Функциональность на данный момент не поддерживается.

Включение трассировки в Jaeger

Ниже приведены примеры кода включения трассировки в Jaeger в разных YDB SDK.

C++

Функциональность на данный момент не поддерживается.

Go

Native SDK

```
package main

import (
    "context"
    "time"

    "github.com/opentracing/opentracing-go"
    jaegerConfig "github.com/uber/jaeger-client-go/config"

    "github.com/ydb-platform/ydb-go-sdk/v3"
    "github.com/ydb-platform/ydb-go-sdk/v3/trace"

    tracing "github.com/ydb-platform/ydb-go-sdk-opentracing"
)

const (
    tracerURL   = "localhost:5775"
    serviceName = "ydb-go-sdk"
)

func main() {
    tracer, closer, err := jaegerConfig.Configuration{
        ServiceName: serviceName,
        Sampler: &jaegerConfig.SamplerConfig{
            Type:  "const",
            Param: 1,
        },
        Reporter: &jaegerConfig.ReporterConfig{
            LogSpans:            true,
            BufferFlushInterval: 1 * time.Second,
            LocalAgentHostPort:  tracerURL,
        },
    }.NewTracer()
    if err != nil {
        panic(err)
    }

    defer closer.Close()

    // set global tracer of this application
    opentracing.SetGlobalTracer(tracer)

    span, ctx := opentracing.StartSpanFromContext(context.Background(), "client")
    defer span.Finish()

    db, err := ydb.Open(ctx,
        os.Getenv("YDB_CONNECTION_STRING"),
        tracing.WithTraces(tracing.WithDetails(trace.DetailsAll)),
    )
    if err != nil {
        panic(err)
    }
    defer db.Close(ctx)
    ...
}
```

database/sql

```
package main

import (
    "context"
    "database/sql"
    "time"

    "github.com/opentracing/opentracing-go"
    jaegerConfig "github.com/uber/jaeger-client-go/config"

    "github.com/ydb-platform/ydb-go-sdk/v3"
    "github.com/ydb-platform/ydb-go-sdk/v3/trace"

    tracing "github.com/ydb-platform/ydb-go-sdk-opentracing"
)

const (
    tracerURL   = "localhost:5775"
    serviceName = "ydb-go-sdk"
)
```

```
func main() {
    tracer, closer, err := jaegerConfig.Configuration{
        ServiceName: serviceName,
            Sampler: &jaegerConfig.SamplerConfig{
            Type:  "const",
            Param: 1,
        },
        Reporter: &jaegerConfig.ReporterConfig{
            LogSpans:            true,
            BufferFlushInterval: 1 * time.Second,
            LocalAgentHostPort:  tracerURL,
        },
    }.NewTracer()
    if err != nil {
        panic(err)
    }

    defer closer.Close()

    // set global tracer of this application
    opentracing.SetGlobalTracer(tracer)

    span, ctx := opentracing.StartSpanFromContext(context.Background(), "client")
    defer span.Finish()

    nativeDriver, err := ydb.Open(ctx,
        os.Getenv("YDB_CONNECTION_STRING"),
        tracing.WithTraces(tracing.WithDetails(trace.DetailsAll)),
    )
    if err != nil {
        panic(err)
    }
    defer nativeDriver.Close(ctx)

    connector, err := ydb.Connector(nativeDriver)
    if err != nil {
        panic(err)
    }

    db := sql.OpenDB(connector)
    defer db.Close()
    ...
}
```

Java

Функциональность на данный момент не поддерживается.

Python

Функциональность на данный момент не поддерживается.

C#

Функциональность на данный момент не поддерживается.

JavaScript

Функциональность на данный момент не поддерживается.

Rust

Функциональность на данный момент не поддерживается.

Для распределённой трассировки используйте `tracing` и экспорт через OpenTelemetry (#268).

PHP

Функциональность на данный момент не поддерживается.

Сравнение возможностей SDK

Данный раздел позволяет сравнить возможности YDB SDK, которые реализованы для разных языков программирования.

Общий функционал

Функционал	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
Строка подключения вида 'grpc[s]://endpoint:port/database'	+	+	+	+	+	–	+	–
Поддержка SSL/TLS (системные сертификаты)	+	+	+	+	+	+	+	+
Поддержка SSL/TLS (кастомные сертификаты)	+	+	+	+	+	+	+	+
Configure/enable GRPC KeepAlive (keeping the connection alive in the background)	+	+	+	+	–	–	+	+
Регулярный прогон тестов SLO на последней версии кода	+	+	+	+	–	+	+	–
Шаблоны Issue в GitHub	–	+	+	–	+	+	+	+

Клиентская балансировка

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
Инициализация балансировщика через Discovery/ListEndpoints	+	+	+	+	+	+	+	+
Отключение клиентской балансировки (все запросы в начальный Endpoint)	+	+	+	–	+	+	+	+
Фоновый Discovery/ListEndpoints (раз в минуту по умолчанию)	+	+	+	+	+	+	+	+
Поддержка множества IP адресов в DNS-записи начального Endpoint, часть из которых может быть недоступна (DNS балансировка)	+	+	+	–	+	–	+	+
Пессимизация соединений на транспортных ошибках	+	+	+	+	+	+	+	–
Принудительный Discovery/ListEndpoints если пессимизировано более половины нод	+	+	+	+	–	+	+	–
Автоматическое определение ближайшего ДЦ / зоны доступности по TCP-пингам	–	+	+	+	–	+	+	–
Автоматическое определение ближайшего ДЦ / зоны доступности по ответу Discovery/ListEndpoints	+	+	+	+	–	–	–	–
Равномерный случайный выбор нод (по умолчанию)	+	+	+	+	+	+	+	+
Балансировка среди всех нод всех ДЦ (по умолчанию)	+	+	+	+	+	+	+	+
Балансировка среди всех нод конкретного ДЦ / зоны доступности (например, “a”, “vla”)	+	+	+	+	–	–	–	–
Балансировка среди всех нод всех локального ДЦ	+	+	+	+	–	–	–	–
Все запросы отправлять через соединение к начальному endpoint вместо клиентской балансировки (для использования в serverless)	+	+	+	+	+	+	+	+

Credentials providers

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
Anonymous (по умолчанию)	+	+	+	+	+	+	+	+
Static (user - password)	+	+	+	+	+	+	+	+
Token: IAM, Access token	+	+	+	+	+	+	+	+
Service account (Yandex.Cloud specific)	+	+	+	+	+	+	+	+
Metadata (Yandex.Cloud specific)	+	+	+	+	+	+	+	+

Поддержка типов данных YDB

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
Int/Uint(8,16,32,64)	+	+	+	+	+	+	+	+
Float, Double	+	+	+	+	+	+	+	+
Bool	+	+	+	+	+	+	+	+
String, Bytes	+	+	+	+	+	+	+	+
Utf8, Text	+	+	+	+	+	+	+	+
NULL, Optional, Void	+	+	+	+	+	–	+	–
Struct	+	+	+	+	+	+	+	+
List	+	+	+	+	+	–	+	+
Set	–	+	+	–	–	–	–	–
Tuple	+	+	+	+	+	+	+	+
Variant<Struct>,Variant<Tuple>	+	+	+	+	+	–	–	–
Date, DateTime, Timestamp, Interval	+	+	+	+	+	+	+	–
TzDate, TzDateTime, TzTimestamp	+	+	+	+	+	–	–	–
DyNumber	+	+	+	+	+	–	–	–
Decimal (120 bits)	+	+	+	+	+	+	–	+
Json, JsonDocument, Yson	+	+	+	+	+	+	+	–
Широкие типы дат (Date32, Datetime64, Timestamp64, Interval64)	+	+	+	+	–	+	+	–

Клиенты gRPC-сервисов YDB

Discovery

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
ListEndpoints	+	–	+	–	+	–	–	+
WhoAmI	+	–	+	–	+	–	–	+

Query

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
Пул сессий	+	+	+	+	+	+	+	–
Ограничение количества одновременных сессий на клиенте	+	+	+	+	–	+	+	–
Несгораемый остаток сессий в пуле	+	–	+	+	–	+	+	–
Прогрев пула до указанного значения количества сессий при создании пула	–	–	+	–	–	–	+	–
Фоновый attach-стрим используется для очистки пула от плохих сессий	+	+	+	+	–	+	+	–
Ретраер на пуле сессий (объект для повторов - сессия)	+	+	+	+	–	+	+	–
Ретраер на пуле сессий (объект для повторов - транзакция)	–	+	+	–	–	–	+	–
Поддержка серверной балансировки сессий (запрос CreateSession должен содержать в metadata-заголовке "x-ydb-client-capabilities" значение "session-balancer")	+	–	+	+	–	+	+	–
Транзакции топик-таблица	+	+	+	–	+	–	+	–
* CreateSession	+	+	+	+	+	+	+	–
* DeleteSession	–	+	+	+	+	+	+	–
* AttachSession	+	+	+	+	+	+	+	–
* CommitTransaction	+	+	+	+	+	+	+	–
* BeginTransaction	+	+	+	+	+	–	+	–

* RollbackTransaction	+	+	+	+	+	+	+	–
* ExecuteQuery	+	+	+	+	+	+	+	–
* ExecuteScript	+	–	+	+	–	–	+	–
* FetchScriptResults	+	–	+	+	–	–	+	–
Выполнение запросов без явного создания сессии	+	–	+	–	+	+	+	–

Scheme

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
MakeDirectory	+	+	+	+	+	–	+	+
RemoveDirectory	+	+	+	+	+	–	+	+
ListDirectory	+	+	+	+	+	+	+	+
ModifyPermissions	+	+	+	–	+	–	–	+
DescribePath	+	+	+	+	+	–	+	+

Table

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
Пул сессий	+	+	+	+	+	+	+	+
Ограничение количества одновременных сессий на клиенте	+	+	+	+	+	+	+	–
Несгораемый остаток сессий в пуле	+	+	+	+	–	+	+	–
Прогрев пула до указанного значения количества сессий при создании пула	–	+	+	–	+	–	+	–
Фоновый KeepAlive для простаивающих сессий в пуле	+	+	+	+	+	+	+	–
Фоновое закрытие простаивающих сессий в пуле (лишние сессии)	+	+	+	+	–	+	+	–
Автоматическое выбрасывание сессии из пула при получении ошибок BAD_SESSION / SESSION_BUSY	+	+	+	+	+	+	+	+
Ретраер на пуле сессий (объект для повторов - сессия)	+	+	+	+	+	+	+	+
Ретраер на пуле сессий (объект для повторов - транзакция на сессии)	–	–	+	–	–	+	+	–
Поддержка graceful shutdown сессий ("session-close" в metadata "x-ydb-server-hints" - означает надо "забыть" сессию и больше ее не использовать)	+	+	+	+	–	+	+	–
Поддержка серверной балансировки сессий (запрос CreateSession должен содержать в metadata-заголовке "x-ydb-client-capabilities" значение "session-balancer")	+	+	+	+	–	+	+	–
Транзакции топик-таблица	+	+	–	–	+	+	+	–
CreateSession	+	+	+	+	+	+	+	+
DeleteSession	+	+	+	+	+	+	+	+
KeepAlive	+	+	+	+	+	+	+	+
CreateTable	+	+	+	+	+	+	+	+
DropTable	+	+	+	+	+	+	+	+
AlterTable	+	+	+	+	+	–	+	+
CopyTable	+	+	+	+	–	+	+	+
CopyTables	+	+	+	+	–	+	+	+
DescribeTable	+	+	+	+	+	–	+	+
ExplainDataQuery	+	+	+	+	–	–	+	+
PrepareDataQuery	+	+	+	+	+	+	+	+

ExecuteDataQuery	+	+	+	+	+	+	+	+
* Серверный кэш по умолчанию для всех запросов с параметрами (KeepInCache)	–	+	+	+	+	+	+	+
* Отдельная опция для включения/выключения серверного кэша для конкретного запроса	+	+	+	+	+	+	+	+
* Truncated result как ошибка (по дефолту)	–	–	+	–	+	+	+	+
* Truncated result как ошибка (как опция opt-in, opt-out)	–	–	+	–	+	–	+	–
ExecuteSchemeQuery	+	+	+	+	–	+	+	+
BeginTransaction	+	+	+	+	+	+	+	+
CommitTransaction	+	+	+	+	+	+	+	+
RollbackTransaction	+	+	+	+	+	+	+	+
DescribeTableOptions	+	+	+	+	–	–	+	+
StreamExecuteScanQuery	+	+	+	+	+	+	+	+
StreamReadTable	+	+	+	+	+	+	+	+
BulkUpsert	+	+	+	+	+	+	+	+
Управление атрибутами таблиц	+	–	+	+	–	–	+	–

Operation

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
Consumed Units из метаданных ответа на grpc-запрос (чтобы пользователь мог получить это)	+	+	+	+	+	–	–	–
Получение OperationId операции для long-polling статуса выполнения операции	+	+	+	+	–	+	+	–

ScriptingYQL

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
ExecuteYql	+	+	+	–	–	–	–	+
ExplainYql	+	+	+	–	–	–	–	+
StreamExecuteYql	+	–	+	–	–	–	–	–

Coordination

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
CreateNode	+	+	+	+	+	–	+	–
AlterNode	+	+	+	+	+	–	+	–
DropNode	+	+	+	+	+	–	+	–
DescribeNode	+	+	+	+	+	–	+	–
Session (leader election, распределенный лок)	+	+	+	+	+	–	+	–

Topic

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
CreateTopic	+	+	+	+	+	+	+	–
DescribeTopic	+	+	+	+	+	+	+	–
AlterTopic	+	+	+	+	+	+	+	–
DropTopic	+	+	+	+	+	+	+	–
StreamWrite	+	+	+	+	+	+	+	–
StreamRead	+	+	+	+	+	–	+	–
CommitOffset	+	+	+	+	–	–	–	–

Ratelimiter

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
CreateResource	+	–	+	–	–	–	–	–
AlterResource	+	–	+	–	–	–	–	–
DropResource	+	–	+	–	–	–	–	–
ListResources	+	–	+	–	–	–	–	–
DescribeResource	+	–	+	–	–	–	–	–
AcquireResource	+	–	+	–	+	–	–	–

Диагностика

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
Логирование событий SDK	–	–	+	+	+	+	+	+
Метрики SDK в Solomon / Monitoring	+	–	+	–	+	–	–	–
Метрики SDK в Prometheus	–	–	+	–	+	–	+	–
Трассировка событий SDK в OpenTelemetry	+	–	+	–	+	+	+	–

Примеры приложений

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
Аутентификация с помощью токена	–	+	+	+	+	+	+	+
Анонимная аутентификация	–	+	+	+	+	+	–	+
Аутентификация с помощью переменных окружения	–	–	+	+	+	–	+	+
Аутентификация с помощью metadata-сервиса	–	+	+	+	+	+	+	+
Аутентификация с помощью файла сервисного аккаунта	–	+	+	+	+	+	+	+
Аутентификация с помощью логина и пароля	–	+	+	+	+	+	+	–
Basic (series)	+	+	+	+	+	+	+	+
Bulk Upsert	+	+	+	+	+	+	+	+
Containers (Struct, Variant, List, Tuple)	+	+	+	–	–	+	–	–
Pagination	+	+	+	+	–	–	–	–
Partition policies	+	–	+	–	–	–	–	–
Read table	–	–	+	–	+	–	+	+
Secondary index Workaround	+	–	+	+	–	–	–	–
Secondary index builtin	+	–	+	–	–	–	–	–
TTL	+	–	+	–	–	–	–	–
TTL Readtable	+	–	+	–	–	–	–	–
URL Shortener (serverless yandex function)	–	–	+	–	+	–	–	–
Topic reader	+	+	+	+	+	+	+	–
Topic writer	+	+	+	+	+	+	+	–

Языки и API

- ADO.NET — доступ к YDB из .NET
- JDBC-драйвер для YDB
- YDB Model Context Protocol Server
- Генерация кода из YQL (sqlc-ydb)

ADO.NET — доступ к YDB из .NET

`Ydb.Sdk` — ADO.NET Data Provider для YDB, позволяющий программам, написанным на C#, Visual Basic или F#, получить доступ к YDB. Провайдер полностью реализован на C#, является бесплатным и с открытым исходным кодом.

Документация

- Начало работы с ADO.NET
- Установка ADO.NET
- Основное использование ADO.NET
- Параметры подключения ADO.NET
- Поддерживаемые типы данных в интеграции с ADO.NET и их соответствие
- Подключение ADO.NET к Yandex Cloud
- Использование Dapper

Начало работы с ADO.NET

Лучший способ использовать `Ydb.Sdk` — установить его NuGet-пакет.

`Ydb.Sdk.Ado` стремится быть полностью совместимым с ADO.NET; его API должен быть практически идентичен другим драйверам баз данных .NET.

Вот базовый фрагмент кода, который поможет вам начать:

```
await using var ydbDataSource = new YdbDataSource("Host=localhost;Port=2136;Database=/local;MaxSessionPool=5
0");
await using var ydbConnection = await ydbDataSource.OpenConnectionAsync();

var ydbCommand = ydbConnection.CreateCommand();
ydbCommand.CommandText = """
                         SELECT series_id, season_id, episode_id, air_date, title
                         FROM episodes
                         WHERE series_id = @series_id AND season_id > @season_id
                         ORDER BY series_id, season_id, episode_id
                         LIMIT @limit_size;
                         """;
ydbCommand.Parameters.Add(new YdbParameter("series_id", DbType.UInt64, 1U));
ydbCommand.Parameters.Add(new YdbParameter("season_id", DbType.UInt64, 1U));
ydbCommand.Parameters.Add(new YdbParameter("limit_size", DbType.UInt64, 3U));

var ydbDataReader = await ydbCommand.ExecuteReaderAsync();

_logger.LogInformation("Selected rows:");
while (await ydbDataReader.ReadAsync())
{
    _logger.LogInformation(
        "series_id: {series_id}, season_id: {season_id}, episode_id: {episode_id}, air_date: {air_date}, titl
e: {title}",
        ydbDataReader.GetUint64(0), ydbDataReader.GetUint64(1), ydbDataReader.GetUint64(2),
        ydbDataReader.GetDateTime(3), ydbDataReader.GetString(4));
}
```

Вы можете найти более подробную информацию о ADO.NET API в документации MSDN или во многих других руководствах в интернете.

Установка ADO.NET

Официальные версии `Ydb.Sdk` всегда доступны на nuget.org. Это рекомендуемый способ использования `Ydb.Sdk`.

```
dotnet add package Ydb.Sdk
```

Основное использование ADO.NET

В этой статье рассматриваются основные сценарии использования ADO.NET с YDB, включая подключение к базе данных, выполнение запросов и обработку результатов. Дополнительные сведения см. в основной документации.

Data Source

Отправной точкой для любой операции с базой данных является DbDataSource.

YdbDataSource можно создать несколькими способами.

Без параметров

Следующий код создаёт источник данных с настройками по умолчанию:

```
await using var ydbDataSource = new YdbDataSource();
```

В этом случае используется адрес подключения `grpc://localhost:2136/local` и анонимная аутентификация.

С использованием строки подключения

Пример создания с помощью строки подключения ADO.NET:

```
await using var ydbDataSource = new YdbDataSource(
    "Host=database-sample-grpc;Port=2135;Database=/root/database-sample");
```

Источник данных будет использовать URL: `grpc://database-sample-grpc:2135/root/database-sample`. Поддерживаемый набор настроек описан на странице параметров подключения.

Использование конструктора с аргументом `YdbConnectionStringBuilder`

```
var ydbConnectionBuilder = new YdbConnectionStringBuilder
{
    Host = "localhost",
    Port = 2136,
    Database = "/local",
    UseTls = false
};

await using var ydbDataSource = new YdbDataSource(ydbConnectionBuilder);
```

`YdbConnectionStringBuilder` поддерживает дополнительные настройки, помимо строки подключения, такие как логирование, расширенные параметры аутентификации.

Подключения

Подключение к YDB устанавливается через `YdbConnection`. Получение подключений осуществляется через `YdbDataSource`, это можно сделать через следующие методы:

YdbDataSource.OpenConnectionAsync

Открывает подключение к YDB с параметрами, указанными при создании `YdbDataSource` (см. раздел Ydb Data Source).

```
await using var ydbConnection = await ydbDataSource.OpenConnectionAsync();
```

Рекомендуется для длинных читающих запросов.

YdbDataSource.OpenRetryableConnectionAsync

Открывает подключение с автоматическими повторами операций (retry), учитывающее YDB Retry Policy (см. раздел Повторные попытки).

```
await using var ydbConnection = ydbDataSource.OpenRetryableConnectionAsync();
```

Особенности режима:

- Транзакции не поддерживаются. Попытка использовать транзакцию приведет к исключению (см. раздел Транзакции).
- Команды (`YdbCommand`), созданные из такого подключения, автоматически повторяют одиночные операции при временных ошибках.

> **Важно**
>
> При чтении будьте аккуратны с «длинными» выборками: они могут привести к переполнению памяти (Out Of Memory, OOM), поскольку результирующий набор считывается целиком для получения финальных статусов от сервера.

YdbDataSource.CreateConnection

> **Важно**
>
> Использование `YdbDataSource.CreateConnection` и конструктора `YdbConnection` (legacy API ADO.NET) не рекомендуется. По возможности используйте современные способы открытия подключения через `YdbDataSource.OpenConnectionAsync` или `YdbDataSource.OpenRetryableConnectionAsync`.

Если необходим именно этот способ — откройте подключение вручную:

- Вариант со строкой подключения:

```
await using var ydbConnection = new YdbConnection(
    "Host=database-sample-grpc;Port=2135;Database=/root/database-sample");
await ydbConnection.OpenAsync();
```

- Вариант с `YdbConnectionStringBuilder`:

```
var ydbConnectionBuilder = new YdbConnectionStringBuilder
{
    Host = "server",
    Port = 2135,
    Database = "/ru-prestable/my-table",
    UseTls = true
};
await using var ydbConnection = new YdbConnection(ydbConnectionBuilder);
await ydbConnection.OpenAsync();
```

- Вариант `YdbDataSource.CreateConnection`:

```
await using var ydbConnection = ydbDataSource.CreateConnection();
await ydbConnection.OpenAsync();
```

Pooling

Открытие нового подключения к YDB — затратная операция, поэтому провайдер использует пул подключений. При закрытии или освобождении объекта подключения оно не закрывается — экземпляр возвращается в пул, управляемый `Ydb.Sdk.Ado`. При следующем запросе подключение будет повторно использовано из пула. Это делает операции открытия и закрытия быстрыми: открывайте и закрывайте подключения по мере необходимости, не удерживайте их открытыми без нужды.

> **Примечание**
>
> Пул работает для подключений, открытых через `YdbDataSource` (например, `OpenConnectionAsync` / `OpenRetryableConnectionAsync`), а также для подключений, созданных вручную через конструктор `YdbConnection` с последующим вызовом `OpenAsync()`.

> **Примечание**
>
> Как это устроено внутри: для прикладного кода «подключение» — логическое. Под капотом операции — это RPC-вызовы поверх небольшого пула gRPC/HTTP/2-каналов. Провайдер также управляет пулом сессий таблиц. Эти детали прозрачны для пользователя и настраиваются параметрами пула (см. параметры Pooling).

Очистить пул сессий и закрыть все физические подключения с узлами YDB:

- `YdbDataSource.DisposeAsync()`: Освобождает источник данных. Закрывает связанные с ним пулы и сетевые каналы для его `ConnectionString`.

  ```
  await YdbDataSource.DisposeAsync()
  ```

- `YdbConnection.ClearPool`: Немедленно закрывает все незанятые подключения в пуле, связанном с `ConnectionString` указанного подключения. Активные подключения будут закрыты при возврате в пул.

  ```
  await YdbConnection.ClearPool(ydbConnection)
  ```

- `YdbConnection.ClearAllPools()`: Закрывает все незанятые соединения во всех пулах. Активные соединения закрываются при возврате.

  ```
  await YdbConnection.ClearAllPools()
  ```

Базовое выполнение SQL

Как только у вас есть `YdbConnection`, для выполнения SQL-запроса можно использовать `YdbCommand`:

```
await using var ydbCommand = new YdbCommand("SELECT some_field FROM some_table", ydbConnection);
await using var ydbDataReader = await ydbCommand.ExecuteReaderAsync();

while (await ydbDataReader.ReadAsync())
{
    Console.WriteLine(ydbDataReader.GetString(0));
}
```

Другие методы выполнения

Выше мы выполнили SQL с помощью ExecuteReaderAsync. Существуют различные способы выполнения команды, в зависимости от того, каких результатов вы от нее ожидаете:

1. ExecuteNonQueryAsync: выполняет SQL, который не возвращает никаких результатов, обычно это инструкции `INSERT`, `UPDATE` или `DELETE`.

> **Важно**
>
> YDB не возвращает количество затронутых строк.

2. ExecuteScalarAsync: выполняет SQL, который возвращает единственное скалярное значение.
3. ExecuteReaderAsync: выполняет SQL, который вернет полный результирующий набор. Возвращает `YdbDataReader`, который можно использовать для доступа к результирующему набору (как в приведенном выше примере).

Например, чтобы выполнить простой SQL `INSERT`, который ничего не возвращает, вы можете использовать `ExecuteNonQueryAsync` следующим образом:

```
await using var ydbCommand = new YdbCommand("INSERT INTO some_table (some_field) VALUES ('Hello YDB!'u)", ydb
Connection);
await ydbCommand.ExecuteNonQueryAsync();
```

Параметры

При отправке пользовательских данных в базу данных всегда рекомендуется использовать параметры, а не включать значения в SQL следующим образом:

```
await using var ydbConnection = await ydbDataSource.OpenConnectionAsync();

var ydbCommand = ydbConnection.CreateCommand();
ydbCommand.CommandText = """
                         SELECT series_id, season_id, episode_id, air_date, title
                         FROM episodes WHERE series_id = $series_id AND season_id > $season_id
                         ORDER BY series_id, season_id, episode_id
                         LIMIT $limit_size;
                         """;
ydbCommand.Parameters.Add(new YdbParameter("$series_id", DbType.UInt64, 1U));
ydbCommand.Parameters.Add(new YdbParameter("$season_id", DbType.UInt64, 1U));
ydbCommand.Parameters.Add(new YdbParameter("$limit_size", DbType.UInt64, 3U));

var ydbDataReader = await ydbCommand.ExecuteReaderAsync();
```

Параметры SQL-запроса можно задать с помощью класса `YdbParameter`.

В этом примере параметры `$series_id`, `$season_id` и `$limit_size` объявляются в SQL-запросе и затем добавляются в команду с помощью объектов `YdbParameter`.

Альтернативный стиль параметра с префиксом `@`

Параметры также можно указывать с использованием префикса `@`. В этом случае не требуется предварительное объявление переменных в самом запросе. Запрос будет выглядеть следующим образом:

```
ydbCommand.CommandText = """
                         SELECT series_id, season_id, episode_id, air_date, title
                         FROM episodes
                         WHERE series_id = @series_id AND season_id > @season_id
                         ORDER BY series_id, season_id, episode_id
                         LIMIT @limit_size;
                         """;
ydbCommand.Parameters.Add(new YdbParameter("series_id", DbType.UInt64, 1U));
ydbCommand.Parameters.Add(new YdbParameter("season_id", DbType.UInt64, 1U));
ydbCommand.Parameters.Add(new YdbParameter("limit_size", DbType.UInt64, 3U));
```

ADO.NET за вас подготовит запрос, чтобы переменные соответствовали YQL. Тип каждого параметра будет определен по YdbDbType, при его отсутствии — по DbType, иначе выводится из .NET-типа значения.

Parameter types

YDB имеет строго типизированную систему типов: столбцы и параметры имеют тип, а типы обычно не преобразуются неявным образом в другие типы. Это означает, что вам нужно подумать о том, какой тип вы будете отправлять: попытка вставить строку в целочисленный столбец (или наоборот) приведет к сбою.

Для получения дополнительной информации о поддерживаемых типах и их сопоставления смотрите эту страницу.

Транзакции

Интерактивные транзакции и повторы (retry) — ключевая часть работы с YDB.

YdbDataSource предоставляет вспомогательные методы, упрощающие выполнение кода в транзакции с автоматическими повторами.

```
await ydbDataSource.ExecuteInTransactionAsync(async ydbConnection =>
{
    var count = (int)(await new YdbCommand(ydbConnection)
        { CommandText = $"SELECT count FROM {tableName} WHERE id = 1" }
        .ExecuteScalarAsync())!;

    await new YdbCommand(ydbConnection)
    {
        CommandText = $"UPDATE {tableName} SET count = @count + 1 WHERE id = 1",
        Parameters = { new YdbParameter { Value = count, ParameterName = "count" } }
    }.ExecuteNonQueryAsync();
},
new YdbRetryPolicyConfig { MaxAttempts = 5 });
```

Повторные попытки выполняются в соответствии с политиками YDB (см. раздел Повторные попытки).

Транзакции YdbConnection

Создать транзакцию можно стандартным способом ADO.NET:

```
await using var ydbConnection = await ydbDataSource.OpenConnectionAsync();
await using var ydbTransaction = await connection.BeginTransactionAsync();
// ... команды в транзакции ...
await transaction.CommitAsync();
```

Важно

Подробнее про обработку ошибок (Transaction Locks Invalidated).

Примечание

Это допустимый и рекомендуемый для длительных чтений. Используйте транзакции чтения в режиме snapshot read-only. Такие транзакции дают консистентный срез, не берут блокировки записи и минимизируют конфликты.

Есть две сигнатуры этого метода с единственным параметром уровня изоляции:

- `BeginTransaction(TransactionMode transactionMode)`

  Параметр `Ydb.Sdk.Ado.TransactionMode` — это YDB-специфичный уровень изоляции, подробнее ознакомиться можно здесь.
- `BeginTransaction(IsolationLevel isolationLevel)`

  Параметр `System.Data.IsolationLevel` из стандарта ADO.NET. Поддерживаются следующие уровни изоляции: `Serializable` и `Unspecified`. Оба эквивалентны параметру `TxMode.SerializableRW`.

Вызов `BeginTransaction()` без параметров открывает транзакцию с уровнем `TxMode.SerializableRW`.

YDB не поддерживает вложенные или параллельные транзакции. В любой момент времени может выполняться только одна транзакция, и при запуске новой транзакции, в то время как другая уже выполняется, возникает исключение. Следовательно, нет необходимости передавать объект `YdbTransaction`, возвращаемый функцией `BeginTransaction()` в команды, которые вы выполняете. При запуске транзакции все последующие команды автоматически включаются до тех пор, пока не будет выполнена фиксация или откат. Однако для обеспечения максимальной переносимости лучше всего явно задать область транзакции для ваших команд.

Повторные попытки

Повторы (retry) — важная часть дизайна YDB. Провайдер ADO.NET предоставляет гибкую политику повторов с учетом специфики YDB.

Рекомендации по выбору подхода:

- Одиночные пишущие операции (без интерактивных транзакций): используйте `YdbDataSource.OpenRetryableConnectionAsync`.
- Транзакционные сценарии: используйте семейство методов `YdbDataSource.ExecuteInTransactionAsync`.
- Выполнение кода с автоматическими повторами вне транзакции: используйте семейство `YdbDataSource.ExecuteAsync`
- Долгие читающие операции: используйте обычный `YdbConnection`. Для консистентных «снимков» данных выполняйте чтение в read-only Snapshot-транзакции. Не используйте retry-подключение для «длинных» чтений, чтобы избежать избыточного буферизации результатов.

Передача настроек политики

В методы `OpenRetryableConnectionAsync`, `ExecuteInTransactionAsync`, `ExecuteAsync` можно передать `YdbRetryPolicyConfig`:

- Подключение с повторами:

  ```
  await using var conn = await ydbDataSource.OpenRetryableConnectionAsync(
      new YdbRetryPolicyConfig { MaxAttempts = 5 });
  ```

- Транзакция с повторами:

  ```
  await ydbDataSource.ExecuteInTransactionAsync(
      async conn => { /* ваш код */ },
      new YdbRetryPolicyConfig { MaxAttempts = 5 });
  ```

- Выполнение блока кода с повторами:

  ```
  await ydbDataSource.ExecuteAsync(
      async conn => { /* ваш код */ },
      new YdbRetryPolicyConfig { MaxAttempts = 5 });
  ```

Параметр	Описание	Значение по умолчанию
`MaxAttempts`	Общее число попыток, включая первую. Значение 1 полностью отключает повторы.	`10`

`EnableRetryIdempotence`	Включает повторы для статусов с неизвестным результатом выполнения на сервере. Используйте только для идемпотентных операций — иначе возможен повторный эффект выполнения.	`false`
`FastBackoffBaseMs`	Базовая задержка (мс) для быстрых повторов: ошибки, которые обычно быстро проходят (например, временная недоступность, TLI — Transaction Locks Invalidated). Экспоненциальный backoff с jitter.	`5`
`FastCapBackoffMs`	Максимальная задержка (мс) для быстрых повторов. Экспоненциальный backoff с jitter не превышает этот предел.	`500`
`SlowBackoffBaseMs`	Базовая задержка (мс) для «медленных» повторов: перегрузка, исчерпание ресурсов и т.п. Экспоненциальный backoff с jitter.	`50`
`SlowCapBackoffMs`	Максимальная задержка (мс) для «медленных» повторов. Экспоненциальный backoff с jitter не превышает этот предел.	`5000`

Кастомная политика повторных попыток

Для крайних случаев можно реализовать собственную политику, реализовав интерфейс `Ydb.Sdk.Ado.Retry.IRetryPolicy`. Политика получает `YdbException` и номер текущей попытки (attempt) и должна вернуть решение — повторять ли операцию и с какой задержкой.

> **Важно**
>
> Используя данный подход, вы точно должны быть уверены в том, что хотите, так как вы отказываетесь от рекомендуемых настроек.

Обработка ошибок

> **Примечание**
>
> Этот раздел актуален, если вы не используете встроенные повторы провайдера (см. раздел Повторные попытки).

Все исключения, связанные с операциями в базе данных, являются подклассами `YdbException`.

Для безопасной обработки ошибок, которые могут возникнуть при выполнении команд, вы можете использовать блок `try-catch`. Пример:

```
try
{
    await command.ExecuteNonQueryAsync();
}
catch (YdbException e)
{
    Console.WriteLine($"Error executing command: {e}");
}
```

Свойства исключения `YdbException`

Исключение `YdbException` содержит следующие свойства, которые помогут вам более точно обработать ошибку:

- `IsTransient`: Возвращает true, если ошибка временная и может быть устранена повторной попыткой. Например, такая ошибка может возникнуть в случае нарушения блокировки транзакции, когда транзакция не успела завершить коммит.
- `IsTransientWhenIdempotent`: Возвращает true, если ошибка временная и может быть устранена в случае повтора операции, при условии, что операция с базой данных является идемпотентной.
- `StatusCode`: Содержит код ошибки базы данных, который может быть полезен для логирования и детального анализа проблемы.

> **Важно**
>
> Обратите внимание, что ADO.NET не выполняет автоматических попыток повторения, и вам нужно реализовать повтор попытки в своем коде.

Примеры

Примеры приведены на GitHub по ссылке.

Параметры подключения ADO.NET

Для подключения к базе данных приложение предоставляет строку подключения, в которой указываются такие параметры, как хост, пользователь, пароль и т.д. Строки подключения имеют вид `keyword1=value; keyword2=value;`. Для получения дополнительной информации смотрите официальную страницу документации о строках подключения.

Все доступные параметры подключения определены как свойства в `YdbConnectionStringBuilder`.

Ниже приведены параметры строки подключения, которые ожидает `Ydb.Sdk.Ado`.

Базовые параметры подключения

Параметр	Описание	Значение по умолчанию
`Host`	Адрес хоста сервера YDB.	`localhost`
`Port`	Порт сервера YDB.	`2136`
`Database`	Путь к базе данных.	`/local`
`User`	Имя пользователя.	Не определено
`Password`	Пароль пользователя.	Не определено

Безопасность и шифрование

Параметр	Описание	Значение по умолчанию
`UseTls`	Определяет, следует ли устанавливать защищенное соединение по TLS (`grpcs`); при `false` — TLS не используется (`grpc`).	`false`
`RootCertificate`	Путь к файлу доверенного корневого/промежуточного сертификата сервера (PEM). Если параметр установлен, для `UseTls` автоматически устанавливается значение `true`.	Не определено

Pooling

Параметр	Описание	Значение по умолчанию
`MinPoolSize`	Минимальный размер пула сессий.	`0`
`MaxPoolSize`	Максимальный размер пула сессий.	`100`
`SessionIdleTimeout`	Время ожидания (в секундах) перед закрытием неактивных сессий в пуле, если общее число сессий превышает `MinPoolSize`.	`300`

Timeouts And Keepalive

Параметр	Описание	Значение по умолчанию
`ConnectTimeout`	Время ожидания (в секундах) при установлении соединения с сервером. Должно быть больше или равно 0. Установите значение `0` для бесконечного ожидания.	`10`
`CreateSessionTimeout`	Время ожидания (в секундах) при создании новой сессии. Должно быть больше или равно 0. Установите значение `0` для бесконечного ожидания.	`5`
`KeepAlivePingDelay`	Период простоя (в секундах), после которого отправляется keepalive ping при отсутствии трафика. Это свойство используется вместе с параметром `KeepAlivePingTimeout`, чтобы проверить, не разорвано ли соединение. Значение должно быть больше или равно `1` секунде. Установите значение `0`, чтобы отключить проверку связи keep alive.	`10`
`KeepAlivePingTimeout`	Время ожидания (в секундах) после отправки keepalive ping; если за это время не получено ни одного сообщения, соединение закрывается. Время ожидания должно быть больше или равно `1` секунде. Установите значение `0`, чтобы отключить тайм-аут для поддержания соединения в режиме ожидания.	`10`

Performance

Параметр	Описание	Значение по умолчанию
`EnableMultipleHttp2Connections`	Определяет, следует ли автоматически масштабировать HTTP/2-соединения в пределах одного канала gRPC до одного узла кластера. Это требуется редко, но может повысить производительность в сценариях с высокой нагрузкой на один узел. При `false` используется одно HTTP/2-соединение на узел.	`false`
`MaxSendMessageSize`	Максимальный размер исходящих сообщений в байтах. Примечание: на сервере действует лимит `64 МБ`; большие сообщения отклоняются ошибкой `ResourceExhausted`.	`67108864 (64 MB)`
`MaxReceiveMessageSize`	Максимальный размер входящих сообщений, байт.	`67108864 (64 MB)`

`DisableServerBalancer`	Определяет, следует ли отключать балансировку сессий на стороне сервера; при `false` балансировка включена.	`false`

Параметры полезные для Serverless приложений

Параметр	Описание	Значение по умолчанию
`DisableDiscovery`	Определяет, следует ли отключать автоматическое обнаружение узлов и использовать прямое gRPC-подключение по адресу из строки подключения; при false выполняется обнаружение узлов (discovery) для клиентской балансировки.	`false`
`EnableImplicitSession`	Определяет, включено ли неявное управление сессиями: сервер создает и завершает сессии на каждую операцию; на клиенте пул сессий не используется. **В этом режиме интерактивные клиентские транзакции не поддерживаются.**	`false`

Параметры конструктора соединений

Существуют также дополнительные параметры, которые не участвуют в формировании строки `ConnectionString`. Их можно указать только с использованием `YdbConnectionStringBuilder`:

Параметр	Описание	Значение по умолчанию
`LoggerFactory`	Этот параметр принимает экземпляр, реализующий интерфейс ILoggerFactory. `ILoggerFactory` — это стандартный интерфейс для фабрик логирования в .NET. Допускается использование популярных фреймворков, таких как NLog, Serilog, log4net.	`NullLoggerFactory.Instance`
`CredentialsProvider`	Поставщик аутентификации, реализующий `Ydb.Sdk.Auth.ICredentialsProvider`. Стандартные способы аутентификации: 1) `Ydb.Sdk.Auth.TokenProvider` — аутентификация токеном для OAuth-подобных токенов. 2) Для Yandex Cloud рассмотрите возможность использования **ydb-dotnet-yc**.	Анонимное подключение
`ServerCertificates`	Указывает пользовательские серверные сертификаты, используемые для проверки TLS/SSL. Это полезно при работе с облачными провайдерами (например, Yandex Cloud), которые используют пользовательские корневые или промежуточные сертификаты, которым по умолчанию не доверяют.	Не определено

Поддерживаемые типы данных в интеграции с ADO.NET и их соответствие

Ниже перечислены встроенные сопоставления при чтении и записи типов CLR в типы YDB.

Таблица сопоставления типов на чтение

Ниже показаны сопоставления, используемые при чтении значений.

- Возвращаемый тип при использовании `YdbCommand.ExecuteScalarAsync()`, `YdbDataReader.GetValue()` и подобных методов.
- Вы можете читать значения также как другие .NET-типы, вызывая `YdbDataReader.GetFieldValue<T>()`. Для некоторых временных типов можно получить их «сырое» (непреобразованное) значение, как оно хранится в базе — через `GetInt32()` или `GetInt64()`.

YDB тип	.NET тип	Non-default .NET типы
`Bool`	`bool`	
`Int8`	`sbyte`	
`Int16`	`short`	
`Int32`	`int`	
`Int64`	`long`	
`Uint8`	`byte`	
`Uint16`	`ushort`	
`Uint32`	`uint`	
`Uint64`	`ulong`	
`Float`	`float`	
`Double`	`double`	
`Decimal (precision, scale)`	`decimal` (см. раздел Decimal)	
`Bytes` (синоним `String`)	`byte[]`	
`Text` (синоним `Utf8`)	`string`	
`Json`	`string`	
`JsonDocument`	`string`	
`Yson`	`byte[]`	
`Uuid`	`Guid`	
`Date`	`DateTime`	`DateOnly`
`Date32`	`DateTime`	`DateOnly`, `int` (`GetInt32()` — «сырой» value)
`Datetime`	`DateTime`	`DateOnly`
`Datetime64`	`DateTime`	`DateOnly`, `long` (`GetInt64()` — «сырой» value)
`Timestamp`	`DateTime`	`DateOnly`
`Timestamp64`	`DateTime`	`DateOnly`, `long` (`GetInt64()` — «сырой» value)
`Interval`	`TimeSpan`	`long` (`GetInt64()` — «сырой» value)
`Interval64`	`TimeSpan`	`long` (`GetInt64()` — «сырой» value)

Decimal

`Decimal (Precision, Scale)` — это параметрический тип данных в YDB, который позволяет явно указать:

- `Precision` — общее количество значащих цифр;
- `Scale` — количество цифр после запятой.

Подробнее — в документации

По умолчанию используется тип Decimal(22, 9). Если нужно задать другие значения Precision и Scale, это можно сделать с помощью кода.

Пример ниже показывает, как записать в базу данных значение 1.5 с типом Decimal и параметрами Precision = 5 и Scale = 3.

```
await new YdbCommand(ydbConnection)
{
    CommandText = $"INSERT INTO {tableName}(Id, Decimal) VALUES (1, @Decimal);",
    Parameters = new YdbParameter { Name = "Decimal", Value = 1.5m, Precision = 5, Scale = 3 }
}.ExecuteNonQueryAsync();
```

Таблица сопоставления типов на запись

Существует три правила, определяющие тип параметра в YDB:

1. Если для параметра YdbDbType задано значение, оно используется.
2. Если для параметра задан DbType, он используется.
3. Если ничего из вышеперечисленного не задано, тип серверной части будет определен на основе типа значения (CLR).

YDB тип	.NET тип	Non-default .NET типы	YdbDbType	DbType
`Bool`	`bool`		`Bool`	`Boolean`
`Int8`	`sbyte`		`Int8`	`SByte`
`Int16`	`short`	`sbyte`, `byte`	`Int16`	`Int16`
`Int32`	`int`	`sbyte`, `byte`, `short`, `ushort`	`Int32`	`Int32`
`Int64`	`long`	`sbyte`, `byte`, `short`, `ushort`, `int`, `uint`	`Int64`	`Int64`
`Uint8`	`byte`		`Uint8`	`Byte`
`Uint16`	`ushort`	`byte`	`Uint16`	`UInt16`
`Uint32`	`uint`	`byte`, `ushort`	`Uint32`	`UInt32`
`Uint64`	`ulong`	`byte`, `ushort`, `uint`	`Uint64`	`UInt64`
`Float`	`float`		`Float`	`Single`
`Double`	`double`	`float`	`Double`	`Double`
`Decimal (precision, scale)`	`decimal`		`Decimal`	`Decimal`, `Currency`
`Bytes (синоним String)`	`byte[]`		`Bytes`	`Binary`
`Text (синоним Utf8)`	`string`		`Text`	`String`, `AnsiString`, `AnsiStringFixedLength`, `StringFixedLength`
`Json`	`string`		`Json`	
`JsonDocument`	`string`		`JsonDocument`	
`Yson`	`byte[]`		`Yson`	
`Uuid`	`Guid`		`Uuid`	`Guid`
`Date`	`DateOnly`	`DateTime`	`Date`	`Date`
`Date32`		`DateTime`, `DateOnly`, `int` («сырой» value)	`Date32`	
`Datetime`		`DateTime`, `DateOnly`	`Datetime`	`Datetime`
`Datetime64`		`DateTime`, `long` («сырой» value)	`Datetime64`	
`Timestamp`	`DateTime`		`Timestamp`	`DateTime2`
`Timestamp64`		`DateTime`, `long` («сырой» value)	`Timestamp64`	
`Interval`	`TimeSpan`	`long` («сырой» value)	`Interval`	
`Interval64`		`TimeSpan`, `long` («сырой» value)	`Interval64`	
`List`	`T[]`, `List<T>`	`T[]`, `List<T>`	`List` | `YdbDbType`	

Примечание

При использовании List указывайте тип как побитовое **OR** YdbDbType.List с типом элемента. Например, чтобы задать YdbDbType для List, используйте `YdbDbType.List | YdbDbType.Int32`.

Подключение ADO.NET к Yandex Cloud

Установка

Чтобы использовать аутентификацию Yandex Cloud в вашем .NET-приложении, установите `Ydb.Sdk.Yc.Auth` NuGet-пакет:

```
dotnet add package Ydb.Sdk.Yc.Auth
```

Этот пакет предоставляет необходимые инструменты для аутентификации в сервисах Yandex Cloud.

Аутентификация

Поддерживаемые Yandex Cloud методы аутентификации:

- `Ydb.Sdk.Yc.ServiceAccountProvider` — аутентификация через сервисный аккаунт. Пример использования:

  ```
  var saProvider = new ServiceAccountProvider(
      saFilePath: file, // Path to file with service account JSON info
      loggerFactory: loggerFactory
  );
  ```

- `Ydb.Sdk.Yc.MetadataProvider` — аутентификация через сервис метаданных; используется внутри облачных виртуальных машин Яндекса и облачных функций. Пример использования:

  ```
  var metadataProvider = new MetadataProvider(loggerFactory: loggerFactory);
  ```

Сертификаты

Библиотека включает в себя сертификаты Yandex Cloud, которые необходимы для подключения к Dedicated YDB:

```
var certs = Ydb.Sdk.Yc.YcCerts.GetYcServerCertificates();
```

Как подключиться с ADO.NET

Чтобы установить безопасное соединение с YDB с помощью ADO.NET с требуемыми параметрами аутентификации и TLS, используйте следующий пример:

```
var builder = new YdbConnectionStringBuilder
{
    // More settings ...
    UseTls = true,
    Port = 2135,
    CredentialsProvider = saProvider, // For service account
    ServerCertificates = YcCerts.GetYcServerCertificates() // custom certificates Yandex Cloud
};
```

Пример

ADO.NET подключение к Yandex Cloud

JDBC-драйвер для YDB

JDBC-драйвер для YDB обеспечивает взаимодействие между YDB и приложениями, написанными на Java или других языках программирования, байт-код которых выполняется на JVM. Примеры таких языков — Kotlin и Scala.

Загрузка

Скачайте JDBC-драйвер для YDB с страницы Releases на GitHub.

Документация

- Быстрый старт
- Использование JDBC-драйвера с Maven
- Режимы аутентификации
- Свойства JDBC-драйвера
- Сборка JDBC-драйвера для YDB

Быстрый старт

1. Скачайте JDBC-драйвер для YDB.
2. Скопируйте JAR-файл в директорию, указанную в переменной окружения `CLASSPATH`, или загрузите JAR-файл в интегрированной среде разработки (IDE).
3. Установите соединение с YDB. Примеры JDBC URL:
   - Локальный Docker контейнер с анонимной аутентификацией и без TLS:
     `jdbc:ydb:grpc://localhost:2136/local`
   - Удаленный кластер, размещенный на собственном сервере:
     `jdbc:ydb:grpcs://<host>:2135/Root/<testdb>?secureConnectionCertificate=~/<myca>.cer`
   - Экземпляр облачной базы данных с токеном:
     `jdbc:ydb:grpcs://<host>:2135/<path/to/database>?tokenFile=~/my_token`
   - Экземпляр облачной базы данных с файлом сервисного аккаунта:
     `jdbc:ydb:grpcs://<host>:2135/<path/to/database>?saKeyFile=~/sa_key.json`
4. Выполните проверочный запрос к базе данных YDB. См. пример YdbDriverExampleTest.java.

Использование JDBC-драйвера с Maven

Рекомендованный способ использования JDBC-драйвера для YDB в проекте — это добавить драйвер как зависимость в Maven. Укажите JDBC-драйвер для YDB в секции `dependencies` файла `pom.xml`:

```
<dependencies>
    <dependency>
        <groupId>tech.ydb.jdbc</groupId>
        <artifactId>ydb-jdbc-driver</artifactId>
        <version><!-- актуальная версия --></version>
    </dependency>
</dependencies>
```

Режимы аутентификации

JDBC-драйвер для YDB поддерживает следующие режимы аутентификации:

- **Anonymous** — без аутентификации. Этот режим используется, если не указаны имя пользователя и пароль и не настроены другие методы аутентификации.
- **Static credentials** — используется, если указаны имя пользователя и пароль.
- **Access token** — используется, если указано свойство `token` или `tokenFile`. Требуется YDB токен, который можно получить, выполнив команду `ydb auth get-token` в YDB CLI.
- **Metadata** — используется, если в свойстве `useMetadata` указано значение `true`. В этом режиме данные для аутентификации получаются из метаданных виртуальной машины, контейнера serverless или функции serverless, выполненной в облаке.
- **Service account key** — используется, если указано свойство `saKeyFile`. В этом режиме для аутентификации применяется ключ сервисной учётной записи.

Свойства JDBC-драйвера

JDBC-драйвер для YDB поддерживает следующие конфигурационные свойства, которые можно указать в JDBC URL или передать через дополнительные свойства:

- `saKeyFile` — путь к файлу с ключом сервисной учётной записи (Service Account Key).
- `iamEndpoint` — IAM-эндпойнт для аутентификации с помощью ключа сервисной учётной записи (Service Account Key).
- `token` — токен для аутентификации.
- `tokenFile` — путь к файлу с токеном для аутентификации.
- `useMetadata` — использование режима аутентификации **Metadata**. Возможные значения:
  - `true` — использовать режим аутентификации **Metadata**;
  - `false` — не использовать режим аутентификации **Metadata**.

  Значение по умолчанию: `false`.
- `metadataURL` — эндпойнт для получения токена в режиме аутентификации **Metadata**.
- `localDatacenter` — название локального датацентра, в котором выполняется приложение.
- `secureConnectionCertificate` — путь к файлу с сертификатом CA для TLS-соединения.

> **Примечание**
>
> Значения свойства `saKeyFile`, `tokenFile` или `secureConnectionCertificate` могут быть как абсолютными от корня файловой системы, так и относительными от домашней директории пользователя. Примеры:
>
> - `saKeyFile=~/mysakey1.json`
> - `tokenFile=/opt/secret/token-file`
> - `secureConnectionCertificate=/etc/ssl/cacert.cer`

Примеры JDBC URL

- Локальный Docker контейнер с анонимной аутентификацией и без TLS:
  `jdbc:ydb:grpc://localhost:2136/local`
- Удаленный кластер, размещенный на собственном сервере:
  `jdbc:ydb:grpcs://<host>:2135/Root/<testdb>?secureConnectionCertificate=~/<myca>.cer`
- Экземпляр облачной базы данных с токеном:
  `jdbc:ydb:grpcs://<host>:2135/<path/to/database>?tokenFile=~/my_token`
- Экземпляр облачной базы данных с файлом сервисного аккаунта:
  `jdbc:ydb:grpcs://<host>:2135/<path/to/database>?saKeyFile=~/sa_key.json`

Сборка JDBC-драйвера для YDB

Для запуска всех тестов проекта используется команда `mvn test`.

По умолчанию все тесты выполняются на локальном экземпляре YDB в Docker (при условии, что на хосте установлен Docker или Docker Machine).

Чтобы отключить эти тесты, выполните команду: `mvn test -DYDB_DISABLE_INTEGRATION_TESTS=true`.

YDB Model Context Protocol Server

YDB Model Context Protocol (MCP) server позволяет работать с базами данных YDB из любой большой языковой модели (LLM), которая поддерживает MCP, используя любой из MCP клиентов. Эта интеграция обеспечивает работу с базами данных YDB с помощью ИИ на естественном языке.

Начало работы

Предварительные требования

1. Установите MCP-клиент, поддерживающий MCP tools (большинство поддерживают). Примеры конфигурации ниже используют распространённый формат, поддерживаемый несколькими популярными MCP-клиентами (Claude Desktop, Cursor и др.), но вам может потребоваться адаптировать формат под требования вашего клиента.
2. MCP-сервер YDB — это приложение на Python, которое обычно размещается вместе с MCP-клиентом. Существует несколько вариантов установки и запуска MCP-сервера YDB, которые описаны ниже, но все они требуют предварительно установленного окружения Python 3.10+.

Настройка MCP-клиента

Анонимная аутентификация

uvx

uvx позволяет запускать приложения Python без явной установки.

Настройте YDB MCP в настройках вашего MCP-клиента:

```
{
  "mcpServers": {
    "ydb": {
      "command": "uvx",
      "args": [
        "ydb-mcp",
        "--ydb-endpoint", "grpc://localhost:2136/local"
      ]
    }
  }
}
```

pipx

pipx позволяет запускать приложения из PyPI без явной установки (сам pipx должен быть установлен заранее).

Настройте YDB MCP в настройках вашего MCP-клиента:

```
{
  "mcpServers": {
    "ydb": {
      "command": "pipx",
      "args": [
        "run", "ydb-mcp",
        "--ydb-endpoint", "grpc://localhost:2136/local"
      ]
    }
  }
}
```

pip

При необходимости создайте и активируйте виртуальное окружение Python. Установите YDB MCP с помощью pip:

```
pip install ydb-mcp
```

В настройках вашего MCP-клиента укажите параметры для соединения с YDB MCP:

```
{
  "mcpServers": {
    "ydb": {
      "command": "python3",
      "args": [
        "-m", "ydb_mcp",
        "--ydb-endpoint", "grpc://localhost:2136/local"
      ]
    }
  }
}
```

Аутентификация по логину и паролю

uvx

Настройте аутентификацию по логину/паролю с `uvx`:

```
{
  "mcpServers": {
    "ydb": {
      "command": "uvx",
      "args": [
```

```
        "ydb-mcp",
        "--ydb-endpoint", "grpc://localhost:2136/local",
        "--ydb-auth-mode", "login-password",
        "--ydb-login", "<ваше-имя-пользователя>",
        "--ydb-password", "<ваш-пароль>"
      ]
    }
  }
}
```

pipx

Настройте аутентификацию по логину/паролю с `pipx`:

```
{
  "mcpServers": {
    "ydb": {
      "command": "pipx",
      "args": [
        "run", "ydb-mcp",
        "--ydb-endpoint", "grpc://localhost:2136/local",
        "--ydb-auth-mode", "login-password",
        "--ydb-login", "<ваше-имя-пользователя>",
        "--ydb-password", "<ваш-пароль>"
      ]
    }
  }
}
```

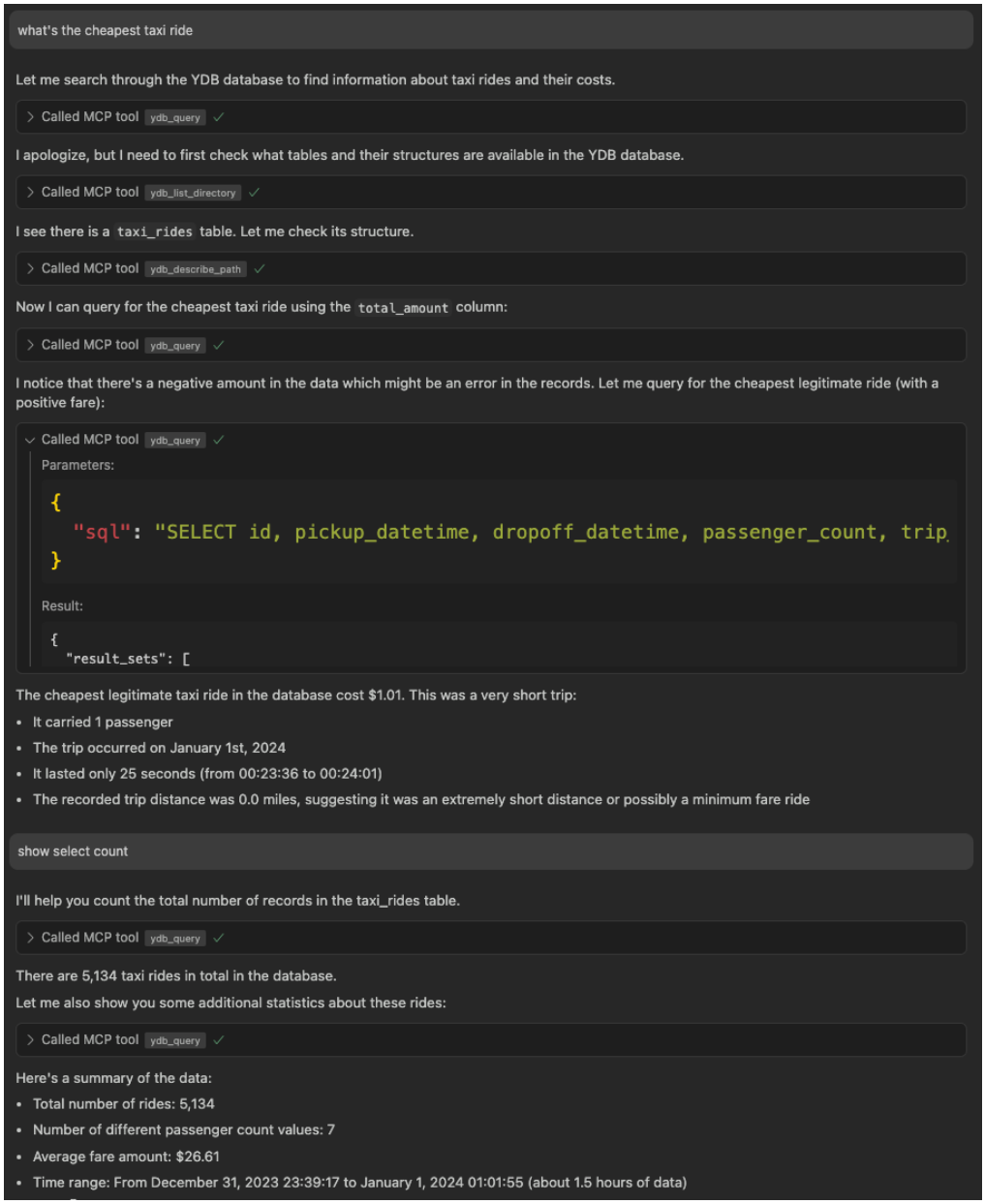
pip

Настройте аутентификацию по логину/паролю с установленным через `pip` YDB MCP:

```
{
  "mcpServers": {
    "ydb": {
      "command": "python3",
      "args": [
        "-m", "ydb_mcp",
        "--ydb-endpoint", "grpc://localhost:2136/local",
        "--ydb-auth-mode", "login-password",
        "--ydb-login", "<ваше-имя-пользователя>",
        "--ydb-password", "<ваш-пароль>"
      ]
    }
  }
}
```

Выполнение запросов

Задавайте вашему LLM вопросы относительно данных, хранящихся в YDB, используя настроенного выше клиента MCP. Языковая модель увидит инструменты, доступные ей через MCP, и будет использовать их для выполнения запросов на YQL и других вызовов к API YDB. Пример того, как это может выглядеть:



Доступные инструменты

MCP-сервер YDB предоставляет следующие инструменты для взаимодействия с базами данных YDB:

- `ydb_query`: Выполнение SQL-запроса к базе данных YDB
  - Параметры:
    - `sql`: Строка SQL-запроса для выполнения
- `ydb_query_with_params`: Выполнение параметризованного SQL-запроса с JSON-параметрами
  - Параметры:
    - `sql`: Строка SQL-запроса с параметрами
    - `params`: JSON-строка, содержащая значения параметров
- `ydb_list_directory`: Просмотр содержимого директории в YDB
  - Параметры:
    - `path`: Путь к директории YDB для просмотра
- `ydb_describe_path`: Получение подробной информации о схемном объекте (таблица, директория и т.д.) по указанному пути YDB
  - Параметры:
    - `path`: Путь YDB для описания
- `ydb_status`: Получение текущего статуса подключения к YDB

Аргументы командной строки и переменные окружения

Следующая таблица описывает параметры командной строки и переменные окружения MCP-сервера YDB:

Аргумент	Переменная окружения	Значение по умолчанию	Описание
`--ydb-endpoint`	`YDB_ENDPOINT`	—	Строка подключения к YDB, включающая протокол, имя хоста, порт и имя базы данных
`--ydb-login`	`YDB_LOGIN`	—	Логин YDB

`--ydb-password`	`YDB_PASSWORD`	—	Пароль YDB
`--ydb-auth-mode`	`YDB_AUTH_MODE`	`anonymous`	Режим аутентификации YDB. Возможные значения: `anonymous`, `login-password`
`--log-level`	—	`INFO`	Уровень логирования. Возможные значения: `DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`

Примечание

Аргументы командной строки имеют приоритет над соответствующими переменными окружения.

Узнать больше

Для получения дополнительной информации посетите репозиторий YDB MCP на GitHub.

AI-навыки YDB (ydb-ai-skills)

ydb-ai-skills — это набор навыков (skills) для AI-агентов разработки, помогающий при работе с YDB: написании запросов на YQL, проектировании схемы данных и ревью кода приложений на корректность использования YDB SDK. Навыки подключаются к AI-агенту и автоматически активируются в зависимости от контекста запроса.

> **Важно**
>
> Проект находится в стадии активной разработки. Набор навыков и их наполнение расширяются, поэтому состав и поведение могут меняться. Актуальную информацию смотрите в репозитории проекта.

Что внутри

Навык	Назначение
ydb-core	Базовый навык: обзор YDB, подключение, аутентификация, основы схемы, CLI. Устанавливается автоматически.
ydb-table	Аудит и ревью кода приложений на YDB SDK, проектирование схемы, написание YQL.

Поддерживаемые агенты

Навыки распространяются в универсальном формате и устанавливаются в каталог соответствующего агента.

Агент	Каталог в проекте	Глобальный каталог
Claude Code	`.claude/skills/`	`~/.claude/skills/`
Cursor	`.cursor/skills/`	—
Windsurf	`.windsurf/skills/`	—
GitHub Copilot	`.github/skills/`	`~/.copilot/skills/`
Codex CLI	`.agents/skills/`	`~/.codex/skills/`
Roo Code	`.roo/skills/`	—
Gemini CLI	`.gemini/skills/`	`~/.gemini/skills/`
Amp	`.agents/skills/`	`~/.config/agents/skills/`
Kiro	`.kiro/skills/`	—
Trae	`.trae/skills/`	—
Generic	`.agents/skills/`	`~/.agents/skills/`

Установка

```
git clone https://github.com/ydb-platform/ydb-ai-skills.git
cd ydb-ai-skills

# Автоопределение агентов в текущем проекте
./install.sh --detect

# Установка для конкретного агента
./install.sh --agent=claude

# Установить только ydb-table (ydb-core добавится как базовый навык)
./install.sh --agent=claude --skills=ydb-table

# Установить только ydb-table без базового навыка
./install.sh --agent=claude --skills=ydb-table --no-core

# Пробный запуск — показать, что будет сделано, без изменений
./install.sh --agent=claude --dry-run
```

Использование

Отдельных действий для запуска не требуется: агент сам подключает подходящий навык, когда задача относится к YDB. Если агент не поддерживает автоподключение, укажите имя навыка (`ydb-core` или `ydb-table`) в запросе явно.

Ревью кода

Навык `ydb-table` проверяет код приложения на типичные ошибки работы с YDB SDK и указывает нарушенные правила. Для запуска ревью укажите агенту файл или диапазон изменений.

Локальный запуск через Ollama

Навыки можно использовать с агентом, работающим на локальной модели через Ollama, без обращения к облачному API. Команда `ollama launch` доступна начиная с Ollama v0.15.

```
# 1. Загрузить локальную модель (нужны поддержка tool calling и большой контекст)
ollama pull gpt-oss:20b

# 2. Запустить Claude Code на этой модели — навыки активируются как обычно
ollama launch claude --model gpt-oss:20b
```

Для агентных сценариев рекомендуется контекст не меньше 64 000 токенов (настраивается в Ollama).

Дополнительные материалы

- Репозиторий ydb-ai-skills — исходный код, документация по написанию навыков и тестированию.
- YQL - Обзор
- Справка по YDB SDK
- YDB CLI

Kafka API

YDB поддерживает работу с топиками по протоколу Kafka версия 3.4.0.

Документация по SDK содержит следующие разделы:

- Аутентификация
- Чтение и запись
- Kafka Connect
- Ограничения

Kafka API

YDB поддерживает работу с топиками по протоколу Kafka версия 3.4.0.

Документация по SDK содержит следующие разделы:

- Аутентификация
- Чтение и запись
- Kafka Connect
- Ограничения

Аутентификация Kafka API

Включение аутентификации

При самостоятельном развертывании YDB по умолчанию используется анонимная аутентификация , не требующая логина-пароля.

Чтобы включить обязательную аутентификацию, следуйте инструкции в статье Аутентификация.

Аутентификация всегда включена при использовании Kafka API в Yandex Cloud

Механизмы аутентификации

В Kafka API поддержано два механизма SASL аутентификации: `PLAIN` и `SCRAM-SHA-256`, а также mTLS аутентификация.

Аутентификация с помощью PLAIN и SCRAM-SHA-256

Оба механизма могут осуществляться как внутри протокола `TLS`, так и вне, порождая соответственно комбинации:

- `SASL_PLAINTEXT/PLAIN`;
- `SASL_SSL/PLAIN`;
- `SASL_PLAINTEXT/SCRAM-SHA-256`;
- `SASL_SSL/SCRAM-SHA-256`.

Важно

Для использования механизма `SCRAM-SHA-256` при аутентификации существующих пользователей может потребоваться смена пароля.

Для аутентификации необходимы:

- `<user-name>` — имя пользователя. Об управлении пользователями читайте в разделе Аутентификация.
- `<password>` — пароль пользователя. Об управлении пользователями читайте в разделе Аутентификация.
- `<database>` — путь к базе данных, с которой предполагается дальнейшее взаимодействие. (Необходим только для механизма `PLAIN`).

Для механизма `SCRAM-SHA-256` база данных определяется на основе настроек подключения Kafka Connect.
Целевой базой данных считается та база, к которой относится узел базы данных, имеющий указанный `<ydb-endpoint>`.

Из этих параметров формируются следующие переменные, которые вы можете использовать в `sasl.jaas.config` параметре конфигурации клиента Kafka:

- `<sasl.username>` = `<user-name>[@<database>]`
- `<sasl.password>` = `<password>`

Важно

Обратите внимание, логика формирования `<sasl.username>` и `<sasl.password>` в облачных инсталляциях YDB может отличаться от приведенной здесь.

Примеры аутентификации смотрите в Чтение и запись.

Аутентификация устройств по mTLS

Чтобы Kafka-клиент мог пройти аутентификацию устройств по mTLS, нужно выполнить следующие шаги.

Создание серверного и клиентского сертификатов

Для каждого шага ниже представлены примеры команд. Замените *** на ваши значения.

1. Создайте Certificate Authority (CA)

```
openssl genrsa -out ca-key.pem 4096
```

```
openssl req -new -x509 -days 3650 -key ca-key.pem -out ca-cert.pem -subj "/C=***/ST=***/L=***/O=***/CN=MyKafkaRootCA"
```

2. Создайте сертификат для сервера

```
openssl genrsa -out server-key.pem 4096
```

В следующей команде вместо `serverhost.com` также укажите название вашего хоста.

```
openssl req -new -key server-key.pem -out server-cert.csr -subj "/C=***/ST=***/L=***/O=***/CN=serverhost.com"
```

```
cat > server-ext.cnf << EOF
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment
subjectAltName = DNS:serverhost.com
EOF
```

```
openssl x509 -req -in server-cert.csr -CA ca-cert.pem -CAkey ca-key.pem -CAcreateserial -out server-cert.pem
-days 365 -extfile server-ext.cnf
```

3. Создайте клиентский сертификат

```
openssl genrsa -out client-key.pem 4096
```

Замените `clienthost.com` на hostname вашего клиента.

```
openssl req -new -key client-key.pem -out client-cert.csr -subj "/C=***/ST=***/L=***/O=***/CN=clienthost.com"
```

```
cat > client-ext.cnf << EOF
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment
subjectAltName = DNS:clienthost.com
EOF
```

```
openssl x509 -req -in client-cert.csr -CA ca-cert.pem -CAkey ca-key.pem -CAcreateserial -out client-cert.pem
-days 365 -extfile client-ext.cnf
```

4. Добавьте сертификаты в keystore и truststore

Для сервера:

```
openssl pkcs12 -export -in server-cert.pem -inkey server-key.pem -out server.p12 -name kafka-server -CAfile c
a-cert.pem -caname root -password pass:changeit

keytool -importkeystore -deststorepass changeit -destkeystore server.keystore.jks -srckeystore server.p12 -sr
cstoretype PKCS12 -srcstorepass changeit -alias kafka-server

keytool -import -trustcacerts -alias ca -file ca-cert.pem -keystore server.truststore.jks -storepass changeit
-noprompt
```

Для клиента:

```
openssl pkcs12 -export -in client-cert.pem -inkey client-key.pem -out client.p12 -name kafka-client -CAfile c
a-cert.pem -caname root -password pass:changeit

keytool -importkeystore -deststorepass changeit -destkeystore client.keystore.jks -srckeystore client.p12 -sr
cstoretype PKCS12 -srcstorepass changeit -alias kafka-client

keytool -import -trustcacerts -alias ca -file ca-cert.pem -keystore client.truststore.jks -storepass changeit
-noprompt
```

После этих пунктов у вас должны появиться нужные keystore и truststore, а также файлы с сертификатами и ключами.

Конфигурация клиента

Пример для Java SDK

```
props.put("security.protocol", "SSL");
props.put("ssl.truststore.password", "changeit");
props.put("ssl.truststore.location", "/full/path/to/client.truststore.jks");
props.put("ssl.keystore.location", "/full/path/to/client.keystore.jks");
props.put("ssl.keystore.password", "changeit");
props.put("ssl.key.password", "changeit");
props.put("ssl.endpoint.identification.algorithm", "");
```

Пример для Kafka cli

```
security.protocol=SSL
ssl.truststore.password=changeit
ssl.truststore.location=/full/path/to/client.truststore.jks
ssl.keystore.location=/full/path/to/client.keystore.jks
ssl.keystore.password=changeit
ssl.key.password=changeit
ssl.endpoint.identification.algorithm=
```

Конфигурация YDB

Требуется указать нужные поля в конфигурации kafka_proxy_config.

```
kafka_proxy_config:
  enable_kafka_proxy: true
  listening_port: your_port

  mtls_enable: true
  key: "server-key.pem" # укажите правильные пути до файлов
  cert: "server-cert.pem"
```

```
  ca: "ca-cert.pem"
  enable_self_signed_certs: true # разрешаете ли вы самоподписанные сертификаты
```

Также укажите в конфигурации client_certificate_authorization правила, по которым будет проходить аутентификация:

```
client_certificate_authorization:
  client_certificate_definitions:
    - require_same_issuer: true
      subject_terms:
        - short_name: CN
          suffixes:
            - '.myhost.net' # нужно заменить на нужный суффикс
      member_groups:
        - user@cert # заменить на нужную member группу
  request_client_certificate: true
```

Для корректной работы нужно использовать тот же сертификат, что и в настройках grpc, поэтому нужно указать путь до этого же серверного сертификата в конфигурации grpc.

Сейчас настроить конфигурации Kafka и grpc с разными серверными сертификатами или указать серверный сертификат только в настройках kafka_proxy_config при использовании mtls нельзя.

```
grpc_config:
  cert: "/path/to/server-cert.pem"
```

Примеры чтения и записи по Kafka API

В этой статье приведены примеры чтения и записи в топики с использованием Kafka API.

Перед выполнением примеров:

1. Создайте топик.
2. Добавьте читателя.
3. Если у вас включена аутентификация, создайте пользователя.

Начало работы

В Docker

Запустите Docker по этой инструкции. Kafka API будет доступен на 9092 порте.

Примеры работы с Kafka API

Чтение

При чтении отличительной особенностью Kafka API является отсутствие поддержки опции check.crcs. Поэтому в конфигурации читателя всегда нужно указывать параметр: `check.crcs=false`.

Ниже даны примеры чтения по Kafka протоколу для разных приложений, языков программирования и фреймворков подключения без аутентификации.
Примеры того, как настроить аутентификацию, смотри в разделе Примеры с аутентификацией

Консольные утилиты Kafka

> **Примечание**
>
> При использовании консольных утилит Kafka с Java 23 и получении ошибки `java.lang.UnsupportedOperationException: getSubject is supported only if a security manager is allowed`
> , либо запустите команду, используя другую версию Java (как сменить версию Java на macos)
> , либо запустите команду, указав для java флаг `-Djava.security.manager=allow`.
> Например: `KAFKA_OPTS=-Djava.security.manager=allow kafka-topics --boostratp-servers localhost:9092 --list`

```
kafka-console-consumer --bootstrap-server localhost:9092 \
    --topic my-topic  \
    --group my-group \
    --from-beginning \
    --consumer-property check.crcs=false \
    --consumer-property partition.assignment.strategy=org.apache.kafka.clients.consumer.RoundRobinAssignor
```

kcat

```
kcat -C \
  -b <ydb-endpoint> \
  -X check.crcs=false \
  -X partition.assignment.strategy=org.apache.kafka.clients.consumer.RoundRobinAssignor \
  -G <consumer-name> <topic-name>
```

Java

```
String HOST = "<ydb-endpoint>";
String TOPIC = "<topic-name>";
String CONSUMER = "<consumer-name>";

Properties props = new Properties();

props.put("bootstrap.servers", HOST);

props.put("key.deserializer", StringDeserializer.class.getName());
props.put("value.deserializer", StringDeserializer.class.getName());

props.put("check.crcs", false);
props.put("partition.assignment.strategy", RoundRobinAssignor.class.getName());

props.put("group.id", CONSUMER);
Consumer<String, String> consumer = new KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList(new String[] {TOPIC}));

while (true) {
  ConsumerRecords<String, String> records = consumer.poll(10000); // timeout 10 sec
  for (ConsumerRecord<String, String> record : records) {
      System.out.println(record.key() + ":" + record.value());
  }
}
```

Spark

```
public class ExampleReadApp {
  public static void main(String[] args) {
    var conf = new SparkConf().setAppName("my-app").setMaster("local");
```

```
    var context = new SparkContext(conf);

    context.setCheckpointDir("checkpoints");
    SparkSession spark = SparkSession.builder()
            .sparkContext(context)
            .config(conf)
            .appName("Simple Application")
            .getOrCreate();

    Dataset<Row> df = spark
            .read()
            .format("kafka")
            .option("kafka.bootstrap.servers", "localhost:9092")
            .option("subscribe", "flink-demo-input-topic")
            .option("kafka.group.id", "spark-example-app")
            .option("startingOffsets", "earliest")
            .option("kafka." + ConsumerConfig.CHECK_CRCS_CONFIG, "false")
            .load();

    df.foreach((ForeachFunction<Row>) row -> {
        System.out.println(row);
    });
  }
}
```

В примере выше использовался Apache Spark 2.12:3.5.3 с зависимостью на `org.apache.spark:spark-streaming-kafka-0-10_2.12:3.5.3`.

Flink

```
public class YdbKafkaApiReadExample {

    public static void main(String[] args) throws Exception {
        final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment()
                .enableCheckpointing(5000, CheckpointingMode.AT_LEAST_ONCE);

        Configuration config = new Configuration();
        config.set(CheckpointingOptions.CHECKPOINT_STORAGE, "filesystem");
        config.set(CheckpointingOptions.CHECKPOINTS_DIRECTORY, "file:///path/to/your/checkpoints");
        env.configure(config);

        KafkaSource<String> kafkaSource = KafkaSource.<String>builder()
                .setBootstrapServers("localhost:9092")
                .setProperty(ConsumerConfig.CHECK_CRCS_CONFIG, "false")
                .setGroupId("flink-demo-consumer")
                .setTopics("my-topic")
                .setStartingOffsets(OffsetsInitializer.earliest())
                .setBounded(OffsetsInitializer.latest())
                .setValueOnlyDeserializer(new SimpleStringSchema())
                        .build();

        env.fromSource(kafkaSource, WatermarkStrategy.noWatermarks(), "kafka-source").print();

        env.execute("YDB Kafka API example read app");
    }
}
```

В примере выше используется Apache Flink версии 1.20 и flink datastream connector к Kafka.

Запись

Консольные утилиты Kafka

> **Примечание**
>
> При использовании консольных утилит Kafka с Java 23 и получении ошибки `java.lang.UnsupportedOperationException: getSubject is supported only if a security manager is allowed`
> , либо запустите команду, используя другую версию Java (как сменить версию Java на macos)
> , либо запустите команду, указав для java флаг `-Djava.security.manager=allow`.
> Например: `KAFKA_OPTS=-Djava.security.manager=allow kafka-topics --boostratp-servers localhost:9092 --list`

```
kafka-console-producer --broker-list localhost:9092 --topic my-topic
```

kcat

```
echo "test message" | kcat -P \
    -b <ydb-endpoint> \
    -t <topic-name> \
    -k key
```

Java

```
String HOST = "<ydb-endpoint>";
String TOPIC = "<topic-name>";

Properties props = new Properties();
props.put("bootstrap.servers", HOST);
props.put("acks", "all");

props.put("key.serializer", StringSerializer.class.getName());
props.put("key.deserializer", StringDeserializer.class.getName());
props.put("value.serializer", StringSerializer.class.getName());
props.put("value.deserializer", StringDeserializer.class.getName());

props.put("compression.type", "none");

Producer<String, String> producer = new KafkaProducer<>(props);
producer.send(new ProducerRecord<String, String>(TOPIC, "msg-key", "msg-body"));
producer.flush();
producer.close();
```

Spark

```
public class ExampleWriteApp {
public static void main(String[] args) {
    var conf = new SparkConf().setAppName("my-app").setMaster("local");
    var context = new SparkContext(conf);
    context.setCheckpointDir("path/to/dir/with/checkpoints");
    SparkSession spark = SparkSession.builder()
        .sparkContext(context)
          .config(conf)
          .appName("Simple Application")
          .getOrCreate();

    spark
          .createDataset(List.of("spark-1", "spark-2", "spark-3", "spark-4"), Encoders.STRING())
          .write()
          .format("kafka")
          .option("kafka.bootstrap.servers", "localhost:9092")
          .option("topic", "flink-demo-output-topic")
          .option("kafka.group.id", "spark-example-app")
          .option("startingOffsets", "earliest")
          .save();
  }
}
```

В примере выше использовался Apache Spark 2.12:3.5.3 с зависимостью на `org.apache.spark:spark-streaming-kafka-0-10_2.12:3.5.3`.

Flink

```
public class YdbKafkaApiProduceExample {
  private static final String TOPIC = "my-topic";

  public static void main(String[] args) throws Exception {
      final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

      Sink<String> kafkaSink = KafkaSink.<String>builder()
              .setBootstrapServers("localhost:9092") // assuming ydb is running locally with kafka proxy on 9
092 port
              .setRecordSerializer(KafkaRecordSerializationSchema.builder()
                      .setTopic(TOPIC)
                      .setValueSerializationSchema(new SimpleStringSchema())
                      .setKeySerializationSchema(new SimpleStringSchema())
                      .build())
              .setRecordSerializer((el, ctx, ts) -> new ProducerRecord<>(TOPIC, el.getBytes()))
              .setDeliveryGuarantee(DeliveryGuarantee.AT_LEAST_ONCE)
                      .build();

      env.setParallelism(1)
              .fromSequence(0, 10)
              .map(i -> i + "")
              .sinkTo(kafkaSink);

      // Execute program, beginning computation.
      env.execute("ydb_kafka_api_write_example");
  }
}
```

В примере выше используется Apache Flink версии 1.20 и flink datastream connector к Kafka.

Logstash

```
  output {
    kafka {
      codec => json
      topic_id => "<topic-name>"
      bootstrap_servers => "<ydb-endpoint>"
```

```
    compression_type => none
  }
}
```

Fluent Bit

```
[OUTPUT]
  name                        kafka
  match                       *
  Brokers                     <ydb-endpoint>
  Topics                      <topic-name>
  rdkafka.client.id           Fluent-bit
  rdkafka.request.required.acks 1
  rdkafka.log_level           7
  rdkafka.sasl.mechanism      PLAIN
```

Примеры с аутентификацией

Подробнее про аутентификацию смотри в разделе Аутентификация. Ниже есть примеры аутентификации в облачной базе и в локальной базе.

> **Примечание**
>
> Сейчас единственным доступным механизмом аутентификации с Kafka API в YDB Topics является `SASL_PLAIN`.

Примеры аутентификации в самостоятельно развернутом YDB

Для того, чтобы проверить работу с аутентификацией в локальной базе:

1. Создайте пользователя. Как это сделать в YQL. Как выполнить YQL из CLI.
2. Подключитесь к Kafka API, как в примерах ниже. Во всех примерах предполагается, что:

- YDB запущен локально с переменной окружения YDB_KAFKA_PROXY_PORT=9092 - то есть Kafka API доступен по адресу localhost:9092. Например можно поднять YDB в докере, как указано здесь.
- - это имя пользователя, которое вы указали при создании пользователя.
- - это пароль пользователя, который вы указали при создании пользователя.

Примеры показаны для чтения, но те же самые параметры конфигурации работают и для записи в топик.

Консольные утилиты Kafka

> **Примечание**
>
> При использовании консольных утилит Kafka с Java 23 и получении ошибки `java.lang.UnsupportedOperationException: getSubject is supported only if a security manager is allowed`
> , либо запустите команду, используя другую версию Java (как сменить версию Java на macos)
> , либо запустите команду, указав для java флаг `-Djava.security.manager=allow`.
> Например: `KAFKA_OPTS=-Djava.security.manager=allow kafka-topics --boostratp-servers localhost:9092 --list`

```
kafka-console-consumer --bootstrap-server localhost:9092 \
--topic <topic-name>  \
--group <consumer-name> \
--from-beginning \
--consumer-property check.crcs=false \
--consumer-property partition.assignment.strategy=org.apache.kafka.clients.consumer.RoundRobinAssignor \
--consumer-property security.protocol=SASL_PLAINTEXT \
--consumer-property sasl.mechanism=PLAIN \
--consumer-property "sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required userna
me=\"<username>\" password=\"<password>\";"
```

kcat

```
kcat -C \
  -b localhost:9092 \
  -X security.protocol=SASL_PLAINTEXT \
  -X sasl.mechanism=PLAIN \
  -X sasl.username="<username>" \
  -X sasl.password="<password>" \
  -X check.crcs=false \
  -X partition.assignment.strategy=roundrobin \
  -G <consumer-name> <topic-name>
```

Java

```
String TOPIC = "<topic-name>";
String CONSUMER = "<consumer-name>";

Properties props = new Properties();

props.put("bootstrap.servers", "localhost:9092");
```

```
props.put("key.deserializer", StringDeserializer.class.getName());
props.put("value.deserializer", StringDeserializer.class.getName());

props.put("check.crcs", false);
props.put("partition.assignment.strategy", RoundRobinAssignor.class.getName());

props.put("security.protocol", "SASL_PLAINTEXT");
props.put("sasl.mechanism", "PLAIN");
props.put("sasl.jaas.config", "sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule requi
red username=\"<username>\" password=\"<password>\";");

props.put("group.id", CONSUMER);
Consumer<String, String> consumer = new KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList(new String[] {TOPIC}));

while (true) {
  ConsumerRecords<String, String> records = consumer.poll(10000); // timeout 10 sec
  for (ConsumerRecord<String, String> record : records) {
      System.out.println(record.key() + ":" + record.value());
  }
}
```

Использование Kafka Connect

В разделе приведены примеры настроек популярных коннекторов Kafka Connect для работы с YDB.

Инструмент Kafka Connect предназначен для перемещения данных между Apache Kafka® и другими хранилищами данных. YDB поддерживает работу с топиками по протоколу Kafka, поэтому можно использовать коннекторы Kafka Connect для работы с YDB.

Подробную информацию о Kafka Connect и его настройке см. в документации Apache Kafka®.

Настройка Kafka Connect. Пошаговая инструкция

В разделе приведена пошаговая инструкция по настройке коннектора Kafka Connect для копирования данных из топика YDB в файл.

В инструкции используются:

- `<topic-name>` — имя топика. Допускается указывать как полное имя (вместе с путем базы данных), так и только имя топика.
- `<sasl.mechanism>` - название механизма SASL. Поддерживаются механизмы `PLAIN` и `SCRAM-SHA-256`.
- `<sasl.username>` — имя пользователя SASL. Подробности читайте в разделе Аутентификация.
- `<sasl.password>` — пароль пользователя SASL. Подробности читайте в разделе Аутентификация.

1. Создайте читателя с именем `connect-<connector-name>`. Имя коннектора указывается в конфигурационном файле при его настройке в поле `name`.
2. Скачайте и распакуйте архив с Apache Kafka®:

   ```
   wget https://downloads.apache.org/kafka/3.6.1/kafka_2.13-3.6.1.tgz && tar -xvf kafka_2.13-3.6.1.tgz --st
   rip 1 --directory /opt/kafka/
   ```

   В данном примере используется Apache Kafka® версии `3.6.1`.
3. Создайте каталог с настройками процесса-исполнителя:

   ```
   sudo mkdir --parents /etc/kafka-connect-worker
   ```

4. Создайте файл настроек процесса-исполнителя `/etc/kafka-connect-worker/worker.properties`

   ```
   # Main properties
   bootstrap.servers=<ydb-endpoint>

   # AdminAPI properties
   sasl.mechanism=<sasl.mechanism>
   security.protocol=SASL_SSL
   sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="<sasl.userna
   me>" password="<sasl.password>";

   # Producer properties
   producer.sasl.mechanism=<sasl.mechanism>
   producer.security.protocol=SASL_SSL
   producer.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="<sa
   sl.username>" password="<sasl.password>";

   # Consumer properties
   consumer.sasl.mechanism=<sasl.mechanism>
   consumer.security.protocol=SASL_SSL
   consumer.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="<sa
   sl.username>" password="<sasl.password>";

   consumer.check.crcs=false

   # Converter properties
   key.converter=org.apache.kafka.connect.storage.StringConverter
   value.converter=org.apache.kafka.connect.storage.StringConverter
   key.converter.schemas.enable=false
   value.converter.schemas.enable=false

   # Worker properties
   plugin.path=/etc/kafka-connect-worker/plugins
   ```

5. Создайте файл настроек FileSink коннектора `/etc/kafka-connect-worker/file-sink.properties` для переноса данных из топика YDB в файл:

   ```
   name=local-file-sink
   connector.class=FileStreamSink
   tasks.max=1
   file=/etc/kafka-connect-worker/file_to_write.json
   topics=<topic-name>
   ```

   Где:

   - `file` — имя файла, в который коннектор будет писать данные.
   - `topics` — имя топика, из которого коннектор будет читать данные.
6. Запустите Kafka Connect в режиме Standalone:

   ```
   cd ~/opt/kafka/bin/ && \
   sudo ./connect-standalone.sh \
           /etc/kafka-connect-worker/worker.properties \
           /etc/kafka-connect-worker/file-sink.properties
   ```

Примеры настройки коннекторов

В разделе приведены примеры файлов настройки коннекторов Kafka Connect для работы с YDB по протоколу Kafka.

Из файла в YDB

Пример файла настроек FileSource-коннектора `/etc/kafka-connect-worker/file-sink.properties` для переноса данных из файла в топик:

```
name=local-file-source
connector.class=FileStreamSource
tasks.max=1
file=/etc/kafka-connect-worker/file_to_read.json
topic=<topic-name>
```

Из YDB в PostgreSQL

Пример файла настроек JDBCSink коннектора `/etc/kafka-connect-worker/jdbc-sink.properties` для переноса данных из топика в таблицу PostgreSQL. Используется коннектор Kafka Connect JDBC Connector.

```
name=postgresql-sink
connector.class=io.confluent.connect.jdbc.JdbcSinkConnector

connection.url=jdbc:postgresql://<postgresql-host>:<postgresql-port>/<db>
connection.user=<pg-user>
connection.password=<pg-user-pass>

topics=<topic-name>
batch.size=2000
auto.commit.interval.ms=1000

transforms=wrap
transforms.wrap.type=org.apache.kafka.connect.transforms.HoistField$Value
transforms.wrap.field=data

auto.create=true
insert.mode=insert
pk.mode=none
auto.evolve=true
```

Из PostgreSQL в YDB

Пример файла настроек JDBCSource коннектора `/etc/kafka-connect-worker/jdbc-source.properties` для переноса данных из PostgreSQL таблицы в топик. Используется коннектор Kafka Connect JDBC Connector.

```
name=postgresql-source
connector.class=io.confluent.connect.jdbc.JdbcSourceConnector

connection.url=jdbc:postgresql://<postgresql-host>:<postgresql-port>/<db>
connection.user=<pg-user>
connection.password=<pg-user-pass>

mode=bulk
query=SELECT * FROM "<topic-name>";
topic.prefix=<topic-name>
poll.interval.ms=1000
validate.non.null=false
```

Из YDB в S3

Пример файла настроек S3Sink коннектора `/etc/kafka-connect-worker/s3-sink.properties` для переноса данных из топика в S3. Используется коннектор Aiven's S3 Sink Connector for Apache Kafka.

```
name=s3-sink
connector.class=io.aiven.kafka.connect.s3.AivenKafkaConnectS3SinkConnector
topics=<topic-name>
aws.access.key.id=<s3-access-key>
aws.secret.access.key=<s3-secret>
aws.s3.bucket.name=<bucket-name>
aws.s3.endpoint=<s3-endpoint>
format.output.type=json
file.compression.type=none
```

Ограничения Kafka API

Поддержка протокола Kafka версии 3.4.0 осуществляется в ограниченном объеме:

1. Не поддержано сжатие сообщений.
2. Не поддержана операция удаления топика. Для удаления топика используйте YQL или YDB CLI.
3. Не поддержана проверка crc.
4. Не поддержана работа с ACL. Для управления доступом к топикам используйте YQL.
5. Если на топике включено автопартиционирование, то в такой топик нельзя писать или читать из него по протоколу Kafka API.

Amazon SQS API

YDB поддерживает работу с топиками по протоколу Amazon SQS.

С одним топиком работа может вестись одновременно по нескольким протоколам. Например, запись может осуществляться с использованием Topic API, а чтение — с использованием Amazon SQS API, и наоборот.

При создании топика командой `CreateQueue` Amazon SQS API топик создаётся с включённым автопартиционированием: с одной партицией и автоматической возможностью увеличения до 10 активных партиций. Параметры автопартиционирования можно изменить через YQL или YDB CLI.

Чтение по Amazon SQS API

Для чтения по Amazon SQS API используется разделяемый (общий) читатель, который должен быть создан на топике до начала чтения по Amazon SQS-протоколу. Если топик создаётся командой `CreateQueue` Amazon SQS API, автоматически создаётся разделяемый (общий) читатель с именем `ydb-sqs-consumer`.

Сообщения, записанные через Topic API, могут быть сжаты алгоритмами gzip, lzop или zstd. При чтении по Amazon SQS-протоколу сервер не разжимает их, а передаёт в виде base64: читатель должен выполнить base64-декодирование, а затем разжатие.

Сообщения, записанные через Topic API без сжатия, могут содержать бинарные данные. При чтении по Amazon SQS-протоколу сервер также кодирует их в base64 — читатель должен выполнить base64-декодирование.

Чтобы читатель мог определить алгоритм сжатия, вместе с каждым сообщением отдаётся атрибут `BodyEncoding`. Если атрибут `BodyEncoding` отсутствует, разжимать сообщение не требуется. Возможные значения атрибута: `gzip`, `lzop`, `zstd` или `base64`.

Пример сообщения, содержимое которого сжато:

```
{
    "Messages": [
        {
            "MessageId": "D523647F-5E89-560B-B1D0-152201831603",
            "ReceiptHandle": "CAAQAA==",
            "MD5OfBody": "d620a162c499920254054f78eac4feed",
            "Body": "KLUv/QBYaQAAeWRiIHdyaXRlZCAyCg==",
            "Attributes": {
                "SentTimestamp": "1780660726888",
                "BodyEncoding": "zstd"
            }
        }
    ]
}
```

Запись по Amazon SQS API

При записи в топик по Amazon SQS-протоколу сообщения равномерно распределяются по партициям. При этом гарантируется, что все сообщения с одинаковым `MessageGroupId` попадут в одну партицию.

Для очередей с обработкой в порядке First-In-First-Out (FIFO) каждому сообщению нужен идентификатор дедупликации. Передайте его в параметре `MessageDeduplicationId`. Если для очереди включён параметр `ContentBasedDeduplication`, `MessageDeduplicationId` можно не передавать: YDB вычислит идентификатор как SHA-256-хеш тела сообщения без учёта атрибутов. Если не передан явный идентификатор и выключена дедубликация по содержимому, запрос завершится ошибкой.

В течение пятиминутного окна YDB принимает повторный запрос с тем же идентификатором, но не добавляет в очередь новое сообщение. После окончания окна идентификатор можно использовать для нового сообщения.

Есть ограничение на количество сообщений, которые можно записать в партицию FIFO-очереди, — 1000 сообщений в секунду. Если требуется записывать больше сообщений, следует увеличить количество партиций. Ограничение на очередь вычисляется как 1000 сообщений/сек/партиция × количество партиций. Например, чтобы записывать 10 тыс. сообщений в секунду, создайте FIFO-очередь с 10 партициями.

Разделы документации

- Аутентификация
- Примеры работы с топиком через Amazon SQS API
- Создание очереди с помощью YQL
- Разделяемый читатель топика: устройство и ограничения

Amazon SQS API

YDB поддерживает работу с топиками по протоколу Amazon SQS.

С одним топиком работа может вестись одновременно по нескольким протоколам. Например, запись может осуществляться с использованием Topic API, а чтение — с использованием Amazon SQS API, и наоборот.

При создании топика командой `CreateQueue` Amazon SQS API топик создаётся с включённым автопартиционированием: с одной партицией и автоматической возможностью увеличения до 10 активных партиций. Параметры автопартиционирования можно изменить через YQL или YDB CLI.

Чтение по Amazon SQS API

Для чтения по Amazon SQS API используется разделяемый (общий) читатель, который должен быть создан на топике до начала чтения по Amazon SQS-протоколу. Если топик создаётся командой `CreateQueue` Amazon SQS API, автоматически создаётся разделяемый (общий) читатель с именем `ydb-sqs-consumer`.

Сообщения, записанные через Topic API, могут быть сжаты алгоритмами gzip, lzop или zstd. При чтении по Amazon SQS-протоколу сервер не разжимает их, а передаёт в виде base64: читатель должен выполнить base64-декодирование, а затем разжатие.

Сообщения, записанные через Topic API без сжатия, могут содержать бинарные данные. При чтении по Amazon SQS-протоколу сервер также кодирует их в base64 — читатель должен выполнить base64-декодирование.

Чтобы читатель мог определить алгоритм сжатия, вместе с каждым сообщением отдаётся атрибут `BodyEncoding`. Если атрибут `BodyEncoding` отсутствует, разжимать сообщение не требуется. Возможные значения атрибута: `gzip`, `lzop`, `zstd` или `base64`.

Пример сообщения, содержимое которого сжато:

```
{
    "Messages": [
        {
            "MessageId": "D523647F-5E89-560B-B1D0-152201831603",
            "ReceiptHandle": "CAAQAA==",
            "MD5OfBody": "d620a162c4999202540547f78eac4feed",
            "Body": "KLUv/QBYaQAAeWRiIHdyaXRlZCAyCg==",
            "Attributes": {
                "SentTimestamp": "1780660726888",
                "BodyEncoding": "zstd"
            }
        }
    ]
}
```

Запись по Amazon SQS API

При записи в топик по Amazon SQS-протоколу сообщения равномерно распределяются по партициям. При этом гарантируется, что все сообщения с одинаковым `MessageGroupId` попадут в одну партицию.

Для очередей с обработкой в порядке First-In-First-Out (FIFO) каждому сообщению нужен идентификатор дедупликации. Передайте его в параметре `MessageDeduplicationId`. Если для очереди включён параметр `ContentBasedDeduplication`, `MessageDeduplicationId` можно не передавать: YDB вычислит идентификатор как SHA-256-хеш тела сообщения без учёта атрибутов. Если не передан явный идентификатор и выключена дедубликация по содержимому, запрос завершится ошибкой.

В течение пятиминутного окна YDB принимает повторный запрос с тем же идентификатором, но не добавляет в очередь новое сообщение. После окончания окна идентификатор можно использовать для нового сообщения.

Есть ограничение на количество сообщений, которые можно записать в партицию FIFO-очереди, — 1000 сообщений в секунду. Если требуется записывать больше сообщений, следует увеличить количество партиций. Ограничение на очередь вычисляется как 1000 сообщений/сек/партиция × количество партиций. Например, чтобы записывать 10 тыс. сообщений в секунду, создайте FIFO-очередь с 10 партициями.

Разделы документации

- Аутентификация
- Примеры работы с топиком через Amazon SQS API
- Создание очереди с помощью YQL
- Разделяемый читатель топика: устройство и ограничения

Аутентификация

В Open Source-версии YDB работа с топиками по Amazon SQS-протоколу поддерживается только без аутентификации.

Настройка кластера

Чтобы выключить аутентификацию, в разделе `security_config` файла конфигурации кластера установите параметр `enforce_user_token_requirement` в значение `false`:

```
security_config:
  enforce_user_token_requirement: false
```

Примеры работы с топиком через Amazon SQS API

В этой статье приведены примеры работы с топиками с использованием Amazon SQS API с помощью AWS CLI.

Перед выполнением примеров настройте доступ к Amazon SQS API (в Open Source-версии — отключите требование аутентификации, см. раздел про аутентификацию).

Если вы используете AWS CLI, убедитесь, что он не требует настроенных AWS-креденшалов: либо добавьте `--no-sign-request` к командам, либо задайте любые значения `AWS_ACCESS_KEY_ID` / `AWS_SECRET_ACCESS_KEY`.

Формирование endpoint для подключения

Endpoint для доступа к Amazon SQS API формируется следующим образом:

`https://{db-balancer}:{port}/{database}`

Где:

- `db-balancer` — DNS-имя HTTPS-балансировщика, у которого в качестве backend указаны compute-ноды базы данных (или адрес узла/сервиса, где запущен HTTP Proxy);
- `port` — порт, на котором доступен HTTP Proxy;
- `database` — полный путь базы данных, в которой находятся топики.

Этот endpoint указывается в AWS CLI через параметр `--endpoint`.

> **Примечание**
>
> В примерах используется endpoint `https://my_db.balancer.example.com:8443/Root/my_db`. В нём:
>
> - `my_db.balancer.example.com` — DNS-имя балансировщика, по которому доступен Amazon SQS-протокол;
> - `8443` — сетевой порт;
> - `/Root/my_db` — имя базы данных.

Создание топика

Для создания топика выполните команду:

```
ENDPOINT="https://my_db.balancer.example.com:8443/Root/my_db"

aws --endpoint "$ENDPOINT" \
  sqs create-queue --queue-name "my_topic"
```

После выполнения команды будет создан топик с указанным именем и разделяемым (общим) читателем с именем `ydb-sqs-consumer`. Проверить существование топика можно с помощью команды scheme describe YDB CLI:

```
ydb -e grpcs://my_db.balancer.example.com:2135 -d /Root/my_db scheme describe my_topic
```

Для создания FIFO-очереди используйте атрибут `FifoQueue=true`. Для FIFO-очередей рекомендуется заканчивать имя на `.fifo`, чтобы соответствовать соглашению об именовании Amazon SQS:

```
ENDPOINT="https://my_db.balancer.example.com:8443/Root/my_db"

aws --endpoint "$ENDPOINT" \
  sqs create-queue \
  --queue-name "my_topic.fifo" \
  --attributes FifoQueue=true
```

После выполнения команды будет создан топик с именем `my_topic.fifo` и разделяемым (общим) читателем с именем `ydb-sqs-consumer`, у которого включено сохранение порядка сообщений.

Получение списка топиков

Для получения списка топиков, с которыми возможна работа по протоколу Amazon SQS, выполните команду:

```
ENDPOINT="https://my_db.balancer.example.com:8443/Root/my_db"

aws --endpoint "$ENDPOINT" sqs list-queues
```

Получение QueueUrl топика

Чтобы получить `QueueUrl` топика, выполните команду:

```
ENDPOINT="https://my_db.balancer.example.com:8443/Root/my_db"

aws --endpoint "$ENDPOINT" sqs get-queue-url \
  --queue-name "my_topic@my_consumer"
```

Где:

- `{queue_name}` — значение параметра `--queue-name`, имя Amazon SQS-очереди в формате `{topic_name}@{consumer_name}`. Если разделяемый (общий) читатель имеет имя `ydb-sqs-consumer`, достаточно указать только `{topic_name}`;

- `{topic_name}` — имя топика с путём от корня базы данных, в которой он создан, например `production/order` (топик `order` в директории `production`);
- `{consumer_name}` — имя разделяемого (общего) читателя на топике, например `ydb-sqs-consumer`.

В примере выше `my_topic` — это `{topic_name}`, а `my_consumer` — `{consumer_name}`.

Запись в топик и чтение из топика

Для операций чтения и записи AWS CLI использует параметр `--queue-url`. Его можно получить через `get-queue-url`.

Ниже приведён пример:

- получение `QueueUrl`;
- запись сообщения;
- чтение сообщения.

```
ENDPOINT="https://my_db.balancer.example.com:8443/Root/my_db"

# получить QueueUrl
QUEUE_URL="$(aws --endpoint "$ENDPOINT" sqs get-queue-url --queue-name "my_topic" --query 'QueueUrl' --output
text)"

# записать сообщение в топик
aws --endpoint "$ENDPOINT" sqs send-message \
  --queue-url "$QUEUE_URL" \
  --message-body "hello from aws cli"

# прочитать сообщение (long polling)
aws --endpoint "$ENDPOINT" sqs receive-message \
  --queue-url "$QUEUE_URL" \
  --wait-time-seconds 20 \
  --max-number-of-messages 1
```

Параметры конфигурации кластера

Конфигурация кластера задается в YAML-файле, который передается в параметре `--yaml-config` при запуске узлов кластера. В данной статье приведено описание основных разделов конфигурации и ссылки на подробную документацию по каждому разделу.

Каждый раздел конфигурации служит определенной цели в настройке работы кластера YDB, от распределения аппаратных ресурсов до настроек безопасности и функциональных флагов. Конфигурация организована в логические группы, соответствующие различным аспектам управления кластером и его работы.

Разделы конфигурации

Доступны следующие разделы конфигурации, расположенные в алфавитном порядке:

Раздел	Обязателен	Описание
actor_system_config	Да	Распределение CPU-ресурсов по пулам акторной системы
auth_config	Нет	Настройки аутентификации и авторизации
aws_client_config	Нет	Настройки по умолчанию клиента AWS
blob_storage_config	Нет	Конфигурация статической группы кластера для системных таблеток
bridge_config	Нет	Конфигурация режима bridge
client_certificate_authorization	Нет	Аутентификация с помощью клиентских сертификатов
cms_config	Нет	Конфигурация Cluster Management System (CMS)
domains_config	Нет	Конфигурация домена кластера, включая Blob Storage и State Storage
feature_flags	Нет	Функциональные флаги для включения или отключения определённых возможностей YDB
health_check_config	Нет	Пороговые значения и таймауты сервиса Health Check
hive_config	Нет	Конфигурация запуска таблеток
host_configs	Нет	Типовые конфигурации хостов для узлов кластера
hosts	Да	Конфигурация статических узлов кластера
immediate_controls_config	Нет	Конфигурация динамических настроек кластера
kafka_proxy_config	Нет	Конфигурация Kafka Proxy
log_config	Нет	Конфигурация и параметры логирования
memory_controller_config	Нет	Распределение памяти и лимиты для компонентов базы данных
monitoring_config	Нет	Параметры YDB Monitoring
node_broker_config	Нет	Конфигурация стабильных имен узлов
query_service_config	Нет	Конфигурация внешних источников для федеративных запросов
resource_broker_config	Нет	Брокер ресурсов для контроля потребления CPU и памяти
security_config	Нет	Настройки конфигурации безопасности
self_management_config	Нет	Настройки распределённой конфигурации V2 и автоматического управления компонентами кластера
system_tablet_backup_config	Нет	Конфигурация резервного копирования системных таблеток
table_service_config	Нет	Настройки конфигурации выполнения запросов
tli_config	Нет	Параметры диагностики инвалидации блокировок транзакций (TLI)
tls	Нет	Конфигурация TLS для безопасных соединений

Практические рекомендации

Этот раздел документации посвящён полному описанию доступных настроек, а практические рекомендации по тому, что и когда настраивать, можно найти в следующих местах:

- В рамках первоначального развёртывания кластера YDB:
- Ansible
- Kubernetes
- Вручную
- В рамках поиска и устранения неисправностей
- В рамках усиления безопасности

Примеры конфигураций кластеров

Модельные конфигурации кластера для развертывания можно найти в репозитории. Изучите их перед развертыванием кластера.

actor_system_config

Основной потребитель CPU — акторная система. Все акторы, в зависимости от своего типа, выполняются в одном из пулов (параметр `name`). Конфигурирование заключается в распределении ядер процессора узла по пулам акторной системы. При выделении процессорных ядер пулам нужно учитывать, что PDisk и gRPC API работают вне акторной системы и требуют отдельных ресурсов.

Вы можете использовать автоматическое или ручное конфигурирование акторной системы. В секции `actor_system_config` укажите:

- тип узла и количество ядер CPU, выделяемых процессу `ydbd` в случае использования автоматической конфигурирования;
- количество ядер CPU для каждой подсистемы кластера YDB при использовании ручного конфигурирования.

Автоматическое конфигурирование является адаптивным, т.е. подстраивается под текущую нагрузку системы. Его рекомендуется использовать в большинстве случаев.

Ручное конфигурирование может быть полезно в случае, когда какой-либо пул акторной системы в процессе работы оказывается перегружен и это влияет на общую производительность БД. Отслеживать нагрузку пулов можно на странице мониторинга в YDB UI.

Автоматическое конфигурирование

Независимо от значения параметра `node_type`, при возникновении голодания (дефицита вычислительных ресурсов) в одном или нескольких пулах применяется следующий приоритет перераспределения ядер (в порядке убывания):

- `IC`
- `System`
- `User`
- `Batch`

Например, если `IC` и `User` голодают, а в `System` имеются свободные ядра, то ядра получит `IC`.

Пример секции `actor_system_config` для автоматического конфигурирования акторной системы:

```
actor_system_config:
  use_auto_config: true
  node_type: STORAGE
  cpu_count: 10
```

Параметр	Описание
`use_auto_config`	Включение автоматического конфигурирования акторной системы.
`node_type`	Тип узла. Определяет ожидаемую нагрузку и соотношение ядер CPU между пулами. Одно из значений: • `STORAGE` — узел работает с блочными устройствами и отвечает за Distributed Storage; • `COMPUTE` — узел обслуживает пользовательскую нагрузку; • `HYBRID` — узел работает со смешанной нагрузкой или потребление `System`, `User` и `IC` узла под нагрузкой приблизительно одинаково.
`cpu_count`	Количество ядер CPU, выделенных узлу.

Ручное конфигурирование

Пример секции `actor_system_config` для ручного конфигурирования акторной системы:

```
actor_system_config:
  executor:
  - name: System
    spin_threshold: 0
    threads: 2
    type: BASIC
  - name: User
    spin_threshold: 0
    threads: 3
    type: BASIC
  - name: Batch
    spin_threshold: 0
    threads: 2
    type: BASIC
  - name: IO
    threads: 1
    time_per_mailbox_micro_secs: 100
    type: IO
  - name: IC
    spin_threshold: 10
    threads: 1
    time_per_mailbox_micro_secs: 100
    type: BASIC
  scheduler:
    progress_threshold: 10000
    resolution: 256
    spin_threshold: 0
```

Параметр	Описание

`executor`	Конфигурация пулов. В конфигах пулов рекомендуется менять только количество ядер процессора (параметр `threads`).
`name`	Имя пула, определяет его назначение. Одно из значений: • `System` — предназначен для выполнения быстрых внутренних операций YDB (обслуживает системные таблетки, State Storage, ввод и вывод Distributed Storage, Erasure Coding); • `User` — обслуживает пользовательскую нагрузку (пользовательские таблетки, выполнение запросов в Query Processor); • `Batch` — обслуживает задачи, которые не имеют строгого лимита на время выполнения, фоновых операции (сборка мусора, тяжелые запросы Query Processor); • `IO` — отвечает за выполнение всех задач с блокирующими операциями (аутентификация, запись логов в файл); • `IC` — Interconnect, включает нагрузку, связанную с коммуникацией между узлами (системные вызовы для ожидания и отправки по сети данных, сериализация данных, разрезание и склеивание сообщений).
`spin_threshold`	Количество тактов процессора перед уходом в сон при отсутствии сообщений. Состояние сна снижает энергопотребление, но может увеличивать latency запросов во время слабой нагрузки.
`threads`	Количество ядер процессора, выделенных пулу. Не рекомендуется суммарно назначать в пулы System, User, Batch, IC больше ядер, чем доступно в системе.
`max_threads`	Максимальное количество ядер процессора, которые могут быть выданы пулу в случае использования простаивающих ядер из других пулов. При выставлении параметра включается механизм увеличения размера пула при полном потреблении пула и наличия свободных ядер. Проверка текущей нагрузки и перераспределение ядер происходит 1 раз в секунду.
`max_avg_ping_deviation`	Дополнительное условие для расширения пула по количеству ядер. При потреблении более чем 90% ядер процессора, выделенных пулу, требуется ухудшение показателя SelfPing более чем на `max_avg_ping_deviation` микросекунд от ожидаемых 10 миллисекунд.
`time_per_mailbox_micro_secs`	Количество сообщений в каждом акторе, которое будет обработано перед переключением на другой актор.
`type`	Тип пула. Одно из значений: • `IO` — укажите для пула IO; • `BASIC` — укажите для всех остальных пулов.
`scheduler`	Конфигурация шедулера. Шедулер акторной системы отвечает за доставку отложенных сообщений между акторами. Не рекомендуется изменять параметры шедулера по умолчанию.
`progress_threshold`	В акторной системе есть возможность запросить отправку сообщения в будущем по расписанию. Возможна ситуация, когда в определенный момент времени системе не удастся отправить все запланированные сообщения. В этом случае система начинает рассылать сообщения в «виртуальном времени», обрабатывая в каждом цикле отправку сообщений за период, не превышающий `progress_threshold` в микросекундах, и продвигая виртуальное время на `progress_threshold`, пока оно не догонит реальное.
`resolution`	При составлении расписания отправки сообщений используются дискретные временные слоты. Длительность слота задается параметром `resolution` в микросекундах.

auth_config

YDB позволяет использовать различные способы аутентификации пользователей в системе. Настройки аутентификации и провайдеров аутентификации задаются в секции `auth_config` файла конфигурации YDB.

Конфигурация аутентификации локальных пользователей YDB

Подробнее об аутентификации локальных пользователей см. в разделе про аутентификацию по логину и паролю. Для настройки аутентификации локальных пользователей по логину и паролю необходимо указать следующие параметры в секции `auth_config`:

Параметр	Описание
`use_login_provider`	Флаг разрешает аутентификацию локальных пользователей по auth-токенам, полученным в результате входа по логину и паролю. Процедура входа в YDB — это обмен логина и пароля на аутентификационный токен. Возможные значения: • `true` — разрешает аутентификацию локальных пользователей по аутентификационным токенам; • `false` — запрещает аутентификацию локальных пользователей по аутентификационным токенам. Значение по умолчанию: `true` **Примечание** Для возможности создания и аутентификации локальных пользователей параметры `use_login_provider` и `enable_login_authentication` должны иметь значение `true`. В противном случае локальные пользователи не смогут аутентифицироваться в YDB.
`enable_login_authentication`	Флаг разрешает создание локальных пользователей и получение для них аутентификационного токена в обмен на логин и пароль. Возможные значения: • `true` — разрешает создание локальных пользователей и получение для них аутентификационного токена; • `false` — запрещает создание локальных пользователей и получение для них аутентификационного токена. Значение по умолчанию: `true`
`domain_login_only`	Флаг определяет границы прав доступа локальных пользователей в кластере YDB. Возможные значения: • `true` — локальные пользователи YDB существуют на уровне кластера и им могут назначаться права на доступ к множеству баз данных. • `false` — локальные пользователи могут существовать как на уровне кластера, так и на уровне каждой отдельной базы данных. Границы прав доступа локальных пользователей, созданных на уровне базы данных, ограничиваются базой данных, в которой они созданы. Значение по умолчанию: `true`
`login_token_expire_time`	Время жизни аутентификационного токена, созданного в обмен на логин и пароль локального пользователя. Значение по умолчанию: `12h`

Конфигурация блокировки пользователя при неправильно введённом пароле

YDB позволяет запретить пользователю аутентифицироваться, если он совершил несколько неудачных попыток ввода пароля. Для настройки условий блокировки пользователя необходимо заполнить секцию `account_lockout`.

Пример секции `account_lockout`:

```
auth_config:
  #...
  account_lockout:
    attempt_threshold: 4
    attempt_reset_duration: "1h"
  #...
```

Параметр	Описание
`attempt_threshold`	Количество неверных попыток ввода пароля, после которых учётная запись пользователя временно блокируется. Если пользователь ввёл неправильный пароль указанное количество раз подряд, ему запрещается аутентифицироваться на время, заданное в параметре `attempt_reset_duration`. Если параметр имеет значение `0`, число попыток ввода неправильного пароля не ограничено. После успешной аутентификации (ввода правильных имени пользователя и пароля), счетчик неуспешных попыток сбрасывается в значение 0. Значение по умолчанию: `4`

`attempt_reset_duration`	Период времени, в течение которого пользователь считается заблокированным. В течение этого периода пользователь не сможет аутентифицироваться в системе даже если введёт правильные имя пользователя и пароль. Период блокировки начинается с момента последней неверной попытки ввода пароля. Если указано нулевое значение (`"0s"` — запись, эквивалентная 0 секунд), пользователь будет заблокирован на неограниченное время. В этом случае снять блокировку можно при помощи команды ALTER USER ... LOGIN. Минимальный интервал времени блокировки 1 секунда. Поддерживаемые единицы измерения: • Секунды. `30s` • Минуты. `20m` • Часы. `5h` • Дни. `3d` Не допускается комбинировать единицы измерения в одной строке. Например такая запись некорректна: `1d12h`. Такую запись нужно заменить на эквивалентную, например `36h`. Значение по умолчанию: `1h`

Конфигурация требований к сложности пароля

YDB позволяет аутентифицировать пользователей по логину и паролю. Подробнее см. в разделе аутентификация по логину и паролю. Для повышения безопасности в YDB предусмотрена возможность настройки сложности используемых паролей локальных пользователей. Для конфигурирования требований к паролю необходимо описать секцию `password_complexity`.

Пример секции `password_complexity`:

```
auth_config:
  #...
  password_complexity:
    min_length: 8
    min_lower_case_count: 1
    min_upper_case_count: 1
    min_numbers_count: 1
    min_special_chars_count: 1
    special_chars: "!@#$%^&*()_+{}|<>?="
    can_contain_username: false
  #...
```

Параметр	Описание
`min_length`	Минимальная длина пароля. Значение по умолчанию: `0` (не ограничено)
`min_lower_case_count`	Минимальное количество строчных букв в пароле. Значение по умолчанию: `0` (не ограничено)
`min_upper_case_count`	Минимальное количество прописных букв в пароле. Значение по умолчанию: `0` (не ограничено)
`min_numbers_count`	Минимальное количество цифр в пароле. Значение по умолчанию: `0` (не ограничено)
`min_special_chars_count`	Минимальное количество специальных символов в пароле из указанных в параметре `special_chars`. Значение по умолчанию: `0` (не ограничено)
`special_chars`	Перечень специальных символов, допустимых при задании пароля. Валидные значения: `!@#$%^&*()_+{}|<>?=` Значение по умолчанию: пустая строка (допускает использование всех валидных специальных символов)
`can_contain_username`	Флаг определяет допустимость включения имени пользователя в пароль. Значение по умолчанию: `false`

Примечание

Любые изменения политики паролей не затрагивают уже действующие пароли пользователей, поэтому изменять существующие пароли не требуется, они будут приниматься в текущем виде.

Конфигурация LDAP аутентификации

Одним из способов аутентификации пользователей в YDB является использование LDAP-каталога. Подробнее о таком виде аутентификации написано в разделе про использование LDAP-каталога. Для конфигурирования LDAP-аутентификации необходимо описать секцию `ldap_authentication`.

Пример секции `ldap_authentication`:

```
auth_config:
  #...
  ldap_authentication:
    hosts:
      - "ldap-hostname-01.example.net"
      - "ldap-hostname-02.example.net"
      - "ldap-hostname-03.example.net"
    port: 389
    base_dn: "dc=mycompany,dc=net"
    bind_dn: "cn=serviceAccaunt,dc=mycompany,dc=net"
    bind_password: "serviceAccauntPassword"
    search_filter: "uid=$username"
    scheme: "ldap"
    requested_group_attribute: "memberOf"
    extended_settings:
      enable_nested_groups_search: true
      enable_sasl_external_bind: true
    use_tls:
      enable: true
      ca_cert_file: "/path/to/ca.pem"
      cert_require: DEMAND
      cert_file: "/path/to/client-cert.pem"
      key_file: "/path/to/client-key.pem"
  ldap_authentication_domain: "ldap"
  refresh_time: "1h"
  #...
```

Параметр	Описание
`hosts`	Список имен хостов, на котором работает LDAP-сервер
`port`	Порт для подключения к LDAP-серверу
`base_dn`	Корень поддерева в LDAP-каталоге, начиная с которого будет производиться поиск записи пользователя
`bind_dn`	Отличительное имя (Distinguished Name, DN) сервисного аккаунта, от имени которого выполняется поиск записи пользователя
`bind_password`	Пароль сервисного аккаунта, от имени которого выполняется поиск записи пользователя. Не задаётся при `extended_settings.enable_sasl_external_bind: true`
`search_filter`	Фильтр для поиска записи пользователя в LDAP-каталоге. В строке фильтра может встречаться последовательность символов *$username*, которая будет заменена на имя пользователя, запрошенное для аутентификации в базе данных
`use_tls`	Настройки для конфигурирования TLS-соединения между YDB и LDAP-сервером
`enable`	Определяет, будет ли произведена попытка установить TLS-соединение с использованием запроса `StartTls`. При установке значения этого параметра в `true`, необходимо отключить использование схемы соединения `ldaps`, присвоив параметру `ldap_authentication.scheme` значение `ldap`
`ca_cert_file`	Путь до файла сертификата удостоверяющего центра
`cert_require`	Уровень требований к сертификату LDAP-сервера. Возможные значения: • `NEVER` - YDB не запрашивает сертификат или проверку проходит любой сертификат. • `ALLOW` - YDB требует, что бы LDAP-сервер предоставил сертификат. Если предоставленному сертификату нельзя доверять, TLS-сессия все равно установится. • `TRY` - YDB требует, что бы LDAP-сервер предоставил сертификат. Если предоставленному сертификату нельзя доверять, установление TLS-соединения прекращается. • `DEMAND` и `HARD` — Эти требования эквивалентны параметру `TRY`. Значение по умолчанию: `DEMAND`
`cert_file`	Путь до файла клиентского сертификата. Используется в качестве аутентификационной информации для сервисного аккаунта.
`key_file`	Путь до файла ключа клиентского сертификата

`scheme`	Схема соединения с LDAP-сервером. Возможные значения: • `ldap` — YDB будет выполнять соединение с LDAP-сервером без какого-либо шифрования. Пароли будут отправляться на LDAP-сервер в открытом виде. • `ldaps` — YDB будет выполнять зашифрованное соединение с LDAP-сервером по протоколу TLS с самого первого запроса. Для успешного установления соединения по схеме `ldaps` необходимо отключить использование запроса `StartTls` в секции `ldap_authentication.use_tls.enable: false` и заполнить информацию о сертификате `ldap_authentication.use_tls.ca_cert_file` и уровне требования сертификата `ldap_authentication.use_tls.cert_require`. • При использовании любого другого значения будет браться значение по умолчанию - `ldap`. Значение по умолчанию: `ldap`
`requested_group_attribute`	Атрибут обратного членства в группе. По умолчанию `memberOf`
`extended_settings.enable_nested_groups_search`	Флаг определяет, будет ли выполнятся запрос для получения всего дерева групп, в которые входят непосредственные группы пользователя. Возможные значения: • `true` — YDB запрашивает информацию о всех группах, в которые входят непосредственные группы пользователя. Запросы о всех родительских группах могут занимать много времени. • `false` — YDB запрашивает плоский список групп пользователя. Такой запрос не получает информацию о возможных вложенных родительских группах. Значение по умолчанию: `false`
`extended_settings.enable_sasl_external_bind`	Флаг определяет, будет ли выполняться аутентификация сервисного аккаунта по протоколу SASL с механизмом EXTERNAL. Возможные значения: • `true` - Для аутентификации сервисного аккаунта будет задействован протокол SASL с механизмом EXTERNAL (аутентификация по клиентскому TLS-сертификату в рамках mTLS). В качестве аутентификационной информации используется клиентский сертификат, указанный в параметрах `use_tls.cert_file`, и `use_tls.key_file`. Параметры `bind_dn` и `bind_password` в таком случае не задаются. • `false` - Для аутентификации сервисного аккаунта будет использоваться метод simple bind. Требуется указать параметры `bind_dn` и `bind_password`. Значение по умолчанию: `false`
`host`	Имя хоста, на котором работает LDAP-сервер. Это устаревший параметр, вместо него должен использоваться параметр `hosts`
`ldap_authentication_domain`	Суффикс имени пользователя, позволяющий отличать пользователей из LDAP-каталога от пользователей аутентифицируемых с помощью других провайдеров. Значение по умолчанию: `ldap`

Конфигурация аутентификации по клиентскому сертификату

YDB поддерживает аутентификацию по клиентскому сертификату. Правила проверки сертификатов задаются в секции client_certificate_authorization. Дополнительно в секции `auth_config` может указываться суффикс имён пользователей, аутентифицированных по сертификату:

Параметр	Описание
`certificate_authentication_domain`	Суффикс имени пользователя, позволяющий отличать пользователей, аутентифицированных по клиентскому сертификату, от пользователей, аутентифицируемых другими способами. Значение по умолчанию: `cert` (то есть суффикс SID по умолчанию — `@cert`)

Конфигурация аутентификации с использованием внешнего IdP

YDB поддерживает аутентификацию по JWT-токенам внешнего провайдера идентификации с использованием OpenID Connect. Для включения аутентификации необходимо добавить секцию `external_idp_config` в `auth_config`.

Пример конфигурации:

```
auth_config:
  #...
  external_idp_config:
    issuer: "https://idp.example.com"
    audience: "ydb-cluster"
    allowed_clock_skew: "30s"
    subject_claim_name: "username"
    groups_claim_name: "groups"
```

```
    discovery_periodic_settings:
      success_refresh_period: "1h"
      min_error_refresh_period: "1s"
      max_error_refresh_period: "5m"
      request_timeout: "15s"
    jwks_periodic_settings:
      success_refresh_period: "30m"
      min_error_refresh_period: "1s"
      max_error_refresh_period: "10s"
      request_timeout: "15s"
    jwks_cache_settings:
      timeout: "2h"
  external_idp_authentication_domain: "sso"
  use_access_service: false
  #...
```

Параметр	Описание
`external_idp_config.issuer`	Ожидаемое значение поля `iss` (issuer), которое идентифицирует издателя JWT-токена, и базовый URL для OIDC Discovery. Обязательный параметр; должен начинаться с `https://` и не должен заканчиваться символом `/` . Значения `issuer` в Discovery-документе и `iss` в JWT должны в точности совпадать с указанным значением. Значение по умолчанию: пустая строка
`external_idp_config.audience`	Ожидаемое значение поля `aud` (audience), которое идентифицирует получателя JWT-токена. Если параметр не задан, получатель токена не проверяется. Значение по умолчанию: пустая строка
`external_idp_config.allowed_clock_skew`	Допустимое расхождение часов при проверке полей JWT, связанных со временем: `exp` задаёт время истечения срока действия токена, `nbf` — время начала его действия, `iat` — время выпуска. Значение по умолчанию: `30s`
`external_idp_config.subject_claim_name`	Имя строкового поля JWT, из которого формируется SID пользователя. Если поле отсутствует или имеет другой тип, используется поле `sub` . Значение по умолчанию: `sub`
`external_idp_config.groups_claim_name`	Имя поля JWT с массивом групп пользователя. Из массива извлекаются только строковые элементы. Значение по умолчанию: `groups`
`external_idp_config.discovery_periodic_settings.success_refresh_period`	Период обновления Discovery-документа после успешного запроса. Значение по умолчанию: `1h`
`external_idp_config.discovery_periodic_settings.min_error_refresh_period`	Минимальный интервал перед повторным запросом Discovery-документа после ошибки. Значение по умолчанию: `1s`
`external_idp_config.discovery_periodic_settings.max_error_refresh_period`	Максимальный интервал перед повторным запросом Discovery-документа после ошибки. Значение по умолчанию: `5m`
`external_idp_config.discovery_periodic_settings.request_timeout`	Тайм-аут запроса Discovery-документа. Значение по умолчанию: `15s`
`external_idp_config.jwks_periodic_settings.success_refresh_period`	Период обновления JWKS после успешного запроса. Значение по умолчанию: `1h`
`external_idp_config.jwks_periodic_settings.min_error_refresh_period`	Минимальный интервал перед повторным запросом JWKS после ошибки. Значение по умолчанию: `1s`
`external_idp_config.jwks_periodic_settings.max_error_refresh_period`	Максимальный интервал перед повторным запросом JWKS после ошибки. Значение по умолчанию: `5m`
`external_idp_config.jwks_periodic_settings.request_timeout`	Тайм-аут запроса JWKS. Значение по умолчанию: `15s`

`external_idp_config.jwks_cache_settings.timeout`	Максимальный возраст кеша JWKS. Если обновить JWKS не удалось, после истечения этого периода ключи удаляются. Значение по умолчанию: `2h`
`external_idp_authentication_domain`	Суффикс имени пользователя, позволяющий отличать пользователей внешнего IdP от пользователей, аутентифицируемых с помощью других провайдеров. Такой же суффикс добавляется к именам групп, полученным из JWT-токена. Значение по умолчанию: `sso`

Важно

Сторонний IAM-провайдер и внешний IdP по протоколу OIDC используют токены типа `Bearer`. Если параметр `use_access_service` включён, IAM-провайдер имеет приоритет и перехватывает все такие токены. Поэтому одновременное использование аутентификации через IAM-провайдер и внешний IdP по протоколу OIDC не поддерживается.

Конфигурация аутентификации с использованием стороннего IAM-провайдера

YDB поддерживает аутентификацию пользователей с использованием сервиса Yandex Identity and Access Management (IAM), который используется в Yandex Cloud, или другого сервиса, совместимого с ним по API. Для конфигурирования IAM-аутентификации необходимо определить следующие параметры:

Параметр	Описание
`use_access_service`	Флаг разрешает аутентификацию пользователей в Yandex Cloud через IAM с использованием AccessService. Значение по умолчанию: `false`
`access_service_endpoint`	Адрес, по которому отправляются запросы в AccessService (IAM). Значение по умолчанию: `as.private-api.cloud.yandex.net:4286`
`use_access_service_tls`	Флаг включает использование TLS-соединений между YDB и AccessService. Значение по умолчанию: `true`
`access_service_domain`	Суффикс «источника пользователя» в SID для пользователей, приходящих в YDB из Yandex Cloud IAM. Значение по умолчанию: `as` ("access service")
`path_to_root_ca`	Путь до файла сертификата удостоверяющего центра, используемого для взаимодействия с AccessService. Значение по умолчанию: `/etc/ssl/certs/YandexInternalRootCA.pem`
`access_service_grpc_keep_alive_time_ms`	Период времени, в миллисекундах, по истечении которого YDB посылает keepalive ping IAM-серверу, чтобы сохранить соединение. Значение по умолчанию: `10000`
`access_service_grpc_keep_alive_timeout_ms`	Период времени ожидания ответа от IAM-сервера на keepalive ping, в миллисекундах. Если по истечении срока ожидания ответ от IAM-сервера не приходит, YDB закрывает соединение. Значение по умолчанию: `1000`
`use_access_service_api_key`	Флаг разрешает использование API-ключей IAM. API-ключ — это секретный ключ, выписываемый в Yandex Cloud IAM для упрощённой авторизации сервисных аккаунтов в API Yandex Cloud. Его применяют, если нет возможности автоматически запрашивать IAM-токен. Значение по умолчанию: `false`

Настройки кеширования результатов аутентификации

Для сокращения количества проверок аутентификационных токенов каждый узел YDB кеширует результаты проверки в токенах пользователей. Подробнее см. в статье Кеширование результатов аутентификации.

Время жизни и другие аспекты работы токена пользователя настраиваются с помощью следующих параметров. Значения временных параметров задаются числом с суффиксом единицы измерения: `ms` — миллисекунды, `s` — секунды, `m` — минуты, `h` — часы, `d` — дни. Например, `300ms`, `30s`, `10m`, `1h` или `2d`.

`refresh_period`	Определяет как часто узел YDB сканирует токены пользователей в кэше на достижение временных лимитов, указанных в параметрах `refresh_time`, `life_time` и `expire_time`, после которых требуется обновление или удаление токена. Чем короче указанный интервал проверки токенов пользователей, тем выше нагрузка на CPU. Значение по умолчанию: `1s`

`refresh_time`	Максимальный интервал между успешным обновлением токена пользователя и следующей попыткой обновления. Конкретное время обновления выбирается в диапазоне от `refresh_time/2` до `refresh_time`. Например, после первого запроса с корректным аутентификационным токеном узел создаёт токен пользователя. Через интервал, выбранный случайно в диапазоне от `refresh_time/2` до `refresh_time`, узел снова проверяет аутентификационный токен. После успешной проверки цикл повторяется. При ретрабельной ошибке узел повторяет проверку с учётом параметров `min_error_refresh_time` и `max_error_refresh_time`, а при постоянной ошибке перестаёт использовать ранее созданный токен пользователя. Параметр применяется к обновляемым аутентификационным токенам, например к токенам входа по логину и паролю и токенам внешнего провайдера идентификации. Значение по умолчанию: `1h`
`life_time`	Период хранения токена пользователя в кэше узла YDB с момента его последнего использования. Если запросы от пользователя, для которого создан токен, не приходили на узел YDB в течение указанного периода, узел удаляет этот токен пользователя из своего кэша. Значение по умолчанию: `1h`
`expire_time`	Период действия результата успешной проверки для большинства типов аутентификационных токенов. После успешного обновления отсчёт начинается заново. По истечении периода запись удаляется из кеша независимо от `life_time`. Для токенов входа по логину и паролю и токенов внешнего провайдера идентификации используется срок действия самого аутентификационного токена. Для запросов с подписью ключом доступа применяется отдельный параметр `as_signature_expire_time`. Значение по умолчанию: `24h`
`as_signature_expire_time`	Период действия результата проверки запроса, аутентифицированного с помощью подписи ключом доступа. Значение по умолчанию: `1m`
`min_error_refresh_time`	Начальный интервал между повторными проверками после ретрабельной ошибки обновления токена пользователя. После ретрабельной ошибки первая повторная проверка выполняется немедленно. Если она также завершается ретрабельной ошибкой, задержка перед следующей проверкой выбирается случайным образом в диапазоне от `min_error_refresh_time/2` до `min_error_refresh_time`. После каждой следующей ретрабельной ошибки текущий интервал удваивается, но не превышает разность `max_error_refresh_time - min_error_refresh_time`. Для каждого текущего интервала `D` фактическая задержка выбирается случайным образом в диапазоне от `D/2` до `D`. **Важно** Не рекомендуется выставлять значение параметра в `0`, так как немедленные повторы создают избыточную нагрузку. Значение по умолчанию: `1s`
`max_error_refresh_time`	Ограничивает увеличение интервала между повторными проверками после ретрабельных ошибок обновления токена пользователя. Не ограничивает общую продолжительность повторных проверок. Значение по умолчанию: `1m`

Пример для аутентификации по логину и паролю:

```
auth_config:
  refresh_period: "1s"
  refresh_time: "1h"
  life_time: "2h"
  expire_time: "6h"
  login_token_expire_time: "12h"
  min_error_refresh_time: "1s"
  max_error_refresh_time: "1m"
```

После создания токена пользователя следующая плановая проверка выполняется через интервал, случайно выбранный в диапазоне от `30m` до `1h` (`refresh_time`). Выполнение этого условия проверяется раз в `1s` (`refresh_period`). После успешной проверки интервал выбирается заново. При возникновении ретрабельной ошибки первая повторная проверка выполняется немедленно. Если она также завершается ретрабельной ошибкой, следующая проверка выполняется через интервал, случайно выбранный в диапазоне от `500ms` до `1s` (`min_error_refresh_time`). Затем текущий интервал удваивается, но не превышает `59s` (`max_error_refresh_time - min_error_refresh_time`). Фактическая задержка каждый раз выбирается случайным образом в диапазоне от половины до полного текущего интервала.

Для аутентификационного токена, полученного по логину и паролю, параметр `expire_time` не применяется: срок действия такого токена задаётся `login_token_expire_time`, поэтому он перестаёт быть действительным через `12h`. Кешированная запись перестаёт использоваться при ближайшей проверке после этого условия, но может быть удалена раньше из-за отсутствия запросов в течение `2h` (`life_time`) или постоянной ошибки.

Пример для запросов с подписью ключом доступа:

```
auth_config:
  refresh_period: "1s"
  life_time: "30m"
  as_signature_expire_time: "1h"
```

Токен пользователя для запроса с подписью ключом доступа не обновляется планово. Запись считается действительной `1h` (`as_signature_expire_time`) с момента успешной проверки и удаляется при ближайшей проверке кеша. При указанных значениях она будет удалена не позднее чем через `1h + 1s` (`as_signature_expire_time + refresh_period`). Запись может быть удалена раньше из-за отсутствия запросов в течение `30m` (`life_time`).

Конфигурация токена регистрации узлов

YDB позволяет настраивать тип аутентификации узлов баз данных при их регистрации в кластере. Настройка этого типа осуществляется через параметр `node_registration_token` секции `auth_config` .

Параметр	Описание
`node_registration_token`	Определяет тип аутентификации узлов баз данных при их регистрации в кластере YDB. Возможные значения: • Пустая строка (`""`) — используется режим аутентификации узлов через сертификаты TLS. В этом случае узлы должны использовать сертификаты для аутентификации при регистрации в кластере. Подробнее о настройке аутентификации узлов через сертификаты см. в разделе Аутентификация и авторизация узлов баз данных. • "root@builtin" — режим аутентификации через особый отладочный токен. Данный режим планируется к исключению в будущих релизах и не рекомендуется к использованию: для обеспечения безопасности кластера рекомендуется использовать режим аутентификации узлов через сертификаты TLS, установив пустое значение параметра.

Пример секции `auth_config` с настройкой регистрации узлов по сертификату:

```
auth_config:
  #...
  node_registration_token: ""
  #...
```

aws_client_config

Секция `aws_client_config` задает значения по умолчанию для клиента AWS SDK, который YDB использует при обращении к S3 и к другим объектным хранилищам с совместимым API. В ней задаются таймауты, параметры TLS и логирование для всех таких исходящих подключений процесса узла кластера; при необходимости их переопределяет конфигурация конкретного хранилища.

Синтаксис

```
aws_client_config:
  log_config:
    log_level: 0
    filename_prefix: ""
  verify_ssl: true
  request_timeout_ms: 0
  http_request_timeout_ms: 0
  connection_timeout_ms: 10000
  enable_tcp_keep_alive: true
  tcp_keep_alive_interval_ms: 30000
  max_connections_count: 32
  executor_threads_count: 32
  ca_path: "/etc/ssl/certs"
```

Параметры

Параметр	Значение по умолчанию	Описание
`log_config.log_level`	`0` (Off)	Числовое значение уровня детализации логов AWS SDK. Имена уровней и числовые значения см. в разделе «Уровни логов».
`log_config.filename_prefix`	—	Префикс для файлов логов AWS SDK. При задании файлы создаются как `<prefix>YYYY-MM-DD-HH.log`; иначе вывод идёт в stdout.
`verify_ssl`	`true`	Проверять TLS-сертификаты при подключении к endpoint AWS (и совместимые API). Отключать стоит только в тестовых средах.
`request_timeout_ms`	`0` (выкл.)	Таймаут на уровне AWS SDK для запроса целиком (включая DNS, установление соединения, TLS и передачу данных). Значение задается в миллисекундах. `0` — без ограничения со стороны SDK.
`http_request_timeout_ms`	`0` (выкл.)	Таймаут на уровне HTTP-клиента (Curl), в миллисекундах. `0` — без таймаута.
`connection_timeout_ms`	`10000`	Таймаут установления TCP-соединения в миллисекундах.
`enable_tcp_keep_alive`	`true`	Включить TCP keep-alive на клиентских сокетах.
`tcp_keep_alive_interval_ms`	`30000` (дефолт AWS SDK)	Интервал между keep-alive пакетами в миллисекундах. Если параметр не задан в конфигурации YDB, используется дефолт AWS SDK — 30000 мс. В Curl значение округляется до секунд; должен быть не меньше 15000.
`max_connections_count`	`32`	Максимум одновременных TCP-соединений, которые может открыть HTTP-клиент в составе AWS SDK.
`executor_threads_count`	`32`	Размер пула потоков, выделяемого для каждого уникального S3 endpoint. Задает параллелизм асинхронных запросов к этому endpoint.
`ca_path`	`"/etc/ssl/certs"`	Путь к каталогу с корневыми сертификатами для проверки TLS при исходящих подключениях.
`ca_file`	—	Путь к файлу с набором корневых сертификатов. Используется, если хранилище доверенных сертификатов расположено нестандартно.

Уровни логов

Уровень	Значение	Описание
`Off`	`0`	Отключить логирование AWS SDK.
`Fatal`	`1`	Только фатальные ошибки.
`Error`	`2`	Некритические ошибки.
`Warn`	`3`	Предупреждения, требующие внимания.
`Info`	`4`	Общая информационная сводка.
`Debug`	`5`	Отладочная информация.
`Trace`	`6`	Максимально подробная диагностика.

См. также

- log_config — логирование серверного процесса YDB; вложенный в `aws_client_config` блок `log_config` относится только к AWS SDK и с ним не связан.
- Выгрузка и загрузка данных — импорт и экспорт через S3 из CLI; доступ к хранилищу задается там, а не в `aws_client_config`.

Связанные сценарии («S3 в YDB»)

Другие задачи с объектным хранилищем; настройки из этих материалов могут переопределять настройки, заданные в секции `aws_client_config`.

- CREATE EXTERNAL DATA SOURCE — внешний источник данных с протоколом S3 в YQL.
- Развёртывание YDB с функцией Federated Query — федеративные запросы и внешние источники.
- Импорт и экспорт данных с использованием федеративных запросов — импорт и экспорт через федеративные запросы к объектному хранилищу.

blob_storage_config

Укажите конфигурацию статической группы кластера. Статическая группа необходима для работы базовых таблеток кластера, в том числе `Hive`, `SchemeShard`, `BlobstorageController`.
Обычно данные таблетки не хранят много информации, поэтому мы не рекомендуем создавать более одной статической группы.

Для статической группы необходимо указать информацию о дисках и узлах, на которых будет размещена статическая группа. Например, для модели `erasure: none` конфигурация может быть такой:

```
blob_storage_config:
  service_set:
    groups:
    - erasure_species: none
      rings:
      - fail_domains:
        - vdisk_locations:
          - node_id: 1
            path: /dev/disk/by-partlabel/ydb_disk_ssd_02
            pdisk_category: SSD
# ...
```

Для конфигурации, расположенной в 3 зонах доступности, необходимо указать 3 кольца. Для конфигураций, расположенных в одной зоне доступности, указывается ровно одно кольцо.

bridge_config

Секция описывает pile кластера для режима bridge. Укажите список имён pile, которые используются для привязки хостов и других сущностей. В режиме bridge для каждого хоста также необходимо указать имя соответствующего pile в секции `hosts` (поле `bridge_pile_name`), см. hosts.

Синтаксис

```
bridge_config:
  piles:
  - name: <pile_name_1>
  - name: <pile_name_2>
  ...
  - name: <pile_name_n>
```

client_certificate_authorization

Секция `client_certificate_authorization` содержит настройки запроса клиентского сертификата при аутентификации устройств по gRPCs и задаёт правила проверки клиентских SSL-сертификатов и формирования SID пользователей при аутентификации по клиентскому сертификату. Настройки указываются в статической конфигурации кластера. Вложенный блок `client_certificate_definitions` задаёт требования к заполнению полей "Subject" и "Subject Alternative Name" клиентских сертификатов, а также список присваиваемых SID групп.

Клиентские сертификаты описываются стандартом X.509. Поле "Subject" сертификата состоит из нескольких компонентов (например, `O` — организация, `OU` — подразделение в составе организации, `C` — страна, `CN` — имя собственное субъекта). Проверки могут быть настроены на соответствие одного или нескольких компонентов поля ожидаемым значениям.

Поле "Subject Alternative Name" сертификата представляет собой список сетевых имён или IP-адресов. Проверка может быть настроена на соответствие сетевых имён в сертификате ожидаемым значениям.

Синтаксис

```
client_certificate_authorization:
  request_client_certificate: Bool
  client_certificate_required: Bool
  default_group: <SID по умолчанию>
  client_certificate_definitions:
    - member_groups: <массив SID>
      require_same_issuer: Bool
      subject_dns:
        suffixes: <массив разрешенных суффиксов>
        values: <массив допустимых значений>
      subject_terms:
      - short_name: <имя компонента Subject Name>
        suffixes: <массив разрешенных суффиксов>
        values: <массив допустимых значений>
    - member_groups: <массив SID>
    ...
```

Ключ	Описание
`request_client_certificate`	Запрос клиентского сертификата при TLS-handshake для gRPCs. Допустимые значения: • `true` — сервер запрашивает клиентский сертификат. Если сертификат предъявлен, он проверяется на TLS-уровне при аутентификации устройств (соединение с недоверенным сертификатом не устанавливается); после успешно пройденной проверки сервер приоритетно использует аутентификационный токен для аутентификации клиента, а при отсутствии токена выполняется аутентификация по клиентскому сертификату. Если сертификат не предъявлен, то при настройке `client_certificate_required: true` соединение не устанавливается. • `false` — сервер не запрашивает клиентский сертификат при TLS-handshake; аутентификация устройств и аутентификация по клиентскому сертификату по gRPCs при такой настройке недоступны. Значение по умолчанию: `false`.
`client_certificate_required`	Требование клиентского сертификата при TLS-handshake для gRPCs. Допустимые значения: • `true` — сервер требует клиентский сертификат: соединение без сертификата или с недоверенным сертификатом не устанавливается. Можно указать `true` только вместе с `request_client_certificate: true`. • `false` — наличие клиентского сертификата не обязательно; поведение при запросе сертификата определяется параметром `request_client_certificate`. Значение по умолчанию: `false`.
`default_group`	SID, присваиваемый всем подключениям с доверенным клиентским сертификатом, если нет явно заданных настроек в секции `client_certificate_definitions`. Значение по умолчанию: `DefaultClientAuth@cert`.
`client_certificate_definitions`	Блок настроек требований к клиентским сертификатам.
`member_groups`	Массив SID групп, присваиваемых подключениям, сертификаты которых соответствуют требованиям этого блока.
`require_same_issuer`	Требовать совпадение поля "Issuer" (наименование центра регистрации) для клиентского и серверного сертификатов. Допустимые значения: • `true` — совпадение требуется (применяется по умолчанию, если параметр не задан); • `false` — совпадение не требуется (клиентский и серверный сертификаты могут быть выпущены разными центрами регистрации).
`subject_dns`	Допустимые значения поля "Subject Alternative Name" в виде массива полных значений (ключ `values`) или массива суффиксов значений (ключ `suffixes`). Проверка считается успешной при совпадении одного из значений поля с любым полным именем либо при соответствии любому указанному суффиксу.
`subject_terms`	Требования к заполнению компонентов поля "Subject". Указывается имя компонента (ключ `short_name`), а также набор полных значений (ключ `values`) или набор суффиксов значений (ключ `suffixes`). Проверка считается успешной, если для каждого проверяемого компонента поля "Subject" его значение соответствует одному из разрешённых полных значений либо одному из разрешённых суффиксов.

Примеры

Следующий фрагмент конфигурации требует, чтобы в поле "Subject" клиентского сертификата были компоненты `O=YDB` и `CN=user1`. Для такого сертификата будет сформирован SID пользователя `O=YDB,CN=user1@cert` и будет назначена группа `group@cert`:

```
client_certificate_authorization:
  request_client_certificate: true
  client_certificate_definitions:
    - member_groups: ["group@cert"]
      subject_terms:
      - short_name: "O"
        values: ["YDB"]
      - short_name: "CN"
        values: ["user1"]
```

В компоненте `CN` может указываться сетевое имя сервера, а не имя пользователя. Такой вариант целесообразно использовать при регистрации динамических узлов. Следующий фрагмент конфигурации требует, чтобы в поле "Subject" клиентского сертификата узла были компоненты `O=YDB` и `CN=server1.internal.corp`. Для такого сертификата будет сформирован SID `O=YDB,CN=server1.internal.corp@cert` и будет назначена группа `registerNode@cert`:

```
client_certificate_authorization:
  request_client_certificate: true
  client_certificate_definitions:
    - member_groups: ["registerNode@cert"]
      subject_terms:
      - short_name: "O"
        values: ["YDB"]
      - short_name: "CN"
        values: ["server1.internal.corp"]
```

cms_config

Cluster Management System (CMS) — компонент YDB, с помощью которого можно выполнять безопасное обслуживание кластера YDB, например, обновлять его версию или заменять сломавшиеся диски без потери доступности. Поведение CMS конфигурируется в секции `cms_config` конфигурации YDB.

Синтаксис

```
cms_config:
  tenant_limits:
    disabled_nodes_limit: 2
    disabled_nodes_ratio_limit: 5
  cluster_limits:
    disabled_nodes_limit: 3
    disabled_nodes_ratio_limit: 5
  disable_maintenance: true
```

Параметры

Параметр	Значение по умолчанию	Описание
`tenant_limits.disabled_nodes_limit`	-	Максимальное количество узлов базы данных, которые могут быть одновременно недоступны или заблокированы.
`tenant_limits.disabled_nodes_ratio_limit`	`13`	Максимальный процент узлов базы данных, которые могут быть одновременно недоступны или заблокированы.
`cluster_limits.disabled_nodes_limit`	-	Максимальное количество узлов кластера, которые могут быть одновременно недоступны или заблокированы.
`cluster_limits.disabled_nodes_ratio_limit`	`13`	Максимальный процент узлов кластера, которые могут быть одновременно недоступны.
`disable_maintenance`	`false`	Флаг приостанавливает новые работы по обслуживанию кластера.

domains_config

Данный раздел содержит конфигурацию кластера YDB, включая конфигурации Blob Storage (хранилища бинарных объектов) и State Storage (хранилища состояний).

> **Примечание**
>
> Не требуется для работы кластера с автоматической конфигурацией State Storage и статической группы.

Синтаксис

```
domains_config:
  domain:
  - name: <имя корня кластера>
    storage_pool_types: <конфигурация Blob Storage>
  state_storage: <конфигурация State Storage>
  security_config: <конфигурация безопасности>
```

> **Примечание**
>
> Формально, поле `domain` может содержать много элементов, так как это список. Но имеет значение только первый элемент, остальные будут проигнорированы (в кластере может быть только один «домен»).

Конфигурация Blob Storage

> **Примечание**
>
> Не требуется для работы кластера с автоматической конфигурацией State Storage и статической группы.

Данный раздел определяет один или более типов пулов хранения, доступных в кластере для данных в базах данных, со следующими возможностями конфигурации:

- имя пула хранения;
- свойства устройств (например, тип дисков);
- шифрование данных (вкл/выкл);
- режим отказоустойчивости.

Синтаксис

```
domains_config:
  domain:
    ...
    storage_pool_types:
    - kind: <имя пула хранения>
      pool_config:
        box_id: 1
        encryption_mode: <опциональный, укажите 1 для шифрования данных на диске>
        erasure_species: <имя режима отказоустойчивости - none, block-4-2, or mirror-3-dc>
        kind: <имя пула хранения - укажите то же значение, что выше>
        pdisk_filter:
        - property:
          - type: <тип устройства для сопоставления с указанным в host_configs.drive.type>
        vdisk_kind: Default
    - kind: <имя пула хранения>
```

Каждой базе данных в кластере назначается как минимум один из доступных пулов хранения, выбираемый в операции создания базы данных. Имена пулов хранения среди назначенных могут быть использованы в атрибуте `DATA` при определении групп колонок в операторах YQL `CREATE TABLE` / `ALTER TABLE`.

Конфигурация State Storage

> **Примечание**
>
> Не требуется для работы кластера с автоматической конфигурацией State Storage и статической группы.

State Storage (хранилище состояний) — это независимое хранилище в памяти для изменяемых данных, поддерживающее внутренние процессы YDB. Оно хранит реплики данных на множестве назначенных узлов.

Обычно State Storage не требует масштабирования в целях повышения производительности, поэтому количество узлов в нём следует делать как можно меньшим с учётом необходимого уровня отказоустойчивости.

Доступность State Storage является ключевой для кластера YDB, так как влияет на все базы данных, независимо от того какие пулы хранения в них используются. Для обеспечения отказоустойчивости State Storate его узлы должны быть выбраны таким образом, чтобы гарантировать рабочее большинство в случае ожидаемых отказов.

Для выбора узлов State Storage можно воспользоваться следующими рекомендациями:

Тип кластера	Мин количество узлов	Рекомендации по выбору
Без отказоустойчивости	1	Выберите один произвольный узел.

В пределах одной зоны доступности	5	Выберите пять узлов в разных доменах отказа пула хранения block-4-2, для гарантии, что большинство в трёх работающих узлах (из пяти) остаётся при сбое двух доменов.
Георазпределенный	9	Выберите три узла в разных доменах отказа внутри каждой из трёх зон доступности пула хранения mirror-3-dc для гарантии, что большинство в пяти работающих узлов (из девяти) остаётся при сбое зоны доступности + домена отказа.

При развёртывании State Storage на кластерах, в которых применяется несколько пулов хранения с возможным сочетанием режимов отказоустойчивости, рассмотрите возможность увеличения количества узлов с распространением их на разные пулы хранения, так как недоступность State Storage приводит к недоступности всего кластера.

```
domains_config:
  ...
  state_storage:
  - ring:
      node: <массив узлов StateStorage>
      nto_select: <количество реплик данных в StateStorage>
    ssid: 1
```

Каждый клиент State Storage (например, таблетка DataShard) использует `nto_select` узлов для записи копий его данных в State Storage. Если State Storage состоит из большего количества узлов чем `nto_select`, то разные узлы могут быть использованы для разных клиентов. Поэтому необходимо обеспечить, чтобы любое подмножество из `nto_select` узлов в пределах State Storage отвечало критериям отказоустойчивости.

Для `nto_select` должны использоваться нечётные числа, так как использование чётных чисел не улучшает отказоустойчивость по сравнению с ближайшим меньшим нечётным числом.

Полные примеры конфигурации

Один дата-центр с `block-4-2` erasure

```
domains_config:
  domain:
  - name: Root
    storage_pool_types:
    - kind: ssd
      pool_config:
        box_id: 1
        erasure_species: block-4-2
        kind: ssd
        pdisk_filter:
        - property:
          - type: SSD
        vdisk_kind: Default
  state_storage:
  - ring:
      node: [1, 2, 3, 4, 5, 6, 7, 8]
      nto_select: 5
    ssid: 1
```

Несколько дата-центров с `mirror-3-dc` erasure

```
domains_config:
  domain:
  - name: Root
    storage_pool_types:
    - kind: ssd
      pool_config:
        box_id: 1
        erasure_species: mirror-3-dc
        kind: ssd
        pdisk_filter:
        - property:
          - type: SSD
        vdisk_kind: Default
  state_storage:
  - ring:
      node: [1, 2, 3, 4, 5, 6, 7, 8, 9]
      nto_select: 9
    ssid: 1
```

Без отказоустойчивости (`none`) - для тестирования

```
domains_config:
  domain:
  - name: Root
    storage_pool_types:
    - kind: ssd
      pool_config:
        box_id: 1
        erasure_species: none
        kind: ssd
        pdisk_filter:
        - property:
          - type: SSD
        vdisk_kind: Default
```

```
  state_storage:
  - ring:
      node: [1]
      nto_select: 1
    ssid: 1
```

Несколько типов пулов хранения

```
domains_config:
  domain:
  - name: Root
    storage_pool_types:
    - kind: ssd
      pool_config:
        box_id: '1'
        erasure_species: block-4-2
        kind: ssd
        pdisk_filter:
        - property:
          - {type: SSD}
        vdisk_kind: Default
    - kind: rot
      pool_config:
        box_id: '1'
        erasure_species: block-4-2
        kind: rot
        pdisk_filter:
        - property:
          - {type: ROT}
        vdisk_kind: Default
    - kind: rotencrypted
      pool_config:
        box_id: '1'
        encryption_mode: 1
        erasure_species: block-4-2
        kind: rotencrypted
        pdisk_filter:
        - property:
          - {type: ROT}
        vdisk_kind: Default
    - kind: ssdencrypted
      pool_config:
        box_id: '1'
        encryption_mode: 1
        erasure_species: block-4-2
        kind: ssdencrypted
        pdisk_filter:
        - property:
          - {type: SSD}
        vdisk_kind: Default
  state_storage:
  - ring:
      node: [1, 16, 31, 46, 61, 76, 91, 106]
      nto_select: 5
    ssid: 1
```

feature_flags

В рамках классического подхода feature toggling проект YDB реализует механизм фича-флагов.

Фича-флаги — настройки, которые включают или отключают отдельные возможности YDB. Они позволяют включать новую функциональность поэтапно: её код уже может присутствовать в установленной версии YDB, а сама функциональность ещё недоступна пользователям.

Кто переключает фича-флаги

Переключать фича-флаги может владелец функциональности, знакомый с её внутренней реализацией. Переключение — сложный процесс: необходимо учитывать состояние кластера, зависимости между возможностями и совместимость версий и форматов данных.

Не переключайте флаги самостоятельно, если у вас недостаточно знаний о внутренней реализации функциональности и последствиях изменения её состояния. Перед переключением изучите документацию нужной версии YDB и при необходимости обсудите изменение с владельцем функциональности.

> **Важно**
>
> Включение флага может повлиять на совместимость данных и заблокировать откат YDB на предыдущую версию. Такой откат может понадобиться, если после обновления обнаружится критическая ошибка.

Экспериментальная функциональность

Почему флаг может быть выключен

Часть функциональности попадает в релиз в экспериментальном состоянии. Она ещё находится в разработке, поэтому по умолчанию выключена. Её поведение, синтаксис, API или формат хранения данных могут измениться без сохранения обратной совместимости. Такая функциональность может долго оставаться экспериментальной или быть удалена, так и не став общедоступной.

При этом не каждый выключенный флаг означает экспериментальную функциональность. Некоторые флаги управляют стабильными возможностями, которые нужны не всем. Статус конкретной возможности проверяйте по её документации для нужной версии YDB.

Наличие описания возможности в документации или её кода в релизе ещё не означает, что она включена на кластере вашей базы.

Чем опасно включение экспериментальной функциональности

Для экспериментальной функциональности не гарантируются стабильность, производительность и обратная совместимость. Включение её флага может привести к:

- ошибкам при выполнении запросов или нестабильной работе кластера;
- снижению производительности;
- записи данных в формате, который не поддерживается предыдущей версией YDB.

В последнем случае отключение флага не обязательно возвращает данные в прежний формат. Поэтому возможность отката версии нужно оценивать до включения функциональности, а не после возникновения проблемы.

Не включайте экспериментальную функциональность в production-кластерах. Для проверки используйте отдельный тестовый кластер с данными, которые не жалко потерять.

Как узнать, доступна ли функциональность

При проверке доступности учитывайте установленную версию YDB и конфигурацию кластера.

1. Прочитайте документацию нужной версии и release notes. Проверьте ограничения функциональности и указания на её экспериментальный статус.
2. Сгенерируйте конечную конфигурацию для узлов, которые обслуживают нужную базу данных, и проверьте значение соответствующего флага. Описание флагов приведено в справочнике конфигурации.

Значение флага по умолчанию определяется кодом соответствующей версии YDB. В конфигурации кластера значение может отличаться от значения по умолчанию. Поэтому справочник флагов не заменяет проверку фактической доступности на вашем кластере.

Как функциональность становится общедоступной

Когда экспериментальная функциональность готова к общему использованию, её флаг включают по умолчанию в соответствующей версии YDB. Из документации этой версии убирают предупреждение об экспериментальном статусе, а в release notes появляется запись о доступности функциональности.

После обновления кластера проверьте его конфигурацию: явно заданное значение флага может отличаться от нового значения по умолчанию.

Настройка флагов

Секция `feature_flags` включает или отключает отдельные возможности YDB в основной конфигурации кластера. С помощью селекторов динамической конфигурации можно переопределить флаги для отдельных баз данных или групп узлов. Чтобы включить возможность, установите соответствующий фича-флаг в значение `true`. Например, для включения поддержки автопартиционирования топиков в CDC добавьте следующие строки в конфигурацию:

```
feature_flags:
  enable_topic_autopartitioning_for_cdc: true
```

Функциональные флаги

Флаг	Функция
`enable_json_index`	JSON-индексы для ускорения поиска в полях JSON

`enable_json_index_auto_select`	Автоматический выбор JSON-индексов при выполнении запросов
`enable_fulltext_index`	Полнотекстовый индекс для полнотекстового поиска
`enable_local_bloom_filter_index`	Локальный Блум-индекс типа `bloom_filter`
`enable_local_bloom_ngram_filter_index`	Локальный Блум-индекс типа `bloom_ngram_filter`
`enable_local_min_max_index`	Локальный min_max-индекс
`enable_topic_autopartitioning_for_cdc`	Автопартиционирование топиков в CDC для строковых таблиц
`enable_access_to_index_impl_tables`	Возможность указания числа реплик для вторичного индекса
`enable_changefeeds_export`, `enable_changefeeds_import`	Поддержка потоков изменений (changefeed) в операциях резервного копирования и восстановления
`enable_view_export`	Поддержка представлений (`VIEW`) в операциях резервного копирования и восстановления
`enable_export_auto_dropping`	Автоудаление временных директорий и таблиц при экспорте в S3
`enable_followers_stats`	Системные представления с информацией об истории перегруженных партиций
`enable_strict_acl_check`	Запрет на выдачу прав несуществующим пользователям и на удаление пользователей, которым выданы права
`enable_strict_user_management`	Строгие правила администрирования локальных пользователей (т.е. администрировать локальных пользователей может только администратор кластера или базы данных)
`enable_database_admin`	Добавление роли администратора базы данных
`enable_kill_session`	Принудительное завершение сессий через KILL SESSION
`enable_kafka_native_balancing`	Клиентская балансировка партиций при чтении по протоколу Kafka
`enable_topic_compactification_by_key`	Включение компактификации топиков в YDB Topics Kafka API
`enable_kafka_transactions`	Включение транзакций в YDB Topics Kafka API
`enable_external_data_sources`	Включение внешних источников данных
`enable_grpc_audit`	Включение аудита изменений состояния gRPC-соединения
`enable_fs_backups`	Включение операций резервного копирования и восстановления на сетевую файловую систему
`switch_to_config_v2`	Переход с конфигурации V1 на конфигурацию V2
`enable_streaming_queries`	Включение потоковых запросов
`enable_stable_node_names`	Включение стабильных имен динамических узлов
`enable_alter_database_create_hive_first`	Создание отдельного Hive для системных таблеток базы при создании базы данных

query_service_config

Секция `query_service_config` описывает параметры работы YDB с внешними источниками данных с использованием федеративных запросов.

Если для доступа к нужному вам источнику требуется развернуть коннектор, его необходимо также настроить по инструкции.

Описание параметров

Параметр	Значение по умолчанию	Описание
`generic.connector.endpoint.host`	`localhost`	Имя хоста коннектора.
`generic.connector.endpoint.port`	`2130`	TCP порт коннектора.
`generic.connector.use_ssl`	`false`	Использовать ли шифрование соединения. При размещении коннектора и динамического узла YDB на одном сервере шифрованное соединение между ними не требуется, но при необходимости его можно включить.
`generic.connector.ssl_ca_crt`	пустая строка	Путь к сертификату CA, который используется для шифрования.
`generic.default_settings.name.UsePredicatePushdown`	`false`	Включает пушдаун предикатов во внешние источники данных: некоторые части SQL-запросов (например, фильтры) будут переданы на исполнение во внешний источник. Это позволит существенно снизить объёмы данных, передаваемых по сети источником данных в сторону федеративной YDB, сэкономить её вычислительные ресурсы и значительно уменьшить время обработки федеративного запроса.
`available_external_data_sources`	пустой список	Список с разрешенными типами внешних источников. Применяется при `all_external_data_sources_are_available: false`. Возможные значения: • `ObjectStorage`; • `ClickHouse`; • `PostgreSQL`; • `MySQL`; • `Greenplum`; • `MsSQLServer`; • `Ydb`.
`all_external_data_sources_are_available`	`false`	Включение всех типов внешних источников. Если включено, настройка `available_external_data_sources` игнорируется.

Примеры

Включение внешних источников ClickHouse и MySQL

```
query_service_config:
  generic:
    connector:
      endpoint:
        host: localhost                  # имя хоста, где развернут коннектор
        port: 2130                       # номер порта коннектора
      use_ssl: false                     # флаг, включающий шифрование соединений
      ssl_ca_crt: "/opt/ydb/certs/ca.crt" # путь к сертификату CA
    default_settings:
    - name: UsePredicatePushdown
      value: "true"
  all_external_data_sources_are_available: false
  available_external_data_sources:
  - ClickHouse
  - MySQL
```

Включение всех типов внешних источников

```
query_service_config:
  generic:
    connector:
      endpoint:
        host: localhost                  # имя хоста, где развернут коннектор
        port: 2130                       # номер порта коннектора
      use_ssl: false                     # флаг, включающий шифрование соединений
      ssl_ca_crt: "/opt/ydb/certs/ca.crt" # путь к сертификату CA
    default_settings:
    - name: UsePredicatePushdown
      value: "true"
  all_external_data_sources_are_available: true
```

См. также

- Развёртывание YDB с функцией Federated Query

health_check_config

Секция `health_check_config` настраивает пороговые значения и таймауты, используемые сервисом Health Check YDB. Эти параметры помогают настраивать обнаружение возможных проблем, таких как чрезмерные перезапуски или расхождение по времени между динамическими узлами.

Синтаксис

```
health_check_config:
  thresholds:
    node_restarts_yellow: 10
    node_restarts_orange: 30
    nodes_time_difference_yellow: 5000
    nodes_time_difference_orange: 25000
    tablets_restarts_orange: 30
  timeout: 20000
```

Параметры

Параметр	Значение по умолчанию	Описание
`thresholds.node_restarts_yellow`	`10`	Количество перезапусков узлов для генерации предупреждения уровня `YELLOW`
`thresholds.node_restarts_orange`	`30`	Количество перезапусков узлов для генерации предупреждения уровня `ORANGE`
`thresholds.nodes_time_difference_yellow`	`5000`	Максимально допустимое расхождение по времени (в µs) между динамическими узлами для уровня `YELLOW`
`thresholds.nodes_time_difference_orange`	`25000`	Максимально допустимое расхождение по времени (в µs) между динамическими узлами для уровня `ORANGE`
`thresholds.tablets_restarts_orange`	`30`	Количество перезапусков таблеток для генерации предупреждения уровня `ORANGE`
`timeout`	`20000`	Максимальное время ответа от healthcheck (в мс)

hive_config

Hive — компонент YDB, отвечающий за запуск таблеток. В различных ситуациях и под разными паттернами нагрузки может возникнуть потребность в настройке его поведения. Поведение Hive конфигурируется в секции `hive_config` конфигурации YDB. Также часть опций из конфигурации доступна для редактирования через интерфейс Hive web-viewer. Настройки, выставленные через интерфейс, имеют приоритет над указанными в конфигурации. Ниже перечислены все доступные опции, с указанием соответствующего названия опции в интерфейсе, если опцию возможно редактировать через интерфейс.

Опции запуска таблеток

С помощью этих опций можно регулировать с какой скоростью запускаются таблетки и как для них выбираются узлы.

Название параметра в конфигурации	Название параметра в Hive web-viewer	Формат	Описание	Значение по умолчанию
`max_tablets_scheduled`	MaxTabletsScheduled	Целое число	Максимальное число таблеток, одновременно находящихся в процессе старта на одном узле.	100
`max_boot_batch_size`	MaxBootBatchSize	Целое число	Максимальное число таблеток из очереди запуска Hive, обрабатываемых за раз.	1000
`node_select_strategy`	NodeSelectStrategy	Перечисление	Стратегия выбора узла для запуска таблетки. Возможные варианты: • `HIVE_NODE_SELECT_STRATEGY_WEIGHTED_RANDOM` — взвешенно-случайный выбор на основе потребления; • `HIVE_NODE_SELECT_STRATEGY_EXACT_MIN` — выбор узла с минимальным потреблением; • `HIVE_NODE_SELECT_STRATEGY_RANDOM_MIN_7P` — выбор случайного узла среди 7% узлов с наименьшим потреблением; • `HIVE_NODE_SELECT_STRATEGY_RANDOM` — выбор случайного узла.	`HIVE_NODE_SELECT_STRATEGY_RANDOM_`
`boot_strategy`	—	Перечисление	Регулирует поведение при запуске большого числа таблеток. Возможные варианты: • `HIVE_BOOT_STRATEGY_BALANCED` — при достижении лимита в `max_tablets_scheduled` на одном узле останавливает запуск новых таблеток на всех узлах. • `HIVE_BOOT_STRATEGY_FAST` — при достижении лимита в `max_tablets_scheduled` на одном узле продолжает запуск таблеток на других узлах. Если при запуске большого количества таблеток один из узлов запускает таблетки чуть медленнее других, то при использовании `HIVE_BOOT_STRATEGY_FAST` на этом узле будет запущено меньше таблеток, чем на остальных. При использовании `HIVE_BOOT_STRATEGY_BALANCED` в той же ситуации таблетки будут равномерно распределены по узлам, но их запуск займёт больше времени.	`HIVE_BOOT_STRATEGY_BALANCED`
`default_tablet_limit`	—	Вложенная секция	Ограничения на запуск таблеток различных типов на одном узле. Указывается в формате списка, где каждый элемент имеет поля `type` и `max_count`.	Пустая секция

`default_tablet_preference`	—	Вложенная секция	Приоритеты по выбору датацентров для запуска таблеток различных типов. Для каждого типа таблетки можно указать несколько групп датацентров. Датацентры внутри одной группы будут иметь одинаковый приоритет, а более ранняя группа будет иметь приориет над последующими. Пример формата:
			`default_tablet_preference:`
			`  - type: Coordinator`
			`  data_centers_preference:`
			`  - data_centers_group:`
			`  - "dc-1"`
			`  - "dc-2"`
			`  - data_centers_group:`
			`  - "dc-3"`
				Пустая секция

`system_category_id`	—	Целое число	При указании любого отличного от 0 числа, все координаторы и медиаторы по возможности запускаются в одном и том же датацентре.	1

Пример

> **Примечание**
>
> В подсекциях `default_tablet_limit` и `default_tablet_preference` нужно указывать типы таблеток. Точные названия типов таблеток указаны в глоссарии.

```
hive_config:
  max_tablets_scheduled: 10
  node_select_strategy: HIVE_NODE_SELECT_STRATEGY_RANDOM
  boot_strategy: HIVE_BOOT_STRATEGY_FAST
  default_tablet_limit:
    - type: PersQueue
      max_count: 15
    - type: DataShard
      max_count: 100
  default_tablet_preference:
    - type: Coordinator
      data_centers_preference:
        - data_centers_group:
          - "dc-1"
          - "dc-2"
        - data_centers_group:
          - "dc-3"
```

Опции автобалансировки

Эти опции регулируют процесс автобалансировки: в каких ситуациях он запускается, сколько таблеток с какими интервалами перевозит, как выбирает узлы и таблетки. Часть опций представлена в двух вариациях: для «emergency-балансировки», то есть балансировки при перегрузке одного или нескольких узлов, и для всех остальных видов балансировки.

Название параметра в конфигурации	Название параметра в Hive web-viewer	Формат	Описание	Значе
`min_scatter_to_balance`	MinScatterToBalance	Вещественное число	Порог метрики Scatter для ресурсов CPU, Memory, Network. Имеет приоритет ниже, чем параметры ниже.	0.5
`min_cpuscatter_to_balance`	MinCPUScatterToBalance	Вещественное число	Порог метрики Scatter для ресурса CPU.	0.5
`min_memory_scatter_to_balance`	MinMemoryScatterToBalance	Вещественное число	Порог метрики Scatter для ресурса Memory.	0.5
`min_network_scatter_to_balance`	MinNetworkScatterToBalance	Вещественное число	Порог метрики Scatter для ресурса Network.	0.5
`min_counter_scatter_to_balance`	MinCounterScatterToBalance	Вещественное число	Порог метрики Scatter для фиктивного ресурса Counter.	0.02
`min_node_usage_to_balance`	MinNodeUsageToBalance	Вещественное число	Потребление ресурсов на узле ниже данного значения приравнивается к данному значению. Используется для того, чтобы не балансировать таблетки между малозагруженными узлами.	0.1
`max_node_usage_to_kick`	MaxNodeUsageToKick	Вещественное число	Порог потребления ресурсов на узле для запуска emergency-автобалансировки.	0.9
`node_usage_range_to_kick`	NodeUsageRangeToKick	Вещественное число	Минимальная разница в уровне потребления ресурсов между узлами, ниже которой автобалансировка считается нецелесообразной.	0.2
`resource_change_reaction_period`	ResourceChangeReactionPeriod	Целое число секунд	Частота обновления агрегированной статистики потребления ресурсов.	10
`tablet_kick_cooldown_period`	TabletKickCooldownPeriod	Целое число секунд	Минимальный период времени между перемещениями одной таблетки.	600
`spread_neighbours`	SpreadNeighbours	true/false	Запуск таблеток одного схемного объекта (таблицы, топика) по возможности на разных узлах.	true
`node_balance_strategy`	NodeBalanceStrategy	Перечисление	Стратегия выбора узла, с которого перевозятся таблетки при автобалансировке. Возможные варианты: • `HIVE_NODE_BALANCE_STRATEGY_WEIGHTED_RANDOM` — взвешенно-случайный выбор на основе потребления; • `HIVE_NODE_BALANCE_STRATEGY_HEAVIEST` — выбор узла с максимальным потреблением; • `HIVE_NODE_BALANCE_STRATEGY_RANDOM` — выбор случайного узла.	`HIVE_`

`tablet_balance_strategy`	TabletBalanceStrategy	Перечисление	Стратегия выбора таблетки для перевоза при автобалансировке. Возможные варианты: • `HIVE_TABLET_BALANCE_STRATEGY_WEIGHTED_RANDOM` — взвешенно-случайный выбор на основе потребления; • `HIVE_TABLET_BALANCE_STRATEGY_HEAVIEST` — выбор таблетки с максимальным потреблением; • `HIVE_TABLET_BALANCE_STRATEGY_RANDOM` — выбор случайной таблетки.	`HIVE_`
`min_period_between_balance`	MinPeriodBetweenBalance	Вещественное число секунд	Минимальный период времени между двумя итерациями автобалансировки. Не относится к emergency-балансировке.	0.2
`balancer_inflight`	BalancerInflight	Целое число	Число таблеток, одновременно перезапускающихся в процессе автобалансировки. Не относится к emergency-балансировке.	1
`max_movements_on_auto_balancer`	MaxMovementsOnAutoBalancer	Целое число	Число перемещений таблеток за одну итерацию автобалансировки. Не относится к emergency-балансировке.	1
`continue_auto_balancer`	ContinueAutoBalancer	true/false	При включении следующая итерация балансировки запускается, не дожидаясь окончания `resource_change_reaction_period`.	true
`min_period_between_emergency_balance`	MinPeriodBetweenEmergencyBalance	Вещественное число секунд	Аналогично `min_period_between_balance`, но для emergency-балансировки.	0.1
`emergency_balancer_inflight`	EmergencyBalancerInfligh	Целое число	Аналогично `balancer_inflight`, но для emergency-балансировки.	1
`max_movements_on_emergency_balancer`	MaxMovementsOnEmergencyBalancer	Целое число	Аналогично `max_movements_on_auto_balancer`, но для emergency-балансировки.	2
`continue_emergency_balancer`	ContinueEmergencyBalancer	true/false	Аналогично `continue_auto_balancer`, но для emergency-балансировки.	true
`check_move_expediency`	CheckMoveExpediency	true/false	Проверка целесообразности перемещений таблеток. Если автобалансировка приводит к повышенному потреблению Hive процессорных ресурсов, можно отключить эту опцию.	true
`object_imbalance_to_balance`	ObjectImbalanceToBalance	Вещественное число	Порог метрики дисбаланса таблеток одного объекта.	0.02
`less_system_tablets_moves`	LessSystemTabletMoves	true/false	Минимизация перемещения системных таблеток при автобалансировке.	true
`balancer_ignore_tablet_types`	BalancerIgnoreTabletTypes	Список типов таблеток. При выставлении через Hive UI — разделённый точкой с запятой.	Типы таблеток, на которые не распространяется автобалансировка.	Пусто

Примеры

С помощью такого файла конфигурации можно полностью отключить все виды автобалансировки таблеток между узлами.

```
hive_config:
  min_cpuscatter_to_balance: 1.0
  min_memory_scatter_to_balance: 1.0
  min_network_scatter_to_balance: 1.0
  min_counter_scatter_to_balance: 1.0
  max_node_usage_to_kick: 3.0
  object_imbalance_to_balance: 1.0
```

Таким файлом конфигурации можно отключить все виды автобалансировки между узлами для таблеток, участвующих в распределении транзакций, то есть координаторов и медиаторов. Точные названия типов таблеток указаны в глоссарии.

```
hive_config:
  balancer_ignore_tablet_types:
    - Coordinator
    - Mediator
```

При использовании Hive UI для такого же эффекта нужно записать `Coordinator;Mediator` в поле ввода для настройки BalancerIgnoreTabletTypes.

Опции сбора метрик потребления вычислительных ресурсов

Hive собирает с каждого узла метрики потребления вычислительных ресурсов — процессорного времени, оперативной памяти, сети — в целом по узлу и с разделением по таблеткам. Эти настройки позволяют регулировать сбор этих метрик, их нормировку и агрегацию.

Название параметра в конфигурации	Название параметра в Hive web-viewer	Формат	Описание	Значение по умолчанию
`max_resource_cpu`	MaxResourceCPU	Целое число микросекунд	Максимальное потребление CPU на узле в секунду. Значение по умолчанию, используется только если узел не предоставляет значение при регистрации в Hive.	10000000
`max_resource_memory`	MaxResourceMemory	Целое число байт	Максимальное потребление памяти на узле. Значение по умолчанию, используется только если узел не предоставляет значение при регистрации в Hive.	512000000000
`max_resource_network`	MaxResourceNetwork	Целое число байт/секунду	Максимальное потребление полосы на узле. Значение по умолчанию, используется только если узел не предоставляет значение при регистрации в Hive.	1000000000
`max_resource_counter`	MaxResourceCounter	Целое число	Максимальное потребление виртуального ресурса Counter на узле.	100000000
`metrics_window_size`	MetricsWindowSize	Целое число миллисекунд	Размер окна, на котором агрегируются метрики потребления ресурсов таблетками.	60000
`resource_overcommitment`	ResourceOvercommitment	Вещественное число	Коэффициент переподписки на ресурсы узлов.	3.0
`pools_to_monitor_for_usage`	—	Названия пулов через запятую	Пулы актор-системы, потребление в которых учитывается при подсчёте потребления ресурсов узла.	System,User,IC

Опции распределения каналов между группами хранения

Здесь перечислены опции, связанные с распределением каналов таблеток по группам хранения: с учётом различных метрик, с выбором групп, с процессом автобалансировки каналов по группам.

Примечание

Эта таблица содержит продвинутые настройки, которые в большинстве случаев не требуют изменения.

Название параметра в конфигурации	Название параметра в Hive web-viewer	Формат	Описание	Значение по умолча
`default_unit_iops`	DefaultUnitIOPS	Целое число	Значение по умолчанию для IOPS одного канала.	1

`default_unit_throughput`	DefaultUnitThroughput	Целое число байт/секунду	Значение по умолчанию для потребления пропускной способности одним каналом.	1000
`default_unit_size`	DefaultUnitSize	Целое число байт	Значение по умолчанию для потребления места на дисках одним каналом.	100000000
`storage_overcommit`	StorageOvercommit	Вещественное число	Коэффициент переподписки на ресурсы групп хранения.	1.0
`storage_balance_strategy`	StorageBalanceStrategy	Перечисление	Выбор параметра используемого для распределения каналов таблеток по группам хранения. Возможные варианты: • `HIVE_STORAGE_BALANCE_STRATEGY_IOPS` — учитывается только IOPS; • `HIVE_STORAGE_BALANCE_STRATEGY_THROUGHPUT` — учитывается только потребление пропускной способности; • `HIVE_STORAGE_BALANCE_STRATEGY_SIZE` — учитывается только объём занятого места; • `HIVE_STORAGE_BALANCE_STRATEGY_AUTO` — учитывается тот из параметров выше, чьё потребление максимально;	`HIVE_STORAGE_BALA`
`storage_safe_mode`	StorageSafeMode	true/false	Проверка превышения максимального потребления ресурсов групп хранения.	true
`storage_select_strategy`	StorageSelectStrategy	Перечисление	Стратегия выбора группы хранения для канала таблетки. Возможные варианты: • `HIVE_STORAGE_SELECT_STRATEGY_WEIGHTED_RANDOM` — взвешенно-случайный выбор на основе потребления; • `HIVE_STORAGE_SELECT_STRATEGY_EXACT_MIN` — выбор группы с минимальным потреблением; • `HIVE_STORAGE_SELECT_STRATEGY_RANDOM_MIN_7P` — выбор случайной группы среди 7% групп с наименьшим потреблением; • `HIVE_STORAGE_SELECT_STRATEGY_RANDOM` — выбор случайной группы; • `HIVE_STORAGE_SELECT_STRATEGY_ROUND_ROBIN` — выбор группы внутри пула хранения по принципу Round-robin.	`HIVE_STORAGE_SELE`
`min_period_between_reassign`	MinPeriodBetweenReassign	Целое число секунд	Минимальный период времени между переназначениями групп хранения для каналов одной таблетки.	300
`storage_pool_fresh_period`	StoragePoolFreshPeriod	Целое число миллисекунд	Периодичность обновления информации о пулах хранения.	60000
`space_usage_penalty_threshold`	SpaceUsagePenaltyThreshold	Вещественное число	Минимальное отношение свободного места в целевой группе к свободному месту в исходной группе, при котором целевая группа будет пессимизирована путём применения к весу мультипликативного штрафа при перевозе канала.	1.1
`space_usage_penalty`	SpaceUsagePenalty	Вещественное число	Коэффициент штрафа пессимизации, описанной выше.	0.2
`channel_balance_strategy`	ChannelBalanceStrategy	Перечисление	Стратегия выбора канала для переназначения при балансировке каналов. Возможные варианты: • `HIVE_CHANNEL_BALANCE_STRATEGY_WEIGHTED_RANDOM` — взвешенно-случайный выбор на основе потребления; • `HIVE_CHANNEL_BALANCE_STRATEGY_HEAVIEST` — выбор канала с максимальным потреблением; • `HIVE_CHANNEL_BALANCE_STRATEGY_RANDOM` — выбор случайного канала.	`HIVE_CHANNEL_BALA`
`max_channel_history_size`	MaxChannelHistorySize	Целое число	Максимальный размер истории каналов.	200
`storage_info_refresh_frequency`	StorageInfoRefreshFrequency	Целое число миллисекунд	Частота обновления информации о пулах хранения.	600000
`min_storage_scatter_to_balance`	MinStorageScatterToBalance	Вещественное число	Порог метрики Scatter для групп хранения.	999
`min_group_usage_to_balance`	MinGroupUsageToBalance	Вещественное число	Порог потребления ресурсов группы хранения, ниже которого не запускается балансировка.	0.1
`storage_balancer_inflight`	StorageBalancerInflight	Целое число	Число таблеток, одновременно перезапускающихся во время балансировки каналов.	1

Опции отслеживания перезапусков

Hive отслеживает, как часто рестартуют различные узлы и таблетки, чтобы определять проблемные. С помощью данных опций можно настроить, какие именно таблетки или узлы будут считаться проблемными, и как это на них повлияет. На основании этой статистики узлы и таблетки попадают в отчёт HealthCheck API.

Опции отслеживания перезапусков таблеток

Название параметра в конфигурации	Название параметра в Hive web-viewer	Формат	Описание	Значение по умолчанию
`tablet_restart_watch_period`	—	Целое число секунд	Размер окна, на котором собирается статистика о числе рестартов таблетки. **Этот период используется только для статистики, передаваемой в HealthCheck.**	3600
`tablet_restarts_period`	—	Целое число миллисекунд	Размер окна, на котором считается количество рестартов таблетки для пессимизации запуска проблемных таблеток.	1000
`tablet_restarts_max_count`	—	Целое число	Количество рестартов на окне `tablet_restarts_period`, при превышении которого применяется пессимизация.	2
`postopone_start_period`	—	Целое число миллисекунд	Периодичность попыток запуска проблемных таблеток.	1000

Опции отслеживания перезапусков узлов

Название параметра в конфигурации	Название параметра в Hive web-viewer	Формат	Описание	Значение по умолчанию
`node_restart_watch_period`	—	Целое число секунд	Размер окна, на котором собирается статистика о числе рестартов узла.	3600
`node_restarts_for_penalty`	NodeRestartsForPenalty	Целое число	Количество рестартов на окне `node_restart_watch_period`, после которого узлы получают понижение приоритета.	3

Прочее

Здесь перечислены дополнительные настройки Hive.

Примечание

Эта таблица содержит продвинутые настройки, которые в большинстве случаев не требуют изменения.

Название параметра в конфигурации	Название параметра в Hive web-viewer	Формат	Описание	Значение по умолчанию
`drain_inflight`	DrainInflight	Целое число	Число таблеток, одновременно перезапускающихся в процессе плавного перемещения всех таблеток с одного узла (drain).	10
`request_sequence_size`	—	Целое число	Количество идентификаторов таблеток, которое за один раз Hive базы данных запрашивает у корневого Hive.	1000
`min_request_sequence_size`	—	Целое число	Минимальное количество идентификаторов таблеток, которое за один раз корневой Hive выделяет для Hive базы данных.	1000
`max_request_sequence_size`	—	Целое число	Максимальное количество идентификаторов таблеток, которое за один раз выделяет для Hive базы данных.	1000000

`node_delete_period`	—	Целое число секунд	Период неактивности, по истечении которого узел удаляется из базы Hive.	3600
`warm_up_enabled`	WarmUpEnabled	true/false	При включении этой опции при старте базы Hive дожидается подключения всех узлов прежде чем начать запускать таблетки. При выключении все таблетки могут быть запущены на первом подключившемся узле.	true
`warm_up_boot_waiting_period`	MaxWarmUpBootWaitingPeriod	Целое число миллисекунд	Время ожидания старта всех известных узлов при старте базы.	30000
`max_warm_up_period`	MaxWarmUpPeriod	Целое число секунд	Максимальное время ожидания старта узлов при старте базы.	600
`enable_destroy_operations`	—	true/false	Разрешены ли деструктивные ручные операции.	false
`max_pings_in_flight`	—	Целое число	Максимальное количество параллельно устаналиваемых соединений с узлами.	1000
`cut_history_deny_list`	—	Список типов таблеток, разделённый запятой	Список типов таблеток, для которых игнорируется операция очистки истории.	ColumnShard,KeyValue,PersQueue,B
`cut_history_allow_list`	—	Список типов таблеток, разделённый запятой	Список типов таблеток, для которых разрешена операция очистки истории.	DataShard
`scale_recommendation_refresh_frequency`	ScaleRecommendationRefreshFrequency	Целое число миллисекунд	Как часто обновляется рекомендация по количеству вычислительных узлов.	60000
`scale_out_window_size`	ScaleOutWindowSize	Целое число	Количество бакетов, на основе которых принимается решение о рекомендации увеличить число вычислительных узлов.	15
`scale_in_window_size`	ScaleInWindowSize	Целое число	Количество бакетов, на основе которых принимается решение о рекомендации уменьшить число вычислительных узлов.	5
`target_tracking_cpumargin`	TargetTrackingCPUMargin	Вещественное число	Допустимое отклонение от целевого значения утилизации CPU при автоскейлинге.	0.1

`dry_run_target_tracking_cpu`	DryRunTargetTrackingCPU	Вещественное число	Целевое значение утилизации CPU для проверки, как работал бы автоскейлинг.	0

host_configs

Кластер YDB состоит из множества узлов, для развертывания которых обычно используется одна или несколько типовых конфигураций серверов. Для того чтобы не повторять её описание для каждого узла, в файле конфигурации существует раздел `host_configs`, в котором перечислены используемые конфигурации, и им присвоены идентификаторы.

Синтаксис

```
host_configs:
- host_config_id: 1
  drive:
  - path: <path_to_device>
    type: <type>
    disk_scope: <disk_scope>  # необязательный атрибут
  - path: ...
- host_config_id: 2
  ...
```

Атрибут `host_config_id` задает числовой идентификатор конфигурации. В атрибуте `drive` содержится коллекция описаний подключенных дисков. Каждое описание состоит из двух обязательных атрибутов:

- `path` : Путь к смонтированному блочному устройству, например `/dev/disk/by-partlabel/ydb_disk_ssd_01`
- `type` : Тип физического носителя устройства: `ssd` , `nvme` или `rot` (rotational - HDD)

Дополнительно может быть указан необязательный атрибут `disk_scope` — метка для вычисления домена отказа для некоторых упрощенных конфигураций, см. Конфигурирование disk_scope.

Примеры

Одна конфигурация с идентификатором 1, с одним диском типа SSD, доступным по пути `/dev/disk/by-partlabel/ydb_disk_ssd_01` :

```
host_configs:
- host_config_id: 1
  drive:
  - path: /dev/disk/by-partlabel/ydb_disk_ssd_01
    type: SSD
```

Две конфигурации с идентификаторами 1 (два SSD диска) и 2 (три SSD диска):

```
host_configs:
- host_config_id: 1
  drive:
  - path: /dev/disk/by-partlabel/ydb_disk_ssd_01
    type: SSD
  - path: /dev/disk/by-partlabel/ydb_disk_ssd_02
    type: SSD
- host_config_id: 2
  drive:
  - path: /dev/disk/by-partlabel/ydb_disk_ssd_01
    type: SSD
  - path: /dev/disk/by-partlabel/ydb_disk_ssd_02
    type: SSD
  - path: /dev/disk/by-partlabel/ydb_disk_ssd_03
    type: SSD
```

Конфигурирование disk_scope

`disk_scope` — необязательный строковый атрибут диска, задающий более детальную зону отказа внутри одного узла. Он учитывается при вычислении доменов отказа во время выбора дисков для размещения VDisk'ов групп хранения.

Домен отказа определяется физическим расположением диска (дата-центр, стойка, сервер, физическое устройство), поэтому либо никакие два VDisk'а одной группы не могут быть размещены на одном узле (если домен отказа соответствует серверу или серверной стойке), либо несколько VDisk'ов одной группы может быть размещено на одном узле на разных дисках (если домен отказа соответствует отдельному физическому устройству). В первом случае невозможно создать конфигурацию в режиме `block-4-2` менее чем с 8 узлами хранения, а во втором случае конфигурация может не переживать отказ 1 узла. Чтобы ограничить количество VDisk'ов одной группы, размещаемых на одном узле, физические устройства можно пометить атрибутом `disk_scope` и включить расчёт доменов отказа по уровню `disk_scope`. Это позволит собирать некоторые отказоустойчивые конфигурации в инсталляциях, в которых серверов меньше, чем требуется доменов отказа для выбранного режима отказоустойчивости.

Пример

Кластер из 4 серверов, на каждом сервере 4 диска в режиме отказоустойчивости `block-4-2` :

```
host_configs:
- host_config_id: 1
  drive:
  - path: /dev/disk/by-partlabel/ydb_disk_ssd_01
    type: SSD
    disk_scope: fail-domain-1
  - path: /dev/disk/by-partlabel/ydb_disk_ssd_02
    type: SSD
    disk_scope: fail-domain-1
  - path: /dev/disk/by-partlabel/ydb_disk_ssd_03
    type: SSD
    disk_scope: fail-domain-2
  - path: /dev/disk/by-partlabel/ydb_disk_ssd_04
```

```
    type: SSD
    disk_scope: fail-domain-2
```

Чтобы домены отказа рассчитывались по уровню `disk_scope` для статической группы и других групп хранения, укажите соответствующий тип домена отказа в yaml-конфиге (верхнеуровневый ключ):

```
fail_domain_type: disk_scope
```

И в конфигурации домена:

```
domains:
- domain_name: <имя домена>
  ...
  storage_pool_kinds:
  - kind: <тип используемых физических устройств>
    fail_domain_type: disk_scope
    ...
```

При такой конфигурации на каждом сервере будут размещены по два VDisk'а каждой группы хранения, и максимальным отказом в такой конфигурации будет отказ одного сервера.

Особенности Kubernetes

YDB Kubernetes operator монтирует NBS диски для Storage узлов на путь `/dev/kikimr_ssd_00`. Для их использования должна быть указана следующая конфигурация `host_configs`:

```
host_configs:
- host_config_id: 1
  drive:
  - path: /dev/kikimr_ssd_00
    type: SSD
```

Файлы с примерами конфигурации, поставляемые в составе YDB Kubernetes operator, уже содержат такую секцию, и её не нужно менять.

hosts

В данной группе перечисляются статические узлы кластера, на которых запускаются процессы работы со Storage, и задаются их основные характеристики:

- Числовой идентификатор узла
- DNS-имя хоста и порт, по которым может быть установлено соединение с узлом в IP network
- Идентификатор типовой конфигурации хоста
- Размещение в определенной зоне доступности, стойке
- Инвентарный номер сервера (опционально)

Синтаксис

```
hosts:
- host: <DNS-имя хоста>
  host_config_id: <числовой идентификатор типовой конфигурации хоста>
  port: <порт> # 19001 по умолчанию
  location:
    unit: <строка с инвентарным номером сервера>
    data_center: <строка с идентификатором зоны доступности>
    rack: <строка с идентификатором стойки>
- host: <DNS-имя хоста>
  # ...
```

Примеры

```
hosts:
- host: hostname1
  host_config_id: 1
  node_id: 1
  port: 19001
  location:
    unit: '1'
    data_center: '1'
    rack: '1'
- host: hostname2
  host_config_id: 1
  node_id: 2
  port: 19001
  location:
    unit: '1'
    data_center: '1'
    rack: '1'
```

Особенности режима bridge

В режиме bridge каждый хост должен быть привязан к одному из pile, объявленных в `bridge_config`. Для этого в разделе `location` укажите поле `bridge_pile_name` с именем pile. Пример:

```
hosts:
- host: hostname1
  host_config_id: 1
  location:
    ...
    bridge_pile_name: 'pile_1'
```

Особенности Kubernetes

При развертывании YDB с помощью оператора Kubernetes секция `hosts` полностью генерируется автоматически, заменяя любой указанный пользователем контент в передаваемой оператору конфигурации. Все Storage узлы используют `host_config_id` = `1`, для которого должна быть задана корректная конфигурация.

immediate_controls_config

Конфигурация `immediate_controls_config` — это набор динамических параметров для настройки компонентов YDB, таких как DataShard, Coordinator, SchemeShard, BlobStorage и другие. Эти настройки позволяют адаптировать поведение кластера под специфичные сценарии, например, настроив пороги автоматического разделения шардов при росте данных или увеличении нагрузки.

Синтаксис

```
immediate_controls_config:
  ...
  scheme_shard_controls:
    force_shard_split_data_size: 2147483648
    ...
```

Параметры

Параметр	Минимальное значение	Максимальное значение	Значение по умолчанию	Описание
`scheme_shard_controls.force_shard_split_data_size`	10 МиБ	16 ГиБ	2 ГиБ	Партиция таблицы будет принудительно разделена при достижении заданного размера, даже если настроенные для таблицы порог размера партиции или максимальное количество партиций не допускают разделение. Значение задаётся в байтах.

kafka_proxy_config

В разделе `kafka_proxy_config` файла конфигурации YDB включается и конфигурируется Kafka Proxy, которая дает доступ к работе с YDB Topics по Kafka API.

Описание параметров

Параметр	Тип	Значение по умолчанию	Описание
`enable_kafka_proxy`	bool	`false`	Включает или отключает Kafka Proxy.
`listening_port`	int32	`9092`	Порт, на котором будет доступен Kafka API.
`listening_address`	string	`[::]`	Сетевой адрес, на котором Kafka Proxy принимает входящие соединения. Значение `[::]` — все интерфейсы (dual-stack, должна быть включена поддержка IPv6), `127.0.0.1` или `[::1]` — только localhost.
`transaction_timeout_ms`	uint32	`300000` (5 минут)	Максимальный таймаут для Kafka транзакций, после которого транзакция будет отменена.
`auto_create_topics_enable`	bool	`false`	Включает автоматическое создание топиков при обращении к ним. Аналог такой же опции в Apache Kafka.
`auto_create_consumers_enable`	bool	`true`	Включает автоматическое заведение консьюмеров при обращении к ним.
`topic_creation_default_partitions`	uint32	`1`	Количество партиций, которое будет создано, если при добавлении топика по Kafka протоколу не было указано количество партиций. Аналог опции num.partitions в Apache Kafka.
`ssl_cerificate`	string	-	Путь к файлу сертификата для доступа по SSL, включающий в себя сразу и файл сертификата и файл ключа. При указании этого параметра Kafka Proxy автоматически начинает обрабатывать запросы с использованием указанного SSL-сертификата.
`cert`	string	-	Путь к файлу сертификата для доступа по SSL. При указании этого параметра Kafka Proxy автоматически начинает обрабатывать запросы с использованием указанного SSL-сертификата.
`key`	string	-	Путь к файлу ключа для доступа по SSL.
`ca`	string	-	Путь к файлу certificate authority для mTLS.
`enable_self_signed_certs`	bool	`false`	Разрешает использование самоподписанных сертификатов для mTLS.
`mtls_enable`	bool	`false`	Включает использование mTLS аутентификации.

Пример заполненного конфига

```yaml
kafka_proxy_config:
  enable_kafka_proxy: true
  listening_port: 9092
  listening_address: "[::]"
  transaction_timeout_ms: 300000 # 5 минут
  auto_create_topics_enable: true
  auto_create_consumers_enable: true
  topic_creation_default_partitions: 1
  cert: /path/to/cert.pem
  key: /path/to/key.pem
```

log_config

Секция `log_config` управляет тем, как серверный процесс YDB обрабатывает и управляет своими логами. Она позволяет настраивать уровни логирования для различных компонентов, а также глобальные форматы логов и методы вывода.

> **Примечание**
>
> Этот документ описывает конфигурацию прикладного и системного логирования. Для информации о логировании для целей аудита и обеспечения безопасности, см. Аудитный лог.

Обзор

Логирование является критически важной частью системы наблюдаемости YDB. Секция `log_config` позволяет настраивать различные аспекты логирования, включая:

- уровень логирования по умолчанию;
- уровни логирования для конкретных компонентов;
- формат вывода логов;
- интеграцию с syslog или внешними сервисами логирования.

Способы вывода логов

- **В stderr**: по умолчанию, YDB отправляет все логи в stderr.
- **В файл**: логи могут быть записаны в файл с помощью параметра `backend_file_name`.
- **В syslog**: при включении параметра `sys_log: true` логи перенаправляются в системный журнал syslog и перестают выводиться в stderr. Логи отправляются с использованием сокета `/dev/log`.
- **В Unified Agent**: при настройке секции `uaclient_config` логи отправляются в Unified Agent и перестают выводиться в stderr.

При одновременном включении `sys_log` и настройке `uaclient_config`, логи будут отправляться и в syslog, и в Unified Agent. Если вам нужно продолжать выводить логи в stderr при использовании других способов, активируйте `sys_log_to_stderr: true`.

Параметры конфигурации

Параметр	Тип	Значение по умолчанию	Описание
`default_level`	uint32	5 (`NOTICE`)	Уровень логирования по умолчанию для всех компонентов.
`default_sampling_level`	uint32	7 (`DEBUG`)	Уровень семплирования по умолчанию для всех компонентов.
`default_sampling_rate`	uint32	0	Частота семплирования по умолчанию для всех компонентов. Если установлено значение N (где N > 0), примерно 1 из каждых N сообщений журнала с приоритетом между `default_level` и `default_sampling_level` будет записано в лог. Например, чтобы записывать каждое 10-е сообщение, установите значение 10. Значение 0 означает, что сообщения в этом диапазоне не будут записаны в лог (все они отбрасываются).
`sys_log`	bool	false	Включить логирование через syslog.
`sys_log_to_stderr`	bool	false	Копировать логи в stderr в дополнение к syslog.
`format`	string	"full"	Формат вывода логов. Возможные значения: "full", "short", "json".
`cluster_name`	string	—	Имя кластера для включения в записи логов. Поле `cluster_name` добавляется в логи только при использовании формата `json` или при отправке в Unified Agent. В форматах `full` или `short` это поле не отображается.
`allow_drop_entries`	bool	true	Разрешить отбрасывание записей логов, если система логирования перегружена. При включении этой опции записи логов буферизуются в памяти и записываются в вывод, когда накапливается 10 сообщений или истекает время, указанное в `time_threshold_ms`. Если буфер переполняется, сообщения с низким приоритетом могут быть отброшены, чтобы освободить место для сообщений с более высоким приоритетом.
`use_local_timestamps`	bool	false	Использовать локальный часовой пояс для временных меток логов (по умолчанию используется UTC).
`backend_file_name`	string	—	Имя файла для вывода логов. Если указано, логи записываются в этот файл.
`sys_log_service`	string	—	Имя сервиса для syslog. Соответствует полю tag в старом протоколе syslog RFC 3164 или полю app-name в современном протоколе RFC 5424.
`time_threshold_ms`	uint64	1000	Если `allow_drop_entries = true`, определяет как часто, в миллисекундах, YDB пишет записи логов из буфера в вывод.
`ignore_unknown_components`	bool	true	Игнорировать запросы логирования от неизвестных компонентов.
`entry`	array	[]	Конфигурация уровня логирования и/или семплирования для конкретных компонентов YDB, см. Объекты Entry ниже.

`uaclient_config`	object	—	Конфигурация для клиента Unified Agent, см. Объект UAClientConfig ниже.
`enable_structured_log_in_json`	bool	false	При выводе журнала в формате `json`: если `false`, текст сообщения (поле `message`) содержит структурированные значения в виде `key=value` (например, `Text (a=1 b=2)`); если `true`, поле `message` содержит только текст, а структурированные значения выводятся отдельными строковыми JSON-полями (например, `{"message":"Text","a":"1","b":"2"}`). Зарезервированные имена полей получают префикс `_`.

Объекты Entry

> **Примечание**
>
> Логирование в YDB предполагает, что система состоит из большого количества внутренних компонентов, за каждым из которых закреплён его уникальный код. С полным списком компонентов и их кодов можно ознакомиться на GitHub.
>
> При записи сообщения в лог должен явно или неявно быть указан код компонента, от имени которого происходит запись этого сообщения. Впоследствии имя компонента выводится в лог и может быть использовано для поиска или фильтрации содержимого лога.

Поле `entry` позволяет задать параметры логирования для отдельных компонентов. Оно содержит массив объектов со следующей структурой:

Параметр	Тип	Описание
`component`	string	Имя компонента. С полным списком доступных компонентов можно ознакомиться на GitHub.
`level`	uint32	Уровень логирования для этого компонента.
`sampling_level`	uint32	Уровень семплирования для этого компонента. Работает аналогично `default_sampling_level`.
`sampling_rate`	uint32	Частота семплирования для этого компонента. Работает аналогично `default_sampling_rate`.

Объект UAClientConfig

Поле `uaclient_config` настраивает интеграцию с Unified Agent:

Параметр	Тип	Значение по умолчанию	Описание
`uri`	string	—	grpc URI сервера Unified Agent.
`shared_secret_key`	string	—	Путь к файлу с секретным ключом для аутентификации клиентских соединений.
`max_inflight_bytes`	uint64	100000000	Максимальное количество байт в передаче при отправке данных.
`grpc_reconnect_delay_ms`	uint64	—	Задержка между попытками переподключения в миллисекундах.
`grpc_send_delay_ms`	uint64	—	Задержка между попытками отправки в миллисекундах.
`grpc_max_message_size`	uint64	—	Максимальный размер сообщения gRPC.
`client_log_file`	string	—	Файл логов для самого клиента UA.
`client_log_priority`	uint32	—	Уровень логирования для самого клиента UA.
`log_name`	string	—	Имя лога, которое передается в метаданных сессии.

Уровни логирования

YDB использует следующие уровни логирования, перечисленные от наивысшей к наименьшей серьезности:

Уровень	Числовое значение	Описание
`EMERG`	0	Возможен сбой системы (например, отказ кластера).
`ALERT`	1	Возможна деградация системы, компоненты системы могут выйти из строя.
`CRIT`	2	Критическое состояние.
`ERROR`	3	Некритическая ошибка.
`WARN`	4	Предупреждение, на которое следует отреагировать и исправить, если оно не временное.
`NOTICE`	5	Произошло событие, существенное для системы или пользователя.
`INFO`	6	Отладочная информация для сбора статистики.
`DEBUG`	7	Отладочная информация для разработчиков.
`TRACE`	8	Очень подробная отладочная информация.

Примеры

Базовая конфигурация

```
log_config:
  default_level: 5  # NOTICE
  format: "full"
```

Данная конфигурация выводит логи в stderr с уровнем логирования `NOTICE` и выше.

Конфигурация с выводом в файл

```
log_config:
  default_level: 5  # NOTICE
  format: "full"
  backend_file_name: "/var/log/ydb/ydb.log"
```

Эта конфигурация отправляет логи в файл, сохраняя уровень логирования `NOTICE` по умолчанию.

Конфигурация с выводом в syslog

```
log_config:
  default_level: 5  # NOTICE
  sys_log: true
  sys_log_service: "ydb"
  format: "full"
```

Эта конфигурация отправляет логи в syslog с именем сервиса "ydb".

Настройка уровней логирования для отдельных компонентов

```
log_config:
  default_level: 5  # NOTICE
  entry:
    - component: "SCHEMESHARD"
      level: 7  # DEBUG
    - component: "TABLET_MAIN"
      level: 6  # INFO
  backend_file_name: "/var/log/ydb/ydb.log"
```

Включение диагностики инвалидации блокировок транзакций (TLI)

Для диагностики ошибок `transaction locks invalidated` включите логирование компонента `TLI`:

```
log_config:
  entry:
    - component: "TLI"
      level: 6  # INFO
```

После включения сервер записывает логи при каждом конфликте блокировок, включая идентификаторы и тексты запросов-жертв и нарушителей. Подробнее о формате записей, корреляции событий и утилите `find_tli_chain` см. в разделе Логирование TLI.

Конфигурация семплирования

```
log_config:
  default_level: 5  # NOTICE
  default_sampling_level: 7  # DEBUG
  default_sampling_rate: 10  # Логировать каждое 10-е сообщение между NOTICE и DEBUG
  entry:
    - component: "BLOBSTORAGE"
      sampling_level: 8  # TRACE
      sampling_rate: 100  # Логировать каждое 100-е сообщение между NOTICE и TRACE
```

Эта конфигурация настраивает семплирование логов. При настройках по умолчанию каждое 10-е сообщение с приоритетом между `NOTICE` и `DEBUG` будет записано в лог. Для компонента `BLOBSTORAGE` каждое 100-е сообщение с приоритетом между `NOTICE` и `TRACE` будет записано в лог.

Конфигурация формата JSON

```
log_config:
  default_level: 5  # NOTICE
  format: "json"
  cluster_name: "production-cluster"
  uaclient_config:
    uri: "[fd53::1]:16400"
    grpc_max_message_size: 4194304
    log_name: "ydb_logs"
```

Эта конфигурация выводит логи в формате JSON и отправляет их в Unified Agent.

Примечания

- Уровни логирования указываются в конфигурации как числовые значения, а не строки. Используйте таблицу выше для сопоставления между числовыми значениями и их значениями.
- Если указан параметр `backend_file_name`, логи записываются в этот файл. Если параметр `sys_log` имеет значение true, логи отправляются в системный регистратор.
- Параметр `format` определяет, как форматируются записи логов. Формат «full» включает всю доступную информацию, «short» предоставляет более компактный формат, а «json» выводит логи в формате JSON, который удобен для парсинга сервисами логирования.
- Внутренний буфер логов имеет следующие ограничения по размеру:
  - Общий размер по умолчанию: 10МБ (10 * 1024 * 1024 байт)
  - Размер гранулы по умолчанию: 64КБ (1024 * 64 байт)
  - Максимальный размер сообщения: 1КБ (1024 байт)

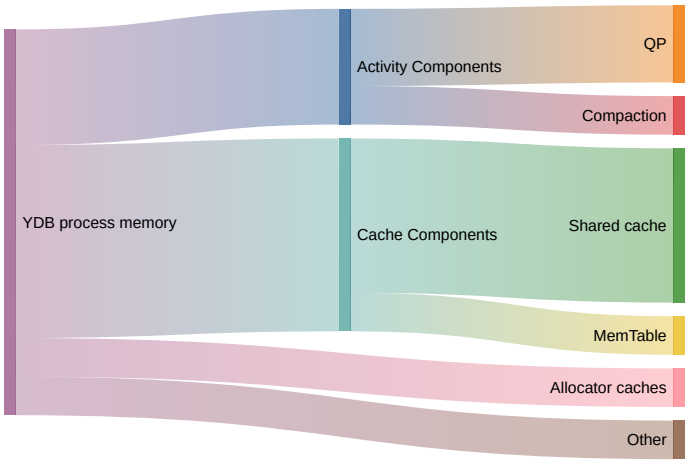
Смотрите также

- Справка по наблюдаемости YDB
- Справка по метрикам
- Трассировка в YDB
- Аудитный лог

memory_controller_config

Внутри узлов YDB работают множество различных компонентов, использующих память. Большинству из них требуется фиксированное количество памяти, но некоторые из них могут гибко варьировать объём используемой памяти, тем самым улучшая производительность всей системы.

Общий вид потребления памяти внутри узла YDB по компонентам



Если компоненты YDB выделяют больше памяти, чем физически доступно, операционная система, вероятно, завершит весь процесс YDB, что крайне нежелательно. Цель контроллера памяти — позволить YDB избегать ситуаций с нехваткой памяти, при этом эффективно используя имеющийся её объём.

Примеры компонентов, управляемых контроллером памяти:

- Общий кеш: хранит недавно доступные страницы данных, считанные из распределённого хранилища, чтобы уменьшить количество операций ввода-вывода с диска и ускорить получение данных.
- MemTable: содержит данные, которые ещё не были записаны в SST.
- Query Processor: хранит промежуточные результаты обработки запросов.
- Compaction: процесс упорядочивания и очистки данных, который выполняется автоматически (в «фоновом» режиме) для оптимизации объёма хранения данных.
- Кеши аллокатора: хранят блоки памяти, которые были освобождены, но ещё не возвращены операционной системе.

Лимиты памяти могут быть настроены для контроля общего использования памяти, обеспечивая эффективную работу базы данных в рамках доступных ресурсов.

Жёсткий лимит памяти

Жёсткий лимит памяти определяет общее количество памяти, доступное для процесса YDB.

По умолчанию жёсткий лимит памяти для процесса YDB равен лимиту памяти, указанному в его cgroups.

В окружениях без лимита памяти cgroups значение жёсткого лимита памяти по умолчанию равно общему объёму доступной памяти хоста. Эта конфигурация позволяет базе данных использовать все доступные ресурсы, но может привести к конкуренции за ресурсы с другими процессами на том же хосте. Хотя контроллер памяти пытается учесть это внешнее потребление, такое использование не рекомендуется.

Жёсткий лимит памяти также может быть задан в конфигурации. Обратите внимание, что процесс базы данных всё равно может превысить этот лимит. Поэтому настоятельно рекомендуется использовать лимиты памяти cgroups в производственных окружениях для строгого контроля памяти.

Большинство других лимитов памяти можно настроить либо в абсолютных байтах, либо в процентах относительно жёсткого лимита памяти. Использование процентов удобно для управления кластерами с узлами разной ёмкости. Если указаны как абсолютные лимиты в байтах, так и процентные лимиты, контроллер памяти использует комбинацию обоих (максимум для нижних лимитов и минимум для верхних лимитов).

Пример секции `memory_controller_config` с указанным жёстким лимитом памяти:

```
memory_controller_config:
  hard_limit_bytes: 16106127360
```

Мягкий лимит памяти

Мягкий лимит памяти определяет опасный порог, который процесс YDB не должен превышать при нормальных обстоятельствах.

Если мягкий лимит превышен, YDB постепенно уменьшает размер общего кеша до нуля. В таком случае следует как можно скорее добавить больше узлов баз данных в кластер или снизить лимиты памяти для отдельных компонентов.

Целевое использование памяти

Целевое использование памяти определяет порог использования памяти процессом YDB, который считается оптимальным.

Гибкие размеры кешей рассчитываются в соответствии с их пределами, чтобы поддерживать потребление памяти процессом вблизи этого значения.

Например, в базе данных, которая расходует немного памяти на выполнение запросов, кеши используют память вблизи этого порога, а остальная память остаётся свободной. Если выполнение запросов начинает потреблять больше памяти, кеши начинают сокращать свои размеры до минимального порога.

Лимиты памяти для отдельных компонентов

Внутри YDB существует два разных типа компонентов.

Первый тип компонентов, или кеш-компоненты, функционируют как кеши, например, храня последние использованные данные. Каждый кеш-компонент имеет минимальный и максимальный пороговые значения лимита памяти, что позволяет ему динамически изменять свою ёмкость в зависимости от текущего потребления памяти процессом YDB.

Второй тип компонентов, или компоненты-активности, выделяют память для конкретных задач, таких как выполнение запросов или процесс компактизации. Каждый компонент-активность имеет фиксированный лимит памяти. Также существует дополнительный общий лимит памяти для таких компонентов, из которого они пытаются получить необходимую память.

Многие другие вспомогательные компоненты и процессы работают параллельно с процессом YDB, потребляя память. В настоящее время эти компоненты не имеют каких-либо лимитов памяти.

Лимиты памяти для кеш-компонентов

К кеш-компонентам относятся:

- Общий кеш;
- MemTable.

Лимиты каждого кеш-компонента динамически пересчитываются каждую секунду, чтобы каждый компонент потреблял память пропорционально своим предельным значениям, а общее потребление памяти оставалось около целевого использования памяти.

Минимальный порог лимита памяти кеш-компонентов не резервируется, что означает, что память остаётся доступной до тех пор, пока она не будет фактически использована. Однако, как только эта память заполнена, компоненты обычно сохраняют данные, действуя в рамках своего текущего лимита памяти. Таким образом, ожидается, что сумма минимальных лимитов памяти кеш-компонентов будет меньше целевого использования памяти.

При необходимости следует переопределить как минимальные, так и максимальные пороговые значения; в противном случае, если пороговое значение отсутствует, оно будет иметь значение по умолчанию.

Пример секции `memory_controller_config` с указанными лимитами общего кеша:

```
memory_controller_config:
  shared_cache_min_percent: 10
  shared_cache_max_percent: 30
```

Лимиты памяти для компонентов-активностей

К компонентам-активностям относятся:

- Query Processor;
- Компактизация.

Лимит памяти для каждого из компонентов-активностей указывает максимальное количество памяти, которое он может попытаться использовать. Однако, чтобы предотвратить превышение процессом YDB мягкого лимита памяти, общее потребление компонентов-активностей ограничивается дополнительным лимитом, называемым лимитом памяти для активностей. Если общее использование памяти активными компонентами превышает этот лимит, любые дополнительные запросы на память будут отклонены. Когда выполнение запросов приближается к лимитам памяти, YDB активирует спиллинг для временного сохранения промежуточных данных на диск, предотвращая нарушение лимитов памяти.

Таким образом, хотя суммарные индивидуальные лимиты компонентов-активностей могут в совокупности превышать лимит памяти для активностей, индивидуальный лимит каждого компонента должен быть меньше этого общего предела. Кроме того, сумма минимальных лимитов памяти для кеш-компонентов плюс лимит памяти для активностей должна быть меньше мягкого лимита памяти.

Существуют и другие компоненты-активности, которые в настоящее время не имеют каких-либо индивидуальных лимитов памяти.

Пример секции `memory_controller_config` с указанным лимитом для QP:

```
memory_controller_config:
  query_execution_limit_percent: 25
```

Лимит памяти Query Processor

Параметры `query_execution_limit_percent` (по умолчанию 20% от жёсткого лимита памяти) и `query_execution_limit_bytes` задают размер пула оперативной памяти, который Resource Manager выделяет запросам на узле:

Если заданы оба параметра, эффективный лимит равен `min(query_execution_limit_percent × hard_limit_bytes / 100, query_execution_limit_bytes)`. Если задан только один параметр — используется его значение.

Вычисленный лимит передаётся в Resource Manager через Resource Broker.

Порог `spilling_percent` в `table_service_config` рассчитывается относительно этого пула.

Параметры конфигурации

Каждый параметр конфигурации применяется в контексте одного узла базы данных.

Как упоминалось ранее, ожидается, что сумма минимальных лимитов памяти для кеш-компонентов плюс лимит памяти для активностей должна быть меньше мягкого лимита памяти.

Это ограничение можно выразить в упрощённой форме:

Или в детализированной форме:

Параметры	Значение по умолчанию	Описание
`hard_limit_bytes`	CGroup memory limit / Память хоста	Жёсткий лимит использования памяти.

`soft_limit_percent` / `soft_limit_bytes`	75%	Мягкий лимит использования памяти.
`target_utilization_percent` / `target_utilization_bytes`	50%	Целевое использование памяти.
`activities_limit_percent` / `activities_limit_bytes`	30%	Лимит памяти для активностей.
`shared_cache_min_percent` / `shared_cache_min_bytes`	20%	Минимальный порог для лимита памяти общего кеша.
`shared_cache_max_percent` / `shared_cache_max_bytes`	50%	Максимальный порог для лимита памяти общего кеша.
`mem_table_min_percent` / `mem_table_min_bytes`	1%	Минимальный порог для лимита памяти MemTable.
`mem_table_max_percent` / `mem_table_max_bytes`	3%	Максимальный порог для лимита памяти MemTable.
`query_execution_limit_percent` / `query_execution_limit_bytes`	20%	Лимит памяти для QP.
`compaction_limit_percent` / `compaction_limit_bytes`	10%	Лимит памяти для компактизации.

monitoring_config

Секция `monitoring_config` файла конфигурации YDB задаёт параметры YDB Monitoring.

Аутентификация на страницах мониторинга

В этом разделе описаны настройки, связанные с аутентификацией на отдельных страницах встроенного мониторинга.

Параметр	Описание
`require_counters_authentication`	Режим обязательной аутентификации на страницах `/counters` и `/counters/hosts`. Возможные значения: • `true` — доступ к `/counters` и `/counters/hosts` только с аутентификационным токеном; запросы проходят аутентификацию и проверку прав. Значение `true` допустимо только при включенном режиме обязательной аутентификации в секции security_config файла конфигурации YDB. • `false` — запросы к `/counters` и `/counters/hosts` могут выполняться без аутентификационного токена. Значение по умолчанию: `false`.
`require_healthcheck_authentication`	Дополнительное требование аутентификации для эндпоинта `/healthcheck` поверх общих правил кластера. Возможные значения: • `true` — любой ответ `/healthcheck`, включая формат Prometheus (параметр `format=prometheus`), выдаётся только при запросе с аутентификационным токеном; запросы проходят аутентификацию и проверку прав. Значение `true` допустимо только при включенном режиме обязательной аутентификации в секции security_config файла конфигурации YDB. • `false` — при обязательной аутентификации в кластере запросы к `/healthcheck` без токена по-прежнему допускаются, если запрошен вывод в формате Prometheus (`format=prometheus`). Для остальных форматов ответа `/healthcheck` действуют общие правила (см. примечание ниже). Значение по умолчанию: `false`. **Примечание** Если в security_config включена обязательная аутентификация, то для ответов `/healthcheck` в любом формате, кроме Prometheus, токен обязателен независимо от значения `require_healthcheck_authentication`.

Пример задания параметров с включённой аутентификацией на отдельных страницах встроенного мониторинга.

```
monitoring_config:
  # аутентификация на страницах /counters и /healthcheck
  require_counters_authentication: true
  require_healthcheck_authentication: true
```

TLS на страницах мониторинга

YDB открывает отдельный HTTP-порт для работы YDB UI, отображения метрик и других вспомогательных команд.

На HTTP-порту можно включить TLS, таким образом порт начнёт принимать только HTTPS-соединения. Обычные HTTP-запросы при этом будут отклоняться на уровне TLS-рукопожатия без какого-либо ответа и без редиректа на HTTPS. Одновременная работа HTTP и HTTPS на разных портах не поддерживается: мониторинг использует один порт, который работает только как HTTP либо только как HTTPS.

TLS включается указанием SSL-сертификата (поддерживаются цепочки сертификатов) и приватного SSL-ключа (поддерживаются ключи следующих типов: RSA, ECDSA и PKCS#8) без пароля, т.к. зашифрованные ключи не поддерживаются.

Ниже описаны параметры TLS для мониторинга. При смене значения этих параметров необходимо перезапустить узлы кластера, на которых были сделаны изменения.

Параметр	Описание
`monitoring_certificate`	Параметр для передачи содержимого SSL-сертификата и приватного SSL-ключа напрямую в формате PEM без использования отдельных файлов. Требования к содержимому параметра: • сначала указывается сертификат сервера; • затем последовательно указываются промежуточные сертификаты при их наличии; • затем указывается приватный ключ без пароля. При указании этого параметра встроенный интерфейс автоматически начинает обрабатывать запросы с использованием указанного SSL-сертификата. Если указан параметр `monitoring_certificate`, параметры `monitoring_certificate_file` и `monitoring_private_key_file` игнорируются. Значение по умолчанию: пустая строка.

`monitoring_certificate_file`	Путь к файлу сертификата для доступа по SSL в формате PEM. Файл может дополнительно содержать приватный SSL-ключ без пароля. Этот приватный ключ будет использоваться в случае, если не указан параметр `monitoring_private_key_file`. При указании параметра `monitoring_certificate_file` встроенный интерфейс автоматически начинает обрабатывать запросы с использованием указанного SSL-сертификата. Значение по умолчанию: пустая строка.
`monitoring_private_key_file`	Путь к файлу приватного SSL-ключа в формате PEM без пароля. При указании этого параметра должен быть задан параметр `monitoring_certificate_file`. Если в файле, указанном в параметре `monitoring_certificate_file`, содержится приватный SSL-ключ, он будет проигнорирован, то есть приоритет в задании приватного ключа имеет параметр `monitoring_private_key_file`. Значение по умолчанию: пустая строка.
`monitoring_ca_file`	Путь к файлу с корневым (CA) сертификатом. Включает запрос клиентского сертификата при TLS-handshake для мониторинга. Возможные значения: • Непустой путь — сервер запрашивает клиентский сертификат. Если сертификат предъявлен, он проверяется на TLS-уровне при аутентификации устройств (соединение с недоверенным сертификатом не устанавливается); после успешно пройденной проверки сервер приоритетно использует аутентификационный токен для аутентификации клиента, а при отсутствии токена выполняется аутентификация по клиентскому сертификату. Если сертификат не предъявлен, то при настройке `client_certificate_required: true` соединение не устанавливается. • Пустая строка — сервер не запрашивает клиентский сертификат при TLS-handshake; аутентификация устройств и аутентификация по клиентскому сертификату для мониторинга при такой настройке недоступны. Параметр игнорируется, если параметрами `monitoring_certificate` или `monitoring_certificate_file` + `monitoring_private_key_file` не включен TLS. Значение по умолчанию: пустая строка.
`client_certificate_required`	Требование клиентского сертификата при TLS-handshake для мониторинга. Возможные значения: • `true` — сервер требует клиентский сертификат: соединение без сертификата или с недоверенным сертификатом не устанавливается. Можно указать `true` только вместе с заданным `monitoring_ca_file`. • `false` — наличие клиентского сертификата не обязательно; поведение при запросе сертификата определяется параметром `monitoring_ca_file`. Значение по умолчанию: `false`.

Пример включения mTLS для мониторинга: запрос клиентского сертификата без обязательности его предъявления.

```
monitoring_config:
  monitoring_certificate_file: /path/to/cert.pem # для включения TLS
  monitoring_private_key_file: /path/to/key.pem
  monitoring_ca_file: /path/to/ca.pem # запрос клиентского сертификата (mTLS)
```

Пример включения mTLS для мониторинга: требование клиентского сертификата с его обязательным предъявлением.

```
monitoring_config:
  monitoring_certificate_file: /path/to/cert.pem
  monitoring_private_key_file: /path/to/key.pem
  monitoring_ca_file: /path/to/ca.pem
  client_certificate_required: true
```

node_broker_config

Секция `node_broker_config` настраивает стабильные имена узлов для динамических узлов в кластерах YDB. Присвоение имен узлам осуществляет Node Broker — системная таблетка, которая отвечает за регистрацию динамических узлов в кластере.

Node Broker присваивает имена динамическим узлам при их регистрации в кластере. По умолчанию имя узла состоит из имени хоста и номера порта, на котором работает узел.

В динамической среде, где имена хостов часто меняются, например в Kubernetes, использование имени хоста и порта приводит к неконтролируемому росту количества уникальных имен узлов, даже для базы данных с небольшим количеством динамических узлов. Подобное поведение может быть нежелательным для системы time series мониторинга — количество метрик растет некотролируемо. Для решения этой проблемы администратор системы может настроить *стабильные* имена узлов.

Стабильное имя идентифицирует узел в рамках тенанта. Оно состоит из префикса и порядкового номера узла в рамках тенанта. Если динамический узел был выключен, то по истечении таймаута его стабильное имя может быть занято новым динамическим узлом, обслуживающим того же тенанта.

Чтобы включить стабильные имена узлов, установите фича-флаг `enable_stable_node_names` в конфигурации кластера:

```
feature_flags:
  enable_stable_node_names: true
```

По умолчанию, префиксом является `slot-`. Для того, чтобы переопределить префикс, необходимо добавить в конфигурацию кластера следующее:

```
node_broker_config:
  stable_node_name_prefix: <новый префикс>
```

resource_broker_config

Секция `resource_broker_config` настраивает брокер ресурсов — акторный сервис, контролирующий потребление ресурсов узла YDB, таких как:

- `CPU` — количество потоков;
- `Memory` — оперативная память.

Разные виды активностей (фоновые операции, удаление данных по TTL и т.д.) запускаются в разных *очередях* брокера ресурсов. Каждая такая очередь имеет лимитированное число ресурсов:

Название очереди	CPU	Memory	Описание
`queue_ttl`	2	—	Операции удаления данных по TTL.
`queue_backup`	2	—	Операции резервного копирования.
`queue_restore`	10	—	Операции восстановления из резервной копии.
`queue_build_index`	10	—	Операции онлайн-создания вторичного индекса.
`queue_cdc_initial_scan`	2	—	Первоначальное сканирование таблицы.

Примечание

Рекомендуется **дополнять** конфигурацию брокера ресурсов, используя теги `!inherit` и `!append`.

Пример дополнения конфигурации брокера ресурсов пользовательским лимитом для очереди `queue_ttl`:

```
resource_broker_config: !inherit
  queues: !append
  - name: queue_ttl
    limit:
      cpu: 4
```

security_config

В разделе `security_config` файла конфигурации YDB задаются режимы аутентификации, первичная конфигурация локальных пользователей и групп и их права.

```
security_config:
  # настройка режима аутентификации
  enforce_user_token_requirement: false
  enforce_user_token_check_requirement: false
  default_user_sids: <SID для анонимных запросов>
  all_authenticated_users: <имя группы всех аутентифицированных пользователей>
  all_users_group: <имя группы всех пользователей>

  # первичные настройки безопасности
  default_users: <список пользователей по умолчанию>
  default_groups: <список групп по умолчанию>
  default_access: <список прав по умолчанию на корне кластера>

  # настройки списков доступа
  database_allowed_sids: <список SID'ов, которым разрешён просмотр состояния базы данных>
  viewer_allowed_sids: <список SID'ов с правами просмотра состояния кластера>
  monitoring_allowed_sids: <список SID'ов с правами мониторинга и изменения состояния кластера>
  administration_allowed_sids: <список SID'ов с правами администрирования кластера>
  bootstrap_allowed_sids: <список SID'ов, которым разрешена начальная инициализация кластера>
  register_dynamic_node_allowed_sids: <список SID'ов с правами регистрации узлов баз данных в кластере>

  # настройки встроенной настройки безопасности
  disable_builtin_security: false
  disable_builtin_groups: false
  disable_builtin_access: false

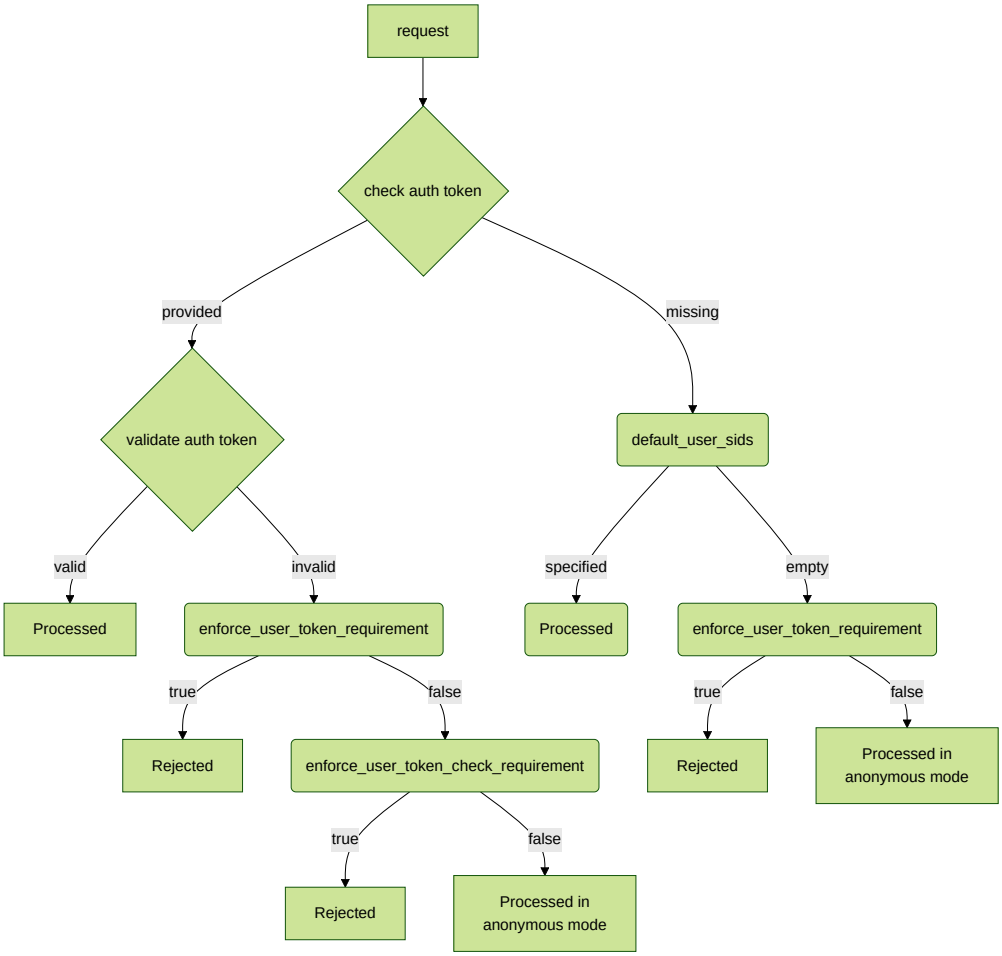
  # настройки сокрытия ошибок доступа
  hide_authentication_failure_reasons: true
```

Настройки режима аутентификации

Параметр	Описание
`enforce_user_token_requirement`	Режим обязательной аутентификации. Возможные значения: • `true` — аутентификация обязательна, запросы к YDB обязаны сопровождаться аутентификационным токеном. Запросы проходят аутентификацию и проверку прав. • `false` — аутентификация опциональна, запросы к YDB могут не сопровождаться аутентификационным токеном. Запросы без токена выполняются в анонимном режиме и без проверки прав. Запросы с токеном проходят аутентификацию и проверку прав. Но в случае ошибок аутентификации, запросы не запрещаются, а выполняются в анонимном режиме. При `enforce_user_token_check_requirement: true` выполнение запросов с ошибкой аутентификации запрещается. Взамен отсутствующего в запросах токена используется значение параметра `default_user_sids`, если параметр определён и не пустой (см. описание ниже). Тогда аутентификация и проверка прав проводится для субъекта доступа, заданного `default_user_sids`. Значение по умолчанию: `false`.
`enforce_user_token_check_requirement`	Запрещает игнорировать ошибки аутентификации в режиме `enforce_user_token_requirement: false`. Значение по умолчанию: `false`.
`default_user_sids`	Список SID'ов для использования в процессе аутентификации в случае, когда входящий запрос не сопровождается явным аутентификационным токеном. `default_user_sids` используется для анонимных запросов. Первым элементом в списке должен быть SID пользователя, за ним должны идти SID'ы групп, к которым пользователь принадлежит. Непустой `default_user_sids` позволяет использовать режим обязательной аутентификации (`enforce_user_token_requirement: true`) вместе с анонимными запросами. Это может быть полезно в определённых сценариях тестирования YDB или в ознакомительных целях для локальных баз данных. Значение по умолчанию: пустой список.

`all_authenticated_users`	Имя виртуальной группы, в которой состоят все аутентифицированные пользователи. Виртуальную группу не нужно явно создавать, она ведётся системой автоматически. Виртуальную группу нельзя удалить, нельзя получить или изменить список её членов. Пользователь может использовать её для выдачи прав на схемных объектах. **Совет** Информацию о правах доступа к схемным объектам можно получить из системных представлений YDB, см. {#T}. Значение по умолчанию: `all-users@well-known`.
`all_users_group`	Имя группы, в которую должны добавляться все внутренние пользователи. Если `all_users_group` не пустая, то все локальные пользователи в момент создания будут добавляться в группу с указанным именем. В момент создания пользователей группа, указанная в этом параметре, должна существовать. `all_users_group` автоматически конфигурируется при выполнении встроенной настройки безопасности. Значение по умолчанию: пустая строка.

Следующая диаграмма показывает как взаимодействуют параметры настройки режима аутентификации:



Первичные настройки безопасности

Параметры `default_users`, `default_groups`, `default_access` влияют на настройку кластера, осуществляемую при первом старте YDB. При последующих запусках первичная настройка не выполняется, эти параметры игнорируются.

См. также раздел по встроенной настройке безопасности и влияющие на неё настройки уровня `domains_config`.

Параметр	Описание

`default_users`	Какие пользователи должны быть созданы на кластере при первом запуске.
	
	Список пар логин-пароль. Первый пользователь становится суперпользователем.
	
	Примечание
	Пароли задаются в открытом виде и оставлять их действующими на долгое время небезопасно. Поэтому после первого запуска кластера и его настройки рекомендуется пароли стартовым пользователям поменять средствами YDB (например, `ALTER USER`).
	
	Пример:
	
	`default_users:`
	`- name: root`
	`  password: <...>`
	`- name: user1`
	`  password: <...>`
	
	Ошибки в списке (повторение логинов) фиксируются в логе, но не влияют на запуск кластера.

`default_groups`	Какие группы должны быть созданы на кластере при первом запуске.
	
	Список групп и их членов.
	
	Важно
	Эти группы создаются для всего кластера YDB.
	
	Пример:
	
	`default_groups:`
	`- name: ADMINS`
	`  members: root`
	`- name: USERS`
	`  members:`
	`  - ADMINS`
	`  - root`
	`  - user1`
	
	Порядок перечисления групп важен: группы создаются в порядке перечисления, и указанные члены группы к моменту создания группы должны существовать, иначе они не будут добавлены в группу.
	
	Ошибки добавления членов групп фиксируются в логе, но не влияют на запуск кластера.

`default_access`	Какие права должны быть выданы на корне кластера.
	
	Список разрешений в формате краткой записи управления доступом.
	
	Пример:
	
	`default_access:`
	`- +(CDB|DDB|GAR):ADMINS`
	`- +(ConnDB):USERS`
	
	Ошибки в строках прав доступа фиксируются в логе, но не влияют на запуск кластера. Право доступа с ошибкой не будет добавлено.

Настройки административных и других привилегий

Списки уровней доступа настраиваются в секции `security_config`. Подробная информация о списках уровней доступа, их иерархии и принципах работы приведена в разделе Списки уровней доступа документации по авторизации.

Параметр	Описание
`database_allowed_sids`	Список SID'ов с уровнем доступа database. Этот уровень менее привилегирован, чем viewer. Субъекты с таким уровнем могут работать только в контексте конкретной базы данных — backend-вызовы без указания базы запрещены. Запросы уровня кластера (например, получение списка узлов кластера) возвращают 403. Запросы в контексте базы данных (например, страницы базы в YDB UI) работают штатно. Предназначен для приложений или пользователей, ограниченных одной базой данных.
`viewer_allowed_sids`	Список SID'ов с уровнем доступа viewer. Даёт возможность просматривать состояние кластера, недоступное публично (включая большую часть страниц YDB UI), без возможности вносить изменения.
`monitoring_allowed_sids`	Список SID'ов с уровнем доступа monitoring. Даёт дополнительные привилегии для мониторинга и изменения состояния кластера. Например, позволяет выполнять резервное копирование, восстановление базы данных или запускать YQL-запросы через YDB UI.

`administration_allowed_sids`	Список SID'ов с уровнем доступа administration. Даёт привилегии для администрирования кластера YDB и его баз данных. Также используется для изменения конфигурации, схемных операций, требующих административных прав, и других административных проверок. Пустой список означает, что любой пользователь считается администратором.
`bootstrap_allowed_sids`	Список SID'ов, которым разрешено выполнять начальную инициализацию кластера. Используется для первоначальной инициализации кластера из неинициализированного состояния, когда подсистема аутентификации ещё не функционирует. Начальная инициализация разрешена, если субъект входит в `bootstrap_allowed_sids` или `administration_allowed_sids`. `bootstrap_allowed_sids` позволяет выделить отдельные идентичности для начальной инициализации без выдачи им полных административных привилегий.
`register_dynamic_node_allowed_sids`	Список SID'ов, которым разрешена регистрация узлов баз данных. По техническим причинам этот список должен включать `root@builtin`.

Важно

По умолчанию списки уровней доступа пусты.

Пустой список означает «разрешить любому пользователю»: подходит любой SID, а если разрешён анонимный доступ — даже отсутствие SID. Пустой `administration_allowed_sids` даёт любому пользователю права администратора; пустой `viewer_allowed_sids` (если остальные UI-списки тоже пусты) даёт любому пользователю viewer-доступ к YDB UI. Если все четыре иерархических списка пусты, любой пользователь получает полный административный доступ.

Для защищённого развёртывания YDB важно заранее спланировать модель доступа и определить списки групп до первого запуска кластера.

Списки уровней доступа могут содержать SID пользователей или групп. Субъект соответствует списку, если в нём присутствует его пользовательский SID или любой SID одной из его групп, включая вложенные группы — достаточно одного совпадения.

Рекомендуется включать в списки уровней доступа `*_allowed_sids` группы пользователей и отдельные сервисные аккаунты. Тогда выдача уровней доступа отдельным пользователям не потребует изменения конфигурации кластера YDB.

Где используются списки:

- **`database_allowed_sids`**, **`viewer_allowed_sids`**, **`monitoring_allowed_sids`**, **`administration_allowed_sids`**
  Для доступа к YDB UI и viewer HTTP endpoints.
- **`administration_allowed_sids`**
  Для всех административных проверок.
- **`bootstrap_allowed_sids`**
  Только для операции начальной инициализации кластера.
- **`register_dynamic_node_allowed_sids`**
  Для регистрации узлов в кластере (discovery service и подсистема cms).

Примечание

Списки уровней доступа представляют собой слои дополнительных привилегий:

- Субъект доступа, не включённый ни в один список уровней доступа, может просматривать только публично доступную информацию о кластере (например, список баз данных в кластере или список узлов кластера).
- Список `database_allowed_sids` определяет роль «database»: такие пользователи менее привилегированы, чем viewers, и не могут выполнять backend-вызовы без указания базы данных; общекластерные запросы для них недоступны.
- Списки `viewer_allowed_sids`, `monitoring_allowed_sids` и `administration_allowed_sids` образуют иерархию: administration включает monitoring и viewer, a monitoring включает viewer. Субъекту достаточно находиться только в одном списке, соответствующем его роли — включение в более высокий список автоматически даёт все более низкие привилегии.

Например:

- пользователя базы данных (например, приложение, которое должно работать только в рамках одной базы) следует добавлять только в `database_allowed_sids`;
- наблюдателя следует добавлять только в `viewer_allowed_sids`;
- пользователя с уровнем monitoring следует добавлять только в `monitoring_allowed_sids` (он автоматически получает viewer-привилегии);
- администратора следует добавлять только в `administration_allowed_sids` (он автоматически получает monitoring- и viewer-привилегии).

Настройки встроенной настройки безопасности

Флаги `disable_builtin_security`, `disable_builtin_groups`, `disable_builtin_access` влияют на настройку кластера, осуществляемую только при первом старте кластера YDB.

Параметр	Описание

`disable_builtin_security`	Не выполнять встроенную настройку безопасности. Встроенная настройка включает автоматическое создание суперпользователя `root`, набора встроенных пользовательских групп и выдачу прав доступа этим группам на корне кластера. Эфемерный флаг, не попадает в конфигурацию, сохраняемую в кластере. Значение по умолчанию: `false`.
`disable_builtin_groups`	Отказаться от создания встроенных групп, даже если явные группы по умолчанию (`security_config.default_groups`) заданы. Значение по умолчанию: `false`
`disable_builtin_access`	Отказаться от добавления прав на корне кластера для встроенных групп, даже если явные права по умолчанию (`security_config.default_access`) заданы. Значение по умолчанию: `false`

Настройки сокрытия ошибок доступа

Параметр	Описание
`hide_authentication_failure_reasons`	Скрывает от пользователя причины ошибок аутентификации. Позволяет предотвратить утечку информации о внутреннем устройстве и содержимом систем аутентификации YDB. Например, усложняет для злоумышленника задачу выявления существующих пользователей методом перебора. Значение по умолчанию: `true`.

self_management_config

Секция `self_management_config` настраивает распределённую конфигурацию V2 и автоматическое управление компонентами кластера.

Синтаксис

```
self_management_config:
  enabled: true
  automatic_static_group_management: true
  static_group_self_heal_allowed_nodes:
  - 1
  - 2
  - 3
```

Параметры

Параметр	Значение по умолчанию	Описание
`enabled`	`false`	Включает механизм распределённой конфигурации V2.
`automatic_static_group_management`	`false`	Разрешает распределённой конфигурации автоматически изменять размещение VDisk статической группы.
`static_group_self_heal_allowed_nodes`	`[]`	Ограничивает набор узлов, на которые SelfHeal может переносить VDisk статической группы. Укажите список идентификаторов узлов. Пустой список означает отсутствие ограничений.

system_tablet_backup_config

Механизм резервного копирования системных таблеток обеспечивает инкрементальное копирование метаданных кластера — таких как Hive, BSController и SchemeShard — на локальную файловую систему хостов кластера. Поведение резервного копирования конфигурируется в секции `system_tablet_backup_config` конфигурации YDB.

Синтаксис

```
system_tablet_backup_config:
    filesystem:
        path: "/path/to/backup/directory"
    exclude_tablet_ids:
        - 72057594037968897
    max_backups_limit: 3
```

Параметры

Параметр	Значение по умолчанию	Описание
`filesystem.path`	—	Абсолютный путь к директории на локальной файловой системе хоста для хранения резервных копий. Директория должна быть доступна для записи процессом YDB на каждом хосте, где работают системные таблетки. Поддерживаются только локальные файловые системы; сетевые файловые системы, такие как NFS, не поддерживаются.
`exclude_tablet_ids`	—	Список идентификаторов системных таблеток, которые необходимо исключить из резервного копирования, например, для снижения нагрузки на диски или экономии места.
`max_backups_limit`	`3`	Максимальное количество резервных копий одной таблетки, хранимых на хосте. После успешного создания новой копии самая старая автоматически удаляется при превышении лимита.

tls

Секция `tls` настраивает параметры TLS для шифрования данных при передаче по сети в YDB. Каждый сетевой протокол может иметь различные настройки TLS для обеспечения безопасной связи между компонентами кластера и клиентами.

Interconnect

Интерконнект акторной системы YDB — это специализированный протокол для обмена данными между узлами YDB.

Пример включения TLS для интерконнекта:

```
interconnect_config:
   start_tcp: true
   encryption_mode: REQUIRED # или OPTIONAL
   path_to_certificate_file: "/opt/ydb/certs/node.crt"
   path_to_private_key_file: "/opt/ydb/certs/node.key"
   path_to_ca_file: "/opt/ydb/certs/ca.crt"
```

YDB в роли сервера

gRPC

Основной API YDB основан на gRPC. Он используется для внешнего взаимодействия с клиентскими приложениями, которые работают напрямую с YDB через SDK или CLI.

Пример включения TLS для gRPC API:

```
grpc_config:
   cert: "/opt/ydb/certs/node.crt"
   key: "/opt/ydb/certs/node.key"
   ca: "/opt/ydb/certs/ca.crt"
```

Протокол Kafka

YDB открывает отдельный сетевой порт для протокола Kafka. Этот протокол используется для внешнего взаимодействия с клиентскими приложениями, изначально разработанными для работы с Apache Kafka.

Пример включения TLS для протокола Kafka с использованием файла, содержащего как сертификат, так и закрытый ключ:

```
kafka_proxy_config:
    ssl_certificate: "/opt/ydb/certs/node.crt"
```

Пример включения TLS для протокола Kafka с раздельными файлами сертификата и закрытого ключа:

```
kafka_proxy_config:
    cert: "/opt/ydb/certs/node.crt"
    key: "/opt/ydb/certs/node.key"
```

HTTP

YDB открывает отдельный HTTP-порт для работы YDB UI, отображения метрик и других вспомогательных команд.

Пример включения TLS на HTTP-порту, что делает его использования HTTPS:

```
monitoring_config:
    monitoring_certificate_file: "/opt/ydb/certs/node.crt"
```

Подробное описание параметров TLS для мониторинга см. в разделе monitoring_config.

YDB в роли клиента

LDAP

YDB поддерживает LDAP для аутентификации пользователей. Протокол LDAP имеет два варианта включения TLS.

Пример включения TLS для LDAP через расширение протокола `StartTls`:

```
auth_config:
  ldap_authentication:
    use_tls:
      enable: true
      ca_cert_file: "/path/to/ca.pem"
      cert_require: DEMAND
  scheme: "ldap"
```

Пример включения TLS для LDAP через `ldaps`:

```
auth_config:
  ldap_authentication:
    use_tls:
      enable: false
      ca_cert_file: "/path/to/ca.pem"
      cert_require: DEMAND
  scheme: "ldaps"
```

Подробнее этот механизм описан в Конфигурация LDAP аутентификации.

Федеративные запросы

Федеративные запросы позволяют YDB выполнять запросы к различным внешним источникам данных. Использование TLS при выполнении таких запросов контролируется параметром `USE_TLS` в запросах CREATE EXTERNAL DATA SOURCE. Изменения в серверной конфигурации не требуются.

Трассировка

YDB может отправлять данные трассировки на внешний коллектор через gRPC.

Пример включения TLS для данных трассировки посредством указания протокола `grpcs://`:

```
tracing_config:
  backend:
    opentelemetry:
      collector_url: grpcs://example.com:4317
      service_name: ydb
```

Асинхронная репликация

Асинхронная репликация синхронизирует данные между двумя базами данных YDB, одна из них выступает в роли клиента по отношению к другой. Использование TLS при такой коммуникации контролируется параметром `CONNECTION_STRING` в запросах CREATE ASYNC REPLICATION. Для TLS-соединений используйте протокол `grpcs://`. Изменения в серверной конфигурации не требуются.

При использовании пользовательского удостоверяющего центра (Certificate Authority, CA) передайте его сертификат в параметре `CA_CERT` при создании экземпляра асинхронной репликации.

table_service_config

Секция `table_service_config` содержит параметры конфигурации для сервиса таблиц, включая настройки спиллинга.

resource_manager

В подсекции `resource_manager` задаются параметры управления ресурсами сервиса таблиц.

Кэш level table векторного индекса

Level table (`indexImplLevelTable`) векторного индекса `vector_kmeans_tree` хранит центроиды, к которым YDB обращается на каждом шаге спуска по дереву кластеров во время векторного поиска. Чтобы не перечитывать эти центроиды из распределённого хранилища при каждом запросе, YDB может кэшировать их в памяти на каждом узле базы данных.

Кэш создаётся отдельно на каждом узле, использует LRU-вытеснение и наращивает свой размер до указанного лимита постепенно.

```
table_service_config:
  resource_manager:
    kqp_level_cache_max_size_bytes: 0
    kqp_level_cache_increase_batch_size_bytes: 33554432
```

resource_manager.kqp_level_cache_max_size_bytes

Тип: `uint64`
По умолчанию: `0` (кэш отключён)
Описание: Максимальный размер кэша level table векторного индекса в байтах на один узел базы данных. Установка ненулевого значения включает кэш.

resource_manager.kqp_level_cache_increase_batch_size_bytes

Тип: `uint64`
По умолчанию: `33554432` (32 МиБ)
Описание: Шаг увеличения размера кэша level table. Кэш растёт порциями такого размера до значения `kqp_level_cache_max_size_bytes` и уменьшается такими же порциями при снижении лимита.

Лимиты памяти выполнения запросов

В подсекции `resource_manager` задаётся порог спиллинга относительно пула памяти запросов на узле. Размер этого пула регулируется параметрами `query_execution_limit_percent` / `query_execution_limit_bytes` в `memory_controller_config`.

```
table_service_config:
  resource_manager:
    spilling_percent: 80
```

resource_manager.spilling_percent

Тип: `double`
По умолчанию: `80`
Описание: Порог заполнения пула памяти запросов, при котором YDB начинает считать спиллинг предпочтительным способом управления памятью. Базой для расчёта служит пул Query Processor, размер которого задаётся `query_execution_limit_percent` / `query_execution_limit_bytes`.

Когда суммарное потребление памяти запросами на узле превышает `spilling_percent` процентов от доступного пула, вычислительные операции, поддерживающие спиллинг (Grace Hash Join, агрегации и др.), получают сигнал выгружать промежуточные данные на диск вместо дальнейшего наращивания потребления RAM.

Например, при значении по умолчанию `80` спиллинг активируется, когда пул запросов заполнен примерно на 80%.

Порог применяется к общему пулу памяти запросов на узле (размер — см. `query_execution_limit_percent` / `query_execution_limit_bytes`).

> **Примечание**
>
> `spilling_percent` не задаёт лимит на объём файлов спиллинга на диске. За дисковые квоты отвечает `local_file_config.max_total_size` в секции `spilling_service_config`.

Взаимодействие с другими параметрами

Параметр	Уровень	Роль
`query_execution_limit_percent` / `query_execution_limit_bytes`	Узел (QP)	Размер пула памяти запросов
`spilling_percent`	Запрос / пул	Порог заполнения пула, после которого предпочтителен спиллинг
`activities_limit_percent`	Узел	Общий лимит памяти для всех компонентов-активностей (QP, компактизация и др.)

`spilling_percent` определяет, когда пул запросов на узле настолько заполнен, что дальнейший рост в RAM должен уступить место спиллингу. `activities_limit_percent` ограничивает память активностей в целом и косвенно влияет на доступный объём RAM, но не заменяет пул, относительно которого считается `spilling_percent`.

Рекомендации

- Уменьшайте `spilling_percent` (например, до `70`), если нужно раньше переводить тяжёлые запросы на диск и снизить риск исчерпания пула памяти;

- Увеличивайте `spilling_percent` (например, до `90`), если дисковый спиллинг слишком часто снижает производительность, а на узле достаточно RAM;
- Согласовывайте `spilling_percent` с `query_execution_limit_percent` / `query_execution_limit_bytes`: при увеличении лимита пула запросов можно поднять порог спиллинга, если на узле достаточно RAM.

spilling_service_config

Спиллинг — это механизм управления памятью в YDB, который временно сохраняет данные на диск при нехватке оперативной памяти.

Включение

Спиллинг включён по умолчанию. Следующий параметр управляет включением и отключением сервиса спиллинга.

local_file_config.enable

Расположение: `table_service_config.spilling_service_config.local_file_config.enable`
Тип: `boolean`
По умолчанию: `true`
Описание: Включает или отключает сервис спиллинга. При отключении (`false`) спиллинг не функционирует, что может привести к ошибкам при обработке больших объёмов данных.

Возможные ошибки

- `Spilling Service not started` / `Service not started` — попытка использования спиллинга при выключенном Spilling Service. См. Spilling Service not started

```
table_service_config:
  spilling_service_config:
    local_file_config:
      enable: true
```

Основные параметры конфигурации

```
table_service_config:
  spilling_service_config:
    local_file_config:
      root: ""
      max_total_size: 21474836480
```

Конфигурация директории

local_file_config.root

Тип: `string`
По умолчанию: `""` (временная директория)
Описание: Файловая директория для сохранения файлов спиллинга. Директория должна существовать заранее и быть доступна процессу `ydbd` на чтение и запись: YDB её не создаёт — см. Требования к директории root. Если значение не задано, используется временная директория ОС — см. Директория по умолчанию.

Для каждого процесса `ydbd` внутри `root` создается отдельная директория:

`spilling-tmp-<node_id>-<spilling_service_id>-<username>`

Где:

- `node_id` — идентификатор узла
- `spilling_service_id` — уникальный идентификатор экземпляра (GUID), который создаётся при инициализации Spilling Service один раз при запуске процесса ydbd
- `username` — имя пользователя ОС, под которым запускается процесс `ydbd`

Файлы спиллинга хранятся внутри каждой такой директории.

Пример полного пути к директории спиллинга:

```
/tmp/spilling-tmp-1-1144b692-f1e5d361-f3960fe8-f607582e-kikimr
```

Где:

- `/tmp` — значение параметра `root` (или системная временная директория, если `root` не задан)
- `1` — идентификатор узла
- `1144b692-f1e5d361-f3960fe8-f607582e` — идентификатор экземпляра Spilling Service
- `kikimr` — имя пользователя ОС

> **Примечание**
>
> Спиллинг выполняется только на узлах базы данных.

Требования к директории root

YDB не создаёт директорию `root`: она должна существовать до запуска узлов. Создать её заранее должен администратор кластера или система развёртывания. Процесс `ydbd` создаёт внутри `root` только свою поддиректорию.

Всем пользователям ОС, под которыми запускаются узлы базы данных, нужен доступ на чтение и запись в `root`: узел создаёт там свою поддиректорию, пишет в неё файлы спиллинга и читает их. Если статические и динамические узлы запускаются под разными пользователями ОС и используют один и тот же `root`, права нужны каждому из этих пользователей — например, режим `1777`, где sticky-бит не даёт пользователям удалять чужие поддиректории.

Важно

Если спиллинг включён, а директория `root` отсутствует или недоступна для чтения и записи, узел базы данных не запустится. Убедитесь, что `root` существует и доступен на всех узлах базы данных до включения спиллинга.

Директория по умолчанию

Если `root` не задан, используется временная директория ОС того процесса `ydbd`, который запускает узел:

- если в окружении процесса задана переменная `TMPDIR` — берётся её значение;
- иначе используется `/tmp`.

Требования, описанные выше, относятся и к этой директории, но обычно `/tmp` уже существует с режимом `1777` и подходит без дополнительной настройки.

Тем не менее рекомендуется задавать `root` явно — отдельной директорией и лучше на отдельном диске.

Очистка директорий при старте

Каждый запуск процесса пишет в свою директорию с новым `spilling_service_id`, а файлы спиллинга нужны только во время работы процесса, поэтому директории прошлых запусков удаляются. Очистка выполняется при старте Spilling Service асинхронно.

Удаляются директории, у которых `node_id` и `username` совпадают с текущим процессом, а `spilling_service_id` отличается от идентификатора текущего запуска. Так после аварийного завершения процесса (например, `SIGKILL` или отключение питания) место освобождается при следующем старте узла.

Не удаляются:

- сам `root` — YDB его не создаёт и не удаляет, а только пишет внутрь;
- директории других узлов (`spilling-tmp-<other_node_id>-...`) — эти узлы могут работать прямо сейчас;
- директории с тем же `node_id`, но другим `username` — под этим пользователем ОС может работать другой процесс `ydbd`;
- любые посторонние файлы и директории внутри `root` — имя должно полностью соответствовать формату, в том числе содержать `spilling_service_id` в виде GUID.

Примечание

В предыдущих версиях YDB директории назывались `node_<node_id>_<spilling_service_id>` и создавались либо непосредственно в `root` (если он задан), либо в `spilling-tmp-<username>` внутри системной временной директории (если `root` не задан). Такие директории этого узла тоже удаляются при старте, причём оба расположения проверяются независимо от текущего значения `root`: директории, оставшиеся во временной директории, будут удалены, даже если `root` уже задан явно. Общая директория `spilling-tmp-<username>` и директории `node_<other_node_id>_*` других узлов сохраняются: во время обновления кластера соседние узлы ещё могут писать туда в старом формате.

Возможные ошибки

- `Permission denied` — недостаточные права доступа к директории. См. Permission denied

local_file_config.max_total_size

Тип: `uint64`
По умолчанию: `21474836480` (20 GiB)
Описание: Максимальный суммарный размер всех файлов спиллинга на каждом узле. При превышении лимита операции спиллинга завершаются ошибкой. Общий лимит спиллинга во всем кластере равен сумме значений `max_total_size` со всех узлов.

Рекомендации

- Устанавливайте значение исходя из доступного дискового пространства

Возможные ошибки

- `Total size limit exceeded: X/YMb` — превышен максимальный суммарный размер файлов спиллинга. См. Total size limit exceeded

Управление памятью

Связь с memory_controller_config

Активация спиллинга тесно связана с настройками контроллера памяти. Подробная конфигурация `memory_controller_config` описана в отдельной статье.

Ключевым параметром для спиллинга является **`activities_limit_percent`**, который определяет объем памяти, выделяемый для активностей по обработке запросов. От этого параметра зависит доступная память для пользовательских запросов и, соответственно, частота активации спиллинга.

Порог спиллинга в resource_manager

Непосредственный порог, при котором вычислительные операции переключаются на спиллинг, регулируется параметром `resource_manager.spilling_percent`. Он определяет, при каком заполнении пула памяти Query Processor промежуточные данные запроса начинают выгружаться на диск. Размер пула задаётся в `memory_controller_config`. Подробнее см. раздел Лимиты памяти выполнения запросов.

Влияние на спиллинг

- При увеличении `activities_limit_percent` больше памяти доступно для запросов → спиллинг активируется реже
- При уменьшении `activities_limit_percent` меньше памяти доступно для запросов → спиллинг активируется чаще
- При уменьшении `spilling_percent` спиллинг включается при меньшем заполнении пула запросов

- При увеличении `spilling_percent` задачи дольше наращивают потребление RAM, прежде чем перейти к спиллингу

> **Важно**
>
> Важно учитывать, что сам спиллинг также требует память. Если установить `activities_limit_percent` слишком высоким, память может все равно закончиться несмотря на спиллинг, поскольку механизм спиллинга сам потребляет ресурсы памяти.

Требования к файловой системе

Файловые дескрипторы

> **Примечание**
>
> Для получения информации о настройке лимитов файловых дескрипторов при первоначальном развертывании см. раздел Лимиты файловых дескрипторов.

Примеры конфигурации

Высоконагруженная система

Для максимальной производительности в высоконагруженных системах рекомендуется увеличить размер спиллинга:

```
table_service_config:
  spilling_service_config:
    local_file_config:
      root: ""
      max_total_size: 107374182400   # 100 GiB
```

Ограниченные ресурсы

Для систем с ограниченными ресурсами рекомендуется использовать консервативные настройки:

```
table_service_config:
  spilling_service_config:
    local_file_config:
      root: ""
      max_total_size: 5368709120     # 5 GiB
```

Полный пример

```
table_service_config:
  spilling_service_config:
    local_file_config:
      enable: true
      root: "/var/spilling"
      max_total_size: 53687091200    # 50 GiB
```

См. также

- Концепция спиллинга
- Архитектура Spilling Service
- Устранение неполадок спиллинга
- Конфигурация контроллера памяти
- Мониторинг YDB
- Диагностика производительности

tli_config

Секция `tli_config` содержит параметры диагностики инвалидации блокировок транзакций (Transaction Lock Invalidation, TLI).

TLI — это механизм, при котором одна транзакция (нарушитель) ломает оптимистичные блокировки другой транзакции (жертвы), вынуждая жертву откатиться и повторить выполнение. Подробнее о диагностике TLI см. в разделе Инвалидация блокировок транзакций.

Параметры конфигурации

Параметр	Тип	По умолчанию	Описание
`ignored_table_regexes`	repeated string	`[]`	Список регулярных выражений путей таблиц, исключённых из TLI-диагностики

ignored_table_regexes

Позволяет исключить определённые таблицы из TLI-логирования и статистики. Если каждая таблица, используемая в SQL запросе на котором возникли TLI, соответствуют хотя бы одному из указанных регулярных выражений, TLI-лог для этого конфликта не формируется.

Изменения применяются к новым сессиям без перезапуска узлов.

Типичные сценарии использования:

- снижение объёма логов для системных или служебных таблиц, конфликты в которых ожидаемы;
- исключение таблиц очередей с высокой частотой конфликтов, не требующих диагностики.

Регулярные выражения применяются к полному пути таблицы, например `/Root/mydb/mytable`. Синтаксис соответствует ECMAScript regex.

Пример конфигурации

```
tli_config:
  ignored_table_regexes:
    - "/Root/.*/queue_.*"
    - "/Root/system/.*"
```

В этом примере из TLI-диагностики исключаются:

- таблицы с именем, начинающимся на `queue_`, в любой базе данных;
- все таблицы в директории `/Root/system/`.

Смотрите также

- Инвалидация блокировок транзакций
- log_config
- Логирование в YDB

Справка по наблюдаемости YDB

В этом разделе документации YDB рассматриваются различные темы, связанные с наблюдаемостью (observability), которые **не** зависят от выбранного метода развёртывания.

- Справка по метрикам
- Дашборды Grafana для YDB
- Трассировка в YDB
- Передача внешнего trace-id в YDB

Справка по наблюдаемости YDB

В этом разделе документации YDB рассматриваются различные темы, связанные с наблюдаемостью (observability), которые **не** зависят от выбранного метода развёртывания.

- Справка по метрикам
- Дашборды Grafana для YDB
- Трассировка в YDB
- Передача внешнего trace-id в YDB

Справка по метрикам

В этом документе приведен справочник по основным метрикам системы YDB, которые используются для мониторинга состояния кластера, диагностики производительности и анализа нагрузки.

Как посмотреть текущие значения метрик во встроенном веб-интерфейсе и настроить их сбор в Prometheus и Grafana описано в разделе Настройка мониторинга кластера YDB.

Метрики использования ресурсов

Имя метрики Тип, единицы измерения	Описание Метки
`resources.storage.used_bytes` `IGAUGE`, байты	Размер пользовательских и служебных данных, сохраненных в распределенном сетевом хранилище. `resources.storage.used_bytes` = `resources.storage.table.used_bytes` + `resources.storage.topic.used_bytes`.
`resources.storage.table.used_bytes` `IGAUGE`, байты	Размер пользовательских и служебных данных, сохраненных таблицами в распределенном сетевом хранилище. К служебным данным относятся данные первичного, вторичных индексов, векторных индексов, полнотекстовых индексов, локальных Блум-индексов и локального min_max-индекса.
`resources.storage.topic.used_bytes` `IGAUGE`, байты	Размер распределенного сетевого хранилища, используемого топиками. Равен сумме значений `topic.storage_bytes` всех топиков.
`resources.storage.limit_bytes` `IGAUGE`, байты	Ограничение на размер пользовательских и служебных данных, которые база данных может сохранить в распределенном сетевом хранилище.

Метрики GRPC API общие

Имя метрики Тип, единицы измерения	Описание Метки
`api.grpc.request.bytes` `RATE`, байты	Размер запросов, которые получены базой данных в определенный период времени. Метки: - *api_service* – название сервиса gRPC API, например `table` или `data_streams`. - *method* – название метода сервиса gRPC API, например `ExecuteDataQuery` (для сервиса `table`), или `PutRecord`, `GetRecords` (для сервиса `data_streams`).
`api.grpc.request.dropped_count` `RATE`, штуки	Количество запросов, обработка которых была прекращена на транспортном (gRPC) уровне из-за ошибки. Метки: - *api_service* – название сервиса gRPC API, например `table`. - *method* – название метода сервиса gRPC API, например `ExecuteDataQuery`.
`api.grpc.request.inflight_count` `IGAUGE`, штуки	Количество запросов, которые одновременно обрабатываются базой данных в определенный период времени. Метки: - *api_service* – название сервиса gRPC API, например `table`. - *method* – название метода сервиса gRPC API, например `ExecuteDataQuery`.
`api.grpc.request.inflight_bytes` `IGAUGE`, байты	Размер запросов, которые одновременно обрабатываются базой данных в определенный период времени. Метки: - *api_service* – название сервиса gRPC API, например `table`. - *method* – название метода сервиса gRPC API, например `ExecuteDataQuery`.
`api.grpc.response.bytes` `RATE`, байты	Размер ответов, которые отправлены базой данных в определенный период времени. Метки: - *api_service* – название сервиса gRPC API, например `table`. - *method* – название метода сервиса gRPC API, например `ExecuteDataQuery`.
`api.grpc.response.count` `RATE`, штуки	Количество ответов, которые отправлены базой в определенный период времени. Метки: - *api_service* – название сервиса gRPC API, например `table`. - *method* – название метода сервиса gRPC API, например `ExecuteDataQuery`. - *status* – статус выполнения запроса, подробнее статусы описаны в разделе Обработка ошибок.
`api.grpc.response.dropped_count` `RATE`, штуки	Количество ответов, отправка которых была прекращена на транспортном (gRPC) уровне из-за ошибки. Метки: - *api_service* – название сервиса gRPC API, например `table`. - *method* – название метода сервиса gRPC API, например `ExecuteDataQuery`.
`api.grpc.response.issues` `RATE`, штуки	Количество ошибок определенного типа, возникших при выполнении запросов в определенный период времени. Метки: - *issue_type* – тип ошибки, единственное значение – `optimistic_locks_invalidation`, подробнее инвалидация блокировок описана в разделе Транзакции и запросы к YDB.

Метрики GRPC API для топиков

Имя метрики Тип, единицы измерения	Описание Метки

`grpc.topic.stream_read.commits` `RATE`, штуки	Количество коммитов метода `Ydb::TopicService::StreamRead`. Метки: - *topic* – название топика. - *consumer* – имя читателя.
`grpc.topic.stream_read.bytes` `RATE`, штуки	Количество байт, прочитанных методом `Ydb::TopicService::StreamRead`. Метки: - *topic* – название топика. - *consumer* – имя читателя.
`grpc.topic.stream_read.messages` `RATE`, штуки	Количество сообщений, прочитанных методом `Ydb::TopicService::StreamRead`. Метки: - *topic* – название топика. - *consumer* – имя читателя.
`grpc.topic.stream_read.partition_session.errors` `RATE`, штуки	Количество ошибок при работе с партицией. Метки: - *topic* – название топика. - *consumer* – имя читателя.
`grpc.topic.stream_read.partition_session.started` `RATE`, штуки	Количество сессий, запущенных в единицу времени. Метки: - *topic* – название топика. - *consumer* – имя читателя.
`grpc.topic.stream_read.partition_session.stopped` `RATE`, штуки	Количество сессий, остановленных в единицу времени. Метки: - *topic* – название топика. - *consumer* – имя читателя.
`grpc.topic.stream_read.partition_session.starting_count` `RATE`, штуки	Количество запускаемых сессий (то есть клиенту пришла команда о запуске сессии, но клиент еще не запустил сессию). Метки: - *topic* – название топика. - *consumer* – имя читателя.
`grpc.topic.stream_read.partition_session.stopping_count` `RATE`, штуки	Количество останавливаемых сессий. Метки: - *topic* – название топика. - *consumer* – имя читателя.
`grpc.topic.stream_read.partition_session.count` `RATE`, штуки	Количество partition_session. Метки: - *topic* – название топика. - *consumer* – имя читателя.
`grpc.topic.stream_write.bytes` `RATE`, байты	Количество байт, записанных методом `Ydb::TopicService::StreamWrite`. Метки: - *topic* – название топика.
`grpc.topic.stream_write.uncommitted_bytes` `RATE`, байты	Количество байт, записанных методом `Ydb::TopicService::StreamWrite` в рамках еще не закомиченных транзакций. Метки: - *topic* – название топика.
`grpc.topic.stream_write.errors` `RATE`, штуки	Количество ошибок при вызове метода `Ydb::TopicService::StreamWrite`. Метки: - *topic* – название топика.
`grpc.topic.stream_write.messages` `RATE`, штуки	Количество сообщений, записанных методом `Ydb::TopicService::StreamWrite`. Метки: - *topic* – название топика.
`grpc.topic.stream_write.uncommitted_messages` `RATE`, штуки	Количество сообщений, записанных методом `Ydb::TopicService::StreamWrite` в рамках еще не закомиченных транзакций. Метки: - *topic* – название топика.
`grpc.topic.stream_write.partition_throttled_milliseconds` `HIST_RATE`, штуки	Гистограммный счетчик. Интервалы заданы в миллисекундах. Показывает количество сообщений, ожидавших на квоте. Метки: - *topic* – название топика.
`grpc.topic.stream_write.sessions_active_count` `GAUGE`, штуки	Количество открытых сессий записи. Метки: - *topic* – название топика.
`grpc.topic.stream_write.sessions_created` `RATE`, штуки	Количество созданных сессий записи. Метки: - *topic* – название топика.

Метрики HTTP API

Имя метрики Тип, единицы измерения	Описание Метки
`api.http.data_streams.request.count` `RATE`, штуки	Количество запросов по протоколу HTTP. Метки: - *method* – название метода сервиса HTTP API, например `PutRecord`, `GetRecords`. - *topic* – название топика.
`api.http.data_streams.request.bytes` `RATE`, байты	Суммарный размер запросов по протоколу HTTP. Метки: - *method* – название метода сервиса HTTP API, в данном случае только `PutRecord`. - *topic* – название топика.
`api.http.data_streams.response.count` `RATE`, штуки	Количество ответов по протоколу HTTP. Метки: - *method* – название метода сервиса HTTP API, например `PutRecord`, `GetRecords`. - *topic* – название топика. - *code* – код ответа HTTP.
`api.http.data_streams.response.bytes` `RATE`, байты	Суммарный размер ответов по протоколу HTTP. Метки: - *method* – название метода сервиса HTTP API, в данном случае только `GetRecords`. - *topic* – название топика.
`api.http.data_streams.response.duration_milliseconds` `HIST_RATE`, штуки	Гистограммный счетчик. Интервалы заданы в миллисекундах. Показывает количество ответов, время выполнения которых попадает в определенный интервал. Метки: - *method* – название метода сервиса HTTP API. - *topic* – название топика.
`api.http.data_streams.get_records.messages` `RATE`, штуки	Количество сообщений, прочитанных методом `GetRecords`. Метки: - *topic* – название топика.
`api.http.data_streams.put_record.messages` `RATE`, штуки	Количество сообщений, отправленных методом `PutRecord`. Метки: - *topic* – название топика.
`api.http.data_streams.put_records.failed_messages` `RATE`, штуки	Количество сообщений, отправленных методом `PutRecords`, которые не были записаны. Метки: - *topic* – название топика.
`api.http.data_streams.put_records.successful_messages` `RATE`, штуки	Количество сообщений, отправленных методом `PutRecords`, которые были успешно записаны. Метки: - *topic* – название топика.
`api.http.data_streams.put_records.total_messages` `RATE`, штуки	Количество сообщений, отправленных методом `PutRecords`. Метки: - *topic* – название топика.

Метрики Kafka API

Имя метрики Тип, единицы измерения	Описание Метки
`api.kafka.request.count` `RATE`, штуки	Количество запросов по протоколу Kafka в единицу времени. Метки: - *method* – название метода сервиса Kafka API, например `PRODUCE`, `SASL_HANDSHAKE`.
`api.kafka.request.bytes` `RATE`, байты	Суммарный размер запросов по протоколу Kafka в единицу времени. Метки: - *method* – название метода сервиса Kafka API, например `PRODUCE`, `SASL_HANDSHAKE`.
`api.kafka.response.count` `RATE`, штуки	Количество ответов по протоколу Kafka в единицу времени. Метки: - *method* – название метода сервиса Kafka API, например `PRODUCE`, `SASL_HANDSHAKE`. - *error_code* – код ответа Kafka.
`api.kafka.response.bytes` `RATE`, байты	Суммарный размер ответов по протоколу Kafka в единицу времени. Метки: - *method* – название метода сервиса Kafka API, например `PRODUCE`, `SASL_HANDSHAKE`.
`api.kafka.response.duration_milliseconds` `HIST_RATE`, штуки	Гистограммный счетчик. Определяет набор интервалов в миллисекундах и для каждого из них показывает количество запросов с попадающим в этот интервал временем выполнения. Метки: - *method* – название метода сервиса Kafka API.

`api.kafka.produce.failed_messages` `RATE`, штуки	Количество сообщений в единицу времени, отправленных методом `PRODUCE`, которые не были записаны. Метки: - *topic* – название топика.
`api.kafka.produce.successful_messages` `RATE`, штуки	Количество сообщений в единицу времени, отправленных методом `PRODUCE`, которые были успешно записаны. Метки: - *topic* – название топика.
`api.kafka.produce.total_messages` `RATE`, штуки	Количество сообщений в единицу времени, отправленных методом `PRODUCE`. Метки: - *topic* – название топика.

Метрики сессий

Имя метрики Тип, единицы измерения	Описание Метки
`table.session.active_count` `IGAUGE`, штуки	Количество сессий, открытых клиентами в данный момент времени.
`table.session.closed_by_idle_count` `RATE`, штуки	Количество сессий, которые закрыты по инициативе сервера баз данных в определенный период времени из-за превышения времени, выделенного на существование неактивной сессии.

Метрики обработки транзакций

Длительность выполнения транзакции можно анализировать с помощью гистограммного счетчика. Интервалы заданы в миллисекундах. График показывает количество транзакций, длительность которых попадает в определенный интервал времени.

Имя метрики Тип, единицы измерения	Описание Метки
`table.transaction.total_duration_milliseconds` `HIST_RATE`, штуки	Количество транзакций определенной длительности выполнения на сервере и клиенте. Длительность выполнения – это время выполнения транзакции от момента явного или неявного открытия транзакции до момента фиксации изменений или отката. Включает время обработки транзакции на сервере и время на клиенте между отправкой разных запросов в одной транзакции. Метки: - *tx_kind* – тип транзакции, возможные значения `read_only`, `read_write`, `write_only`, `pure`.
`table.transaction.server_duration_milliseconds` `HIST_RATE`, штуки	Количество транзакций определенной длительности выполнения на сервере. Длительность выполнения – это время выполнения запросов в транзакции на сервере. Не включает время ожидания на клиенте между отправкой отдельных запросов в одной транзакции. Метки: - *tx_kind* – тип транзакции, возможные значения `read_only`, `read_write`, `write_only`, `pure`.
`table.transaction.client_duration_milliseconds` `HIST_RATE`, штуки	Количество транзакций определенной длительности выполнения на клиенте. Длительность выполнения – это время ожидания на клиенте между отправкой отдельных запросов в одной транзакции. Не включает время выполнения запросов на сервере. Метки: - *tx_kind* – тип транзакции, возможные значения `read_only`, `read_write`, `write_only`, `pure`.

Метрики обработки запросов

Имя метрики Тип, единицы измерения	Описание Метки
`table.query.request.bytes` `RATE`, байты	Размер текста YQL-запросов и значений параметров к запросам, которые поступили в базу данных в определенный период времени.
`table.query.request.parameters_bytes` `RATE`, байты	Размер параметров к запросам, которые поступили в базу данных в определенный период времени.
`table.query.response.bytes` `RATE`, байты	Размер ответов, которые отправлены базой данных в определенный период времени.
`table.query.compilation.latency_milliseconds` `HIST_RATE`, штуки	Гистограммный счетчик. Интервалы заданы в миллисекундах. Показывает количество успешно выполненных запросов на компиляцию, длительность которых попадает в определенный интервал времени.
`table.query.compilation.active_count` `IGAUGE`, штуки	Количество компиляций, которые находятся в процессе выполнения в данный момент времени.
`table.query.compilation.count` `RATE`, штуки	Количество компиляций, которые успешно завершились в определенный период времени.
`table.query.compilation.errors` `RATE`, штуки	Количество компиляций, которые завершились с ошибкой в определенный период времени.

`table.query.compilation.cache_hits` `RATE`, штуки	Количество запросов в определенный период времени, для выполнения которых не потребовалось компилировать запрос, так как в кэше подготовленных запросов был созданный ранее план.
`table.query.compilation.cache_misses` `RATE`, штуки	Количество запросов в определенный период времени, для выполнения которых потребовалось компилировать запрос.
`table.query.execution.latency_milliseconds` `HIST_RATE`, штуки	Гистограммный счетчик. Интервалы заданы в миллисекундах. Показывает количество запросов, время выполнения которых попадает в определенный интервал.

Метрики партиций строковых таблиц

Имя метрики Тип, единицы измерения	Описание Метки
`table.datashard.row_count` `GAUGE`, штуки	Количество строк во всех строковых таблицах базы данных.
`table.datashard.size_bytes` `GAUGE`, байты	Размер данных во всех строковых таблицах базы.
`table.datashard.used_core_percents` `HIST_GAUGE`, %	Гистограммный счетчик. Интервалы заданы в процентах. Показывает количество партиций строковых таблиц, которые используют вычислительные ресурсы в доле, попадающей в определенный интервал.
`table.datashard.read.rows` `RATE`, штуки	Количество строк, которые прочитаны всеми партициями всех строковых таблиц в базе данных в определенный период времени.
`table.datashard.read.bytes` `RATE`, байты	Размер данных, которые прочитаны всеми партициями всех строковых таблиц в базе в определенный период времени.
`table.datashard.write.rows` `RATE`, штуки	Количество строк, которые записаны всеми партициями всех строковых таблиц в базе данных в определенный период времени.
`table.datashard.write.bytes` `RATE`, байты	Размер данных, которые записаны всеми партициями всех строковых таблиц в базе в определенный период времени.
`table.datashard.scan.rows` `RATE`, штуки	Количество строк, которые прочитаны через вызовы gRPC API `StreamExecuteScanQuery` или `StreamReadTable` всеми партициями всех строковых таблиц в базе данных в определенный период времени.
`table.datashard.scan.bytes` `RATE`, байты	Размер данных, которые прочитаны через вызов gRPC API `StreamExecuteScanQuery` или `StreamReadTable` всеми партициями всех строковых таблиц в базе в определенный период времени.
`table.datashard.bulk_upsert.rows` `RATE`, штуки	Количество строк, которые добавлены через вызов gRPC API `BulkUpsert` во все партиции всех строковых таблиц в базе данных в определенный период времени.
`table.datashard.bulk_upsert.bytes` `RATE`, байты	Размер данных, которые добавлены через вызов gRPC API `BulkUpsert` во все партиции всех строковых таблиц в базе в определенный период времени.
`table.datashard.erase.rows` `RATE`, штуки	Количество строк, которые удалены в базе данных из всех строковых таблиц в определенный период времени.
`table.datashard.erase.bytes` `RATE`, байты	Размер данных, которые удалены в базе из всех строковых таблиц в определенный период времени.
`table.datashard.cache_hit.bytes` `RATE`, байты	Общий объем данных строковых таблиц, успешно полученных из памяти (кэша). Больший объем данных, полученных из кэша, свидетельствует об эффективном использовании кэша без доступа к распределенному хранилищу.
`table.datashard.cache_miss.bytes` `RATE`, байты	Общий объем данных строковых таблиц, которые были запрошены, но не найдены в памяти (кэше), и были прочитаны из распределенного хранилища. Указывает на потенциальные области для оптимизации кэша.

Метрики партиций колоночных таблиц

Имя метрики Тип, единицы измерения	Описание Метки
`table.columnshard.write.rows` `RATE`, штуки	Количество строк, которые записаны всеми партициями всех колоночных таблиц в базе данных в определенный период времени.
`table.columnshard.write.bytes` `RATE`, байты	Размер данных, которые записаны всеми партициями всех колоночных таблиц в базе данных в определенный период времени.
`table.columnshard.scan.rows` `RATE`, штуки	Количество строк, которые прочитаны через вызовы gRPC API `StreamExecuteScanQuery` или `StreamReadTable` всеми партициями всех колоночных таблиц в базе данных в определенный период времени.
`table.columnshard.scan.bytes` `RATE`, байты	Размер данных, которые прочитаны через вызов gRPC API `StreamExecuteScanQuery` или `StreamReadTable` всеми партициями всех колоночных таблиц в базе данных в определенный период времени.
`table.columnshard.bulk_upsert.rows` `RATE`, штуки	Количество строк, которые добавлены через вызов gRPC API `BulkUpsert` во все партиции всех колоночных таблиц в базе данных в определенный период времени.

`table.columnshard.bulk_upsert.bytes` `RATE`, байты	Размер данных, которые добавлены через вызов gRPC API `BulkUpsert` во все партиции всех колоночных таблиц в базе данных в определенный период времени.

Метрики использования ресурсов (только для режима Dedicated)

Имя метрики Тип единицы измерения	Описание Метки
`resources.cpu.used_core_percents` `RATE`, %	Использование CPU. Значение `100` означает, что одно из ядер использовано на 100%. Значение может быть больше `100` для конфигураций с более чем 1 ядром. Метки: - *pool* – вычислительный пул, возможные значения `user`, `system`, `batch`, `io`, `ic`.
`resources.cpu.limit_core_percents` `IGAUGE`, %	Доступный базе данных CPU в процентах. Например, для БД из трех нод по 4 ядра в `pool=user` в каждой ноде, значение этого сенсора будет равно `1200`. Метки: - *pool* – вычислительный пул, возможные значения `user`, `system`, `batch`, `io`, `ic`.
`resources.memory.used_bytes` `IGAUGE`, байты	Использованная нодами базы данных оперативная память.
`resources.memory.limit_bytes` `IGAUGE`, байты	Доступная нодам базы данных оперативная память.

Метрики обработки запросов (только для режима Dedicated)

Имя метрики Тип единицы измерения	Описание Метки
`table.query.compilation.cache_evictions` `RATE`, штуки	Количество запросов, вытесненных из кэша подготовленных запросов в определенный период времени.
`table.query.compilation.cache_size_bytes` `IGAUGE`, байты	Размер кэша подготовленных запросов.
`table.query.compilation.cached_query_count` `IGAUGE`, штуки	Количество запросов в кэше подготовленных запросов.

Метрики топиков

Имя метрики Тип, единицы измерения	Описание Метки
`topic.producers_count` `GAUGE`, штуки	Количество уникальных источников топика. Метки: - *topic* – название топика.
`topic.storage_bytes` `GAUGE`, байты	Размер топика в байтах. Метки: - *topic* – название топика.
`topic.read.bytes` `RATE`, байты	Количество байт, прочитанных из топика. Метки: - *topic* – название топика. - *consumer* – имя читателя.
`topic.read.messages` `RATE`, штуки	Количество сообщений, прочитанных из топика. Метки: - *topic* – название топика. - *consumer* – имя читателя.
`topic.read.lag_messages` `GAUGE`, штуки	Суммарное по топику количество невычитанных данным читателем сообщений. Метрика служит индикатором отставания читателя. Рост значения означает, что читатель не успевает за потоком сообщений — например, из-за остановки читателя, ребалансировки партиций или всплеска нагрузки на запись. Метки: - *topic* – название топика. - *consumer* – имя читателя.
`topic.read.lag_milliseconds` `HIST_RATE`, штуки	Гистограммный счетчик. Интервалы заданы в миллисекундах. Показывает количество сообщений, у которых разница между временем чтения и временем создания сообщения попадает в заданный интервал. Метки: - *topic* – название топика. - *consumer* – имя читателя.
`topic.write.bytes` `RATE`, байты	Размер записанных данных. Метки: - *topic* – название топика.
`topic.write.uncommited_bytes` `RATE`, байты	Размер данных, записанных в рамках еще не завершенных транзакций. Метки: - *topic* — название топика.
`topic.write.uncompressed_bytes` `RATE`, байты	Размер разжатых записанных данных. Метки: - *topic* – название топика.

`topic.write.messages` `RATE`, штуки	Количество записанных сообщений. Метки: - *topic* – название топика.
`topic.write.uncommitted_messages` `RATE`, штуки	Количество сообщений, записанных в рамках еще не завершенных транзакций. Метки: - *topic* — название топика.
`topic.write.message_size_bytes` `HIST_RATE`, штуки	Гистограммный счетчик. Интервалы заданы в байтах. Показывает количество сообщений, размер которых соответствует границам интервала. Метки: - *topic* – название топика.
`topic.write.lag_milliseconds` `HIST_RATE`, штуки	Гистограммный счетчик. Интервалы заданы в миллисекундах. Показывает количество сообщений, у которых разница между временем записи и временем создания сообщения попадает в заданный интервал. Метки: - *topic* – название топика.

Агрегированные метрики партиций топика

В следующей таблице приведены агрегированные метрики партиций для топика. Максимальные и минимальные значения считаются по всем партициям заданного топика.

Имя метрики Тип, единицы измерения	Описание Метки
`topic.partition.init_duration_milliseconds_max` `GAUGE`, миллисекунды	Максимальная задержка инициализации партиции. Метки: - *topic* – название топика.
`topic.partition.producers_count_max` `GAUGE`, штуки	Максимальное количество источников в партиции. Метки: - *topic* – название топика.
`topic.partition.storage_bytes_max` `GAUGE`, байты	Максимальный размер партиции в байтах. Метки: - *topic* – название топика.
`topic.partition.uptime_milliseconds_min` `GAUGE`, штуки	Минимальное время работы партиции после рестарта. В норме во время rolling restart-а `topic.partition.uptime_milliseconds_min` близко к 0, после окончания rolling restart-а значение `topic.partition.uptime_milliseconds_min` должно увеличиваться до бесконечности. Метки: - *topic* – название топика.
`topic.partition.total_count` `GAUGE`, штуки	Общее количество партиций в топике. Метки: - *topic* – название топика.
`topic.partition.alive_count` `GAUGE`, штуки	Количество партиций, отправляющих свои метрики. Метки: - *topic* – название топика.
`topic.partition.committed_end_to_end_lag_milliseconds_max` `GAUGE`, миллисекунды	Максимальная (по всем партициям) разница между текущим временем и временем создания последнего закомиченного сообщения. Метки: - *topic* – название топика. - *consumer* – имя читателя.
`topic.partition.committed_lag_messages_max` `GAUGE`, штуки	Максимальная (по всем партициям) разница между последним оффсетом партиции и закомиченным оффсетом партиции. Метки: - *topic* – название топика. - *consumer* – имя читателя.
`topic.partition.committed_read_lag_milliseconds_max` `GAUGE`, миллисекунды	Максимальная (по всем партициям) разница между текущим временем и временем записи самого раннего незакомиченного сообщения. Если метрика больше нуля, значит в топике есть как минимум одно сообщение; оно уже записано, возможно прочитано, но ещё не закоммичено. Метки: - *topic* – название топика. - *consumer* – имя читателя.
`topic.partition.end_to_end_lag_milliseconds_max` `GAUGE`, миллисекунды	Разница между текущим временем и минимальным временем создания среди всех вычитанных за последнюю минуту сообщений во всех партициях. Метки: - *topic* – название топика. - *consumer* – имя читателя.
`topic.partition.lag_messages_max` `GAUGE`, штуки	Максимальная разница (по всем партициям) последнего оффсета в партиции и последнего вычитанного оффсета. Метки: - *topic* – название топика. - *consumer* – имя читателя.

`topic.partition.read.idle_milliseconds_max` `GAUGE`, миллисекунды	Максимальное время простоя (сколько времени не читали из партиции) по всем партициям. Метки: - *topic* – название топика. - *consumer* – имя читателя.
`topic.partition.read.lag_milliseconds_max` `GAUGE`, миллисекунды	Разница между текущим временем и минимальным временем записи среди всех вычитанных за последнюю минуту сообщений во всех партициях. Метки: - *topic* – название топика. - *consumer* – имя читателя.
`topic.partition.write.lag_milliseconds_max` `GAUGE`, миллисекунды	Максимальная разница между временем записи и временем создания среди всех записанных за последнюю минуту сообщений. Метки: - *topic* – название топика.
`topic.partition.write.speed_limit_bytes_per_second` `GAUGE`, байты в секунду	Квота на запись в байтах в секунду на одну партицию. Метки: - *topic* – название топика.
`topic.partition.write.throttled_nanoseconds_max` `GAUGE`, наносекунды	Максимальное время троттлинга записи (ожидания на квоте) по всем партициям. В пределе если `topic.partition.write.throttled_nanoseconds_max` = 10^9, то это означает, что всю секунду ожидали на квоте. Метки: - *topic* – название топика.
`topic.partition.write.bytes_per_day_max` `GAUGE`, байты	Максимальное количество байт, записанное за последние сутки, по всем партициям. Метки: - *topic* – название топика.
`topic.partition.write.bytes_per_hour_max` `GAUGE`, байты	Максимальное количество байт, записанное за последний час, по всем партициям. Метки: - *topic* – название топика.
`topic.partition.write.bytes_per_minute_max` `GAUGE`, байты	Максимальное количество байт, записанное за последнюю минуту, по всем партициям. Метки: - *topic* – название топика.
`topic.partition.write.idle_milliseconds_max` `GAUGE`, миллисекунды	Максимальное время простоя партиции на запись. Метки: - *topic* – название топика.

Метрики пулов ресурсов

Имя метрики Тип, единицы измерения	Описание Метки
`kqp.workload_manager.CpuQuotaManager.AverageLoadPercentage` `RATE`, штуки	Средняя загрузка базы данных, по этой метрике работает `DATABASE_LOAD_CPU_THRESHOLD`.
`kqp.workload_manager.InFlightLimit` `GAUGE`, штуки	Лимит на число одновременно работающих запросов.
`kqp.workload_manager.GlobalInFly` `GAUGE`, штуки	Текущее число одновременно работающих запросов. Отображаются только для пулов с включенным `CONCURRENT_QUERY_LIMIT` или `DATABASE_LOAD_CPU_THRESHOLD`.
`kqp.workload_manager.QueueSizeLimit` `GAUGE`, штуки	Размер очереди запросов, ожидающих выполнения.
`kqp.workload_manager.GlobalDelayedRequests` `GAUGE`, штуки	Количество запросов, ожидающих в очереди на выполнение. Отображаются только для пулов с включенным `CONCURRENT_QUERY_LIMIT` или `DATABASE_LOAD_CPU_THRESHOLD`.

См. также

- Дашборды Grafana для мониторинга метрик YDB

Дашборды Grafana для YDB

На этой странице описаны дашборды Grafana для YDB.

Как установить и настроить дашборды описано в разделе по настройке мониторинга кластера YDB.

DB status

Общий дашборд базы данных.

Скачать шаблон дашборда **DB status**: dbstatus.json.

DB overview

Общий дашборд базы данных по категориям:

- Health
- API
- API details
- CPU
- CPU pools
- Memory
- Storage
- DataShard
- DataShard details
- Latency

Скачать шаблон дашборда **DB overview**: dboverview.json.

YDB Essential Metrics

Дашборд для мониторинга ключевых показателей базы данных.

Секция Health

В этой секции размещены графики, отображающие состояние компонентов кластера и базы данных.

Имя	Описание
Nodes count	Количество запущенных узлов YDB, в шт.
Nodes uptime	Время непрерывной работы каждого узла с момента запуска; помогает обнаруживать перезапуски и нестабильные узлы, в секундах.
VDisks count	Количество доступных VDisk в кластере, в шт.

Секция Saturation

В этой секции размещены графики, отражающие утилизацию ресурсов базы данных.

Имя	Описание
CPU by thread pool (dynnodes)	Потребление CPU динамическими узлами в разрезе пулов исполнения, в ядрах CPU.
CPU utilization (dynnodes)	Утилизация CPU динамическими узлами в разрезе пулов исполнения, в %.
Elapsed Time vs CPU Time	Соотношение реального времени выполнения операций (`ElapsedMicrosec`) и процессорного времени (`CpuMicrosec`) по узлам. Устойчивое превышение отметки 100% означает, что сессии проводят время в ожидании, а не в активной работе: как правило, это ожидание I/O или CPU overcommit на стороне гипервизора.
RSS size by node	Объём оперативной памяти (Resident set size), потребляемой каждым динамическим узлом, с отображением лимитов памяти cgroup, в байтах.
Storage usage	Логический размер базы данных и установленный на него лимит, в байтах.
Overloaded shard count	Количество перегруженных DataShard по диапазонам загрузки CPU — от 50% до 100%, в шт.

Секция Traffic

В этой секции расположены графики, характеризующие нагрузку на базу данных.

Имя	Описание
Queries per second by latency buckets	Количество запросов в секунду с разбивкой по диапазонам задержки (от 1 мс до +∞). Каждый диапазон выделен отдельным цветом — от зелёного для быстрых запросов до фиолетового для медленных. Позволяет оценить распределение задержек и общий rps, в шт/с
Transactions per second by latency buckets	Количество транзакций в секунду с разбивкой по диапазонам задержки (от 1 мс до +∞). Каждый диапазон выделен отдельным цветом — от зелёного для быстрых транзакций до фиолетового для медленных. Позволяет оценить распределение задержек и общий tps, в шт/с
Rows read, uploaded, updated, deleted	Количество операций со строками таблиц в секунду: чтение, создание, обновление и удаление, в операциях/с
Session count by dynnode	Количество активных сессий на каждом динамическом узле, в шт

Секция Latency

В этой секции расположены графики, отображающие время выполнения запросов и транзакций.

Имя	Описание
Query latency percentiles (ms)	Время выполнения запросов к базе данных в перцентилях p50, p90, p95, p99, в миллисекундах
Transaction latency percentiles (ms)	Время выполнения транзакций в базе данных в перцентилях p50, p90, p95, p99, в миллисекундах

Секция Errors

В этой секции расположены графики, отображающие количество возникающих ошибок.

Имя	Описание
YQL Issues per second	Количество ошибок выполнения YQL-запросов по типам ошибок, в шт/с.
gRPC response errors per second	Количество gRPC-ответов с ошибками с разбивкой по статусам, в шт/с.

Скачать шаблон дашборда **YDB Essential Metrics**: ydb-essentials.json.

Actors

Потребление CPU в актор-системе.

Имя	Описание
CPU by execution pool (us)	Потребление CPU в различных пулах исполнения на всех нодах, микросекунды в секунду (один миллион соответствует потреблению одного ядра)
Actor count	Количество акторов (по типу актора)
CPU	Потребление CPU в различных пулах исполнения (по типу актора)
Events	Метрики обработки событий в актор-системе

Скачать шаблон дашборда **Actors**: actors.json.

CPU

Потребление CPU в пулах исполнения.

Имя	Описание
CPU by execution pool	Потребление CPU в различных пулах исполнения на всех нодах, микросекунды в секунду (один миллион соответствует потреблению одного ядра)
Actor count	Количество акторов (по типу актора)
CPU	Потребление CPU в различных пулах исполнения
Events	Метрики обработки событий в различных пулах исполнения

Скачать шаблон дашборда **CPU**: cpu.json.

gRPC

Метрики слоя gRPC.

Имя	Описание
Requests	Количество запросов, получаемых базой данных в секунду (по типу метода gRPC)
Request bytes	Размер запросов, получаемых базой данных, байты в секунду (по типу метода gRPC)
Response bytes	Размер ответов, отправляемых базой данных, байты в секунду (по типу метода gRPC)
Dropped requests	Количество запросов в секунду, обработка которых была прекращена на транспортном уровне из-за ошибки (по типу метода gRPC)
Dropped responses	Количество ответов в секунду, отправка которых была прекращена на транспортном уровне из-за ошибки (по типу метода gRPC)
Requests in flight	Количество запросов, которые одновременно обрабатываются базой данных (по типу метода gRPC)
Request bytes in flight	Размер запросов, которые одновременно обрабатываются базой данных (по типу метода gRPC)

Скачать шаблон дашборда **gRPC**: grpc.json.

Query engine

Сведения о движке исполнения запросов.

Имя	Описание
Requests	Количество входящих запросов в секунду (по типу запроса)
Request bytes	Размер входящих запросов, байты в секунду (`query, parameters, total`)

Responses	Количество ответов в секунду (по типу ответа)
Response bytes	Размеры ответов, байты в секунду (`total, query result`)
Sessions	Сведения об установленных сессиях
Latencies	Гистограммы времен исполнения запросов для различных типов запросов

Скачать шаблон дашборда **Query engine**: queryengine.json.

TxProxy

Информация о транзакциях с уровня `DataShard transaction proxy` .

Имя	Описание
Transactions	Метрики транзакций даташардов
Latencies	Гистограммы времен исполнения различных этапов транзакций даташардов

Скачать шаблон дашборда **TxProxy**: txproxy.json.

DataShard

Метрики таблетки `DataShard` .

Имя	Описание
Operations	Статистика операций с даташардом для разных типов операций
Transactions	Информация о транзакциях таблетки даташарда (по типам транзакций)
Latencies	Гистограммы времен выполнения различных этапов пользовательских транзакций
Tablet latencies	Гистограммы времен выполнения транзакций таблетки
Compactions	Сведения о производимых операциях LSM compaction
ReadSets	Сведения о пересылаемых ReadSets при исполнении пользовательской транзакции
Other	Прочие метрики

Скачать шаблон дашборда **DataShard**: datashard.json.

Database Hive

Метрики таблетки Hive выбранной базы данных.

На дашборде размещены следующие фильтры:

- `database` — используется для выбора базы данных, метрики которой необходимо отобразить;
- `ds` — используется для выбора Prometheus-источника, данные из которого необходимо отобразить на дашборде;
- `Tx type` — определяет тип транзакции, для которого будут выведены графики на панели " `{Tx type}` `average time` ".

Имя	Описание
CPU usage by HIVE_ACTOR, HIVE_BALANCER_ACTOR	Процессорное время, потребляемое `HIVE_ACTOR` и `HIVE_BALANCER_ACTOR` — двумя самыми важными акторами Hive.
Self-ping time	Время ответа таблеткой Hive на собственные запросы. Высокие значения указывают на сильную загрузку (и медленную отзывчивость) Hive.
Local transaction times	Время работы CPU, потребляемое для выполнения различных типов локальных транзакций в Hive. Отображает структуру нагрузки на Hive.
Tablet count	Общее число таблеток в базе данных.
Event queue size	Размер очереди входящих событий. Постоянно высокие значения указывают на то, что Hive не успевает обрабатывать события с требуемой скоростью.
{Tx type} average time	Среднее время выполнения одной локальной транзакции типа, выбранного в фильтре `Tx type` .
Versions	Версии YDB, запущенные на узлах кластера.
Hive node	Узел, на котором запущен Hive.

Скачать шаблон дашборда **Database Hive**: database-hive-detailed.json.

Topic

На дашборде отображаются графики по метрикам одного топика. Название топика задаётся в фильтре `topic` в верхней части дашборда. Ниже перечислены панели и расшифровка метрик.

Имя	Описание
Total incoming records (bytes) per second	Количество байт в секунду, записанных в топик методом `Ydb::TopicService::StreamWrite`
Total incoming records (count) per second	Количество сообщений в секунду, записанных методом `Ydb::TopicService::StreamWrite`

Write latency	Длительность записи: время от создания сообщения до его записи в топик. Доля сообщений (в процентах), для которых длительность записи уложилась в интервалы <100 мс, <200 мс и т.д.
Partition throttling	Длительность троттлинга записи - ожидания доступной квоты на запись. Процент сообщений, длительность троттлинга записи которых уложилась в интервалы <1 мс, <5 мс и т.д.
Partition quota usage	Утилизация квот партиций топика на запись, %
Write sessions active	Количество открытых сессий записи в топик
Write sessions created	Количество создаваемых в секунду сессий записи в топик

Скачать шаблон дашборда **Topic**: topic.json.

Topic — Consumer

На дашборде отображаются графики по метрикам одного топика и связанного с ним читателя. Топик выбирается в фильтре `topic`, читатель — в фильтре `consumer`. Ниже перечислены панели и расшифровка метрик.

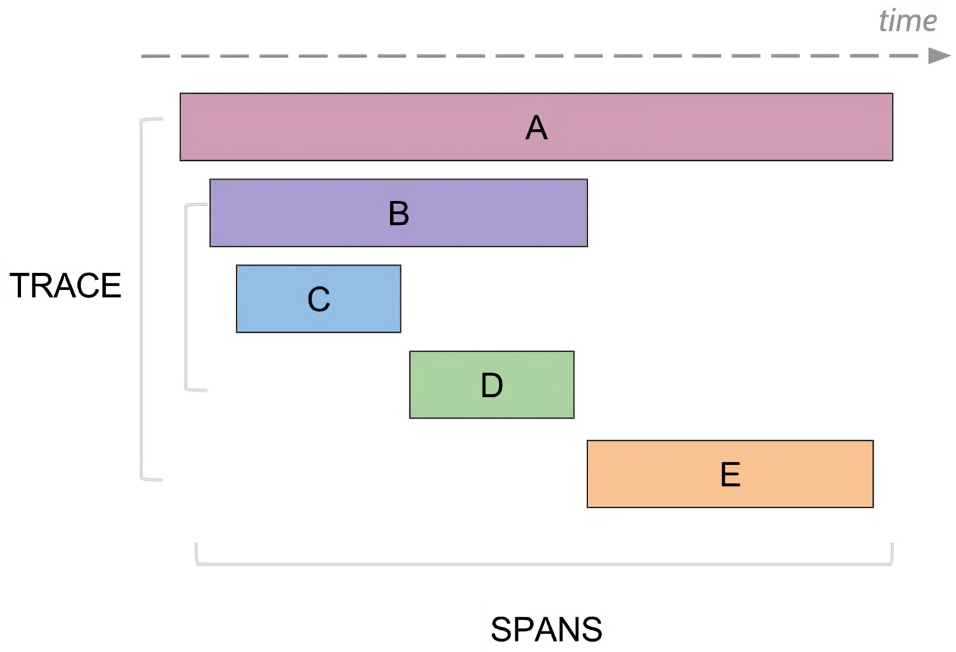
Имя	Описание
Total incoming records (bytes) per second	Количество байт в секунду, записанных в топик методом `Ydb::TopicService::StreamWrite`
Total outgoing records (bytes) per second	Количество байт в секунду, прочитанных из топика читателем методом `Ydb::TopicService::StreamRead`
Total incoming records (count) per second	Количество сообщений в секунду, записанных в топик методом `Ydb::TopicService::StreamWrite`
Total outgoing records (count) per second	Количество сообщений в секунду, прочитанных из топика читателем методом `Ydb::TopicService::StreamRead`
End-to-end latency	End-to-end длительность: время от момента создания сообщения до момента его чтения. Процент сообщений, end-to-end длительность для которых уложилась в интервалы <100 мс, <200 мс и т.д.
Read latency max	Максимальная (по всем партициям) разница между текущим временем и временем записи последнего сообщения в топик, мс
Unread messages max	Максимальная (по всем партициям) разница между последним смещением в партиции и последним прочитанным смещением, в сообщениях
Read idle time max	Максимальное время простоя (сколько времени консьюмер не читал из партиции) по всем партициям топика, мс
Uncommitted messages max	Максимальная (по всем партициям) разница между последним смещением в партиции и последним закомиченным смещением, в сообщениях
Committed read lag max	Максимальная (по всем партициям) разница между текущим временем и временем записи последнего закомиченного сообщения в топик, мс
Partition sessions started	Количество сессий чтения топика читателем, запущенных в секунду

Скачать шаблон дашборда **Topic — Consumer**: topic-consumer.json.

Трассировка в YDB

> **Примечание**
>
> Подробно трассировка описана на сайте opentelemetry в разделе observability primer.

Трассировка – инструмент, позволяющий детально посмотреть путь следования запроса по распределенной системе. Путь следования одного запроса (трасса) описывается набором спанов. Спан является отрезком времени, обычно связанным с временем выполнения определенной операции (например запись информации на диск, выполнение транзакции). Спаны образуют дерево, часто поддерево спана является его детализацией, однако это не всегда так.



Для аггрегации разрозненных спанов в трассы они отправляются в *коллектор*. Это сервис, который аггрегирует и сохраняет получаемые спаны для последующего анализа трасс. YDB не включает в себя этот сервис, администратор должен поднять его самостоятельно. Обычно в качестве коллектора используется jaeger.

Минимальная конфигурация

Для включения трассировки в YDB добавьте в конфигурацию следующую секцию:

```
tracing_config:
  backend:
    opentelemetry:
      collector_url: grpc://example.com:4317
      service_name: ydb
  external_throttling:
    - max_traces_per_minute: 10
```

В ней поле `collector_url` задает url OTLP-совместимого коллектора спанов. Более подробно с описанием секции backend можно ознакомиться в соответствующей секции.

При данной настройке никакие запросы не сэмплируются, не более 10 запросов в минуту с внешним trace-id трассируются каждым узлом кластера.

Описание секций

Backend

Пример секции

```
tracing_config:
  # ...
  backend:
    opentelemetry:
      collector_url: grpc://example.com:4317
      service_name: ydb
```

Описание

В этой секции описывается коллектор спанов. На данный момент единственная опция – `opentelemetry`. Отправка спанов с узла кластера в коллектор происходит по push-модели, от коллектора требуется совместимость с OTLP.

В секции `opentelemetry`:

- `collector_url` — URL коллектора спанов. Схема может быть либо `grpc://` для небезопасного подключения, либо `grpcs://` для TLS-подключения.
- `service_name` — имя сервиса, которым будут подписываться все спаны.

Оба параметра являются обязательными.

Uploader

Пример секции

```
tracing_config:
  # ...
  uploader:
    max_exported_spans_per_second: 30
    max_spans_in_batch: 100
    max_bytes_in_batch: 10485760 # 10 MiB
    max_export_requests_inflight: 3
    max_batch_accumulation_milliseconds: 5000
    span_export_timeout_seconds: 120
```

Описание

Uploader – компонент узла кластера, который отвечает за отправку спанов в коллектор. Во избежание перегрузки коллектора спанов, uploader не будет отправлять больше, чем `max_exported_spans_per_second` спанов в секунду в среднем.

В целях оптимизации uploader отправляет спаны батчами. Каждый батч содержит не более `max_spans_in_batch` спанов с суммарным сериализованным размеров не более `max_bytes_in_batch` байт. Каждый батч накапливается на протяжении не более, чем `max_batch_accumulation_milliseconds` миллисекунд. Батчи могут отправляться параллельно, максимальное количество одновременно отправляемых батчей контролируется параметром `max_export_requests_inflight`. Если с момента получения спана uploader'ом прошло больше `span_export_timeout_seconds` секунд, uploader может удалить его в пользу отправки более свежих спанов.

Значения по умолчанию:

- `max_exported_spans_per_second = inf` (отсутствуют какие-либо ограничения)
- `max_spans_in_batch = 150`
- `max_bytes_in_batch = 20000000`
- `max_batch_accumulation_milliseconds = 1000`
- `span_export_timeout_seconds = inf` (никакие спаны не удаляются uploader'ом)
- `max_export_requests_inflight = 1`

Секция `uploader` может полностью отсутствовать, в таком случае для каждого параметра будет использоваться значение по умолчанию.

> **Примечание**
>
> Uploader – локальный для узла компонент, как следствие, описанные ограничения применяются к каждому узлу отдельно, а не ко всему кластеру в целом.

External throttling

Пример секции

```
tracing_config:
  # ...
  external_throttling:
    - scope:
        database: /Root/db1
      max_traces_per_minute: 60
      max_traces_burst: 3
```

Описание

YDB поддерживает передачу внешних trace-id для построения цельной трассы запроса. Способ передачи внешнего trace-id описан на странице Передача внешнего trace-id в YDB. Для избежания перегрузки коллектора, в YDB есть механизм ограничения количества трассируемых извне запросов. Ограничения описываются в данной секции и являются последовательностью правил, каждое правило содержит:

- `scope` – набор селекторов для фильтрации запроса.
- `max_traces_per_minute` – наибольшее среднее количество запросов в минуту, трассируемое по данному правилу. Ожидается положительное целое число.
- `max_traces_burst` – максимальный всплеск количества трассируемых извне запросов. Ожидается неотрицательное целое число.

Обязательным параметром является только `max_traces_per_minute`.

Подробное описание этих опций приведено в секции Семантика правил

Секция `external_throttling` не является обязательной, в случае ее отсутствия все trace-id в запросах **игнорируются** (никакие внешние трассы не продолжаются).

Данную секцию можно изменять без перезагрузки узла с помощью механизма конфигурации.

Sampling

Пример секции

```
tracing_config:
  # ...
  sampling:
    - fraction: 0.01
```

```
      level: 10
      max_traces_per_minute: 5
      max_traces_burst: 2
    - scope:
        request_types:
          - KeyValue.ExecuteTransaction
          - KeyValue.Read
      fraction: 0.1
      level: 15
      max_traces_per_minute: 5
      max_traces_burst: 2
```

Описание

Для диагностирования проблем в системе, может быть полезно посмотреть на пример трассы запроса вне зависимости от того, трассируют пользователи свои запросы, или нет. Для этого в YDB есть механизм сэмплирования запросов. Для сэмплированного запроса генерируется случайный trace-id. Данная секция контролирует сэмплирование запросов в формате аналогичном `external_throttling`, каждое правило имеет два дополнительные поля:

- `fraction` – доля запросов, сэмплированных по данному правилу. Ожидается дробное число от 0 до 1.
- `level` — уровень детализации трассы. Ожидается целое число от 0 до 15 включительно. Подробнее этот параметр описан в секции Уровни детализации

Оба поля являются обязательными.

Секция `sampling` не является обязательной, в случае ее отсутствия никакие запросы не будут сэмплироваться.

Данную секцию можно изменять без перезагрузки узла с помощью механизма динамической конфигурации.

Семантика правил

Селекторы

Каждое правило включает в себя опциональное поле `scope` с набором селекторов, которые определяют, к каким запросам применяется данное правило. На данный момент поддерживаются селекторы:

- `request_types`

  Принимает список типов запросов. Запрос подходит под данный селектор если его тип содержится в списке.

Возможные значения

- KeyValue.CreateVolume
- KeyValue.DropVolume
- KeyValue.AlterVolume
- KeyValue.DescribeVolume
- KeyValue.ListLocalPartitions
- KeyValue.AcquireLock
- KeyValue.ExecuteTransaction
- KeyValue.Read
- KeyValue.ReadRange
- KeyValue.ListRange
- KeyValue.GetStorageChannelStatus
- Table.CreateSession
- Table.KeepAlive
- Table.AlterTable
- Table.CreateTable
- Table.DropTable
- Table.DescribeTable
- Table.CopyTable
- Table.CopyTables
- Table.RenameTables
- Table.ExplainDataQuery
- Table.ExecuteSchemeQuery
- Table.BeginTransaction
- Table.DescribeTableOptions
- Table.DeleteSession
- Table.CommitTransaction
- Table.RollbackTransaction
- Table.PrepareDataQuery
- Table.ExecuteDataQuery
- Table.BulkUpsert
- Table.StreamExecuteScanQuery
- Table.StreamReadTable
- Table.ReadRows
- Query.ExecuteQuery
- Query.ExecuteScript
- Query.FetchScriptResults
- Query.CreateSession
- Query.DeleteSession
- Query.AttachSession
- Query.BeginTransaction
- Query.CommitTransaction

- Query.RollbackTransaction
- Discovery.WhoAmI
- Discovery.NodeRegistration
- Discovery.ListEndpoints

> **Примечание**
>
> Трассировка поддерживается не только для типов запросов, приведенных выше. В этом списке перечислены типы запросов, которые поддерживатюся селектором `request_types`.

> **Важно**
>
> Обратите внимание, что QueryService API экспериментальное и может быть изменено в будущем.

- `database`

  Фильтрует запросы к указанной базе данных.

Запрос подходит под правило, если он подходит под все селекторы. `scope` может отсутствовать, это равносильно пустому набору селекторов, все запросы будут попадать под это правило.

Rate limiting

Параметры `max_traces_per_minute` и `max_traces_burst` используются для ограничения сверху на количество запросов. В случае sampling они ограничивают количество запросов, сэмплированных по данному правилу. В случае external throttling они ограничивают количество внешних трасс, которые попадают в систему.

Для ограничений используется вариация leaky bucket с размером бакета равным `max_traces_burst + 1`. Например, если `max_traces_per_minute = 60` и `max_traces_burst = 0`, то при потоке в 10000 запросов в минуту будет трассироваться один запрос каждую секунду. Если же `max_traces_burst = 20`, то при поступлении аналогичного потока запросов первые 21 будут оттрассированы, далее аналогично будет трассироваться один запрос в секунду.

> **Важно**
>
> Ограничения на количество трассируемых запросов локальные для узла кластера. Например, если на каждом узле кластера есть правило, в котором указано `max_traces_per_minute = 1`, по этому правилу будет трассироваться не больше одного запроса в минуту **с каждого узла кластера**.

Уровни детализации

Как и в случае с логами, для диагностирования большинства проблем с системой не требуется наличие максимально подробной трассы, поэтому в YDB каждый спан имеет свой уровень, описываемый целым числом от 0 до 15 включительно. Каждое правило в секции `sampling` должно включать в себя уровень детализации генерируемой трассы (`level`), в нее будут включаться спаны с уровнем меньшим, либо равным `level`.

В разделе об архитектуре YDB приводится разделение системы на 5 слоев:

Слой	Копоненты
1	gRPC Proxies
2	Query Processor
3	Distributes Transactions
4	Tablet, System tablet
5	Distributed Storage

Для каждого слоя существует 7 уровней детализации:

Уровень	Значение
`Off`	Отсутствие трассировки
`TopLevel`	Самая низкая детализация, не более 2 спанов на каждый запрос к компоненту
`Basic`	Спаны главных операций компонента
`Detailed`	Самая высокая детализация, применимая для диагностирования проблем в production
`Diagnostic`	Детальная отладочная информация для разработчиков
`Trace`	Очень детальная отладочная информация

Ниже приведено распределение уровней детализации слоев системы по уровням детализации трассы:

Уровень детализации трассы	gRPC Proxies	Query Processor	Distributed Transactions	Tablets	Distributed Storage
0	`TopLevel`	`Off`	`Off`	`Off`	`Off`
1	`TopLevel`	**`TopLevel`**	`Off`	`Off`	`Off`

2	TopLevel	TopLevel	**TopLevel**	Off	Off
3	TopLevel	TopLevel	TopLevel	**TopLevel**	Off
4	TopLevel	TopLevel	TopLevel	TopLevel	**TopLevel**
5	**Basic**	TopLevel	TopLevel	TopLevel	TopLevel
6	Basic	**Basic**	TopLevel	TopLevel	TopLevel
7	Basic	Basic	**Basic**	TopLevel	TopLevel
8	Basic	Basic	Basic	**Basic**	TopLevel
9	Basic	Basic	Basic	Basic	**Basic**
10	**Detailed**	**Detailed**	Basic	Basic	Basic
11	Detailed	Detailed	**Detailed**	Basic	Basic
12	Detailed	Detailed	Detailed	**Detailed**	Basic
13	Detailed	Detailed	Detailed	Detailed	**Detailed**
14	**Diagnostic**	**Diagnostic**	**Diagnostic**	**Diagnostic**	**Diagnostic**
15	**Trace**	**Trace**	**Trace**	**Trace**	**Trace**

Правила

External Throttling

Семантика каждого правила такова: оно выделяет квоту на количество запросов данной категории. Например, если секция `external_throttling` выглядит следующим образом:

```
tracing_config:
  external_throttling:
    - max_traces_per_minute: 60
    - scope:
        request_types:
          - KeyValue.ReadRange
      max_traces_per_minute: 20
```

При достаточном потоке запросов с внешним trace-id, будут трассироваться хотя бы 60 запросов в минуту и хотя бы 20 запросов в минуту типа `KeyValue.ReadRange`. Всего трассироваться будут не более 80 запросов в минуту.

Алгоритм решения следующий: для запроса, имеющего внешний trace-id, определяется набор правил, под которые подходит данный запрос. Запрос потребляет квоту у всех правил, у которых она еще есть. Запрос не трассируется только в том случае, если ни в у одного из правил не осталось квоты.

Sampling

В случае сэмплирования семантика правила аналогична: при достаточно маленьком потоке запросов данной категории, будет сэплироваться хотя бы `fraction` запросов с уровнем хотя бы `level`.

Алгоритм решения аналогичный: для запроса, не имеющего внешнего trace-id (либо из-за его изначального отсутствия, либо из-за принятого ранее решения о не трассировании данного запроса), определяется набор правил, под которые подходит данный запрос. Запрос потребляет квоту у всех правил, у которых она еще есть и которые случайным образом "решили" его сэмплировать. Если ни по одному правилу запрос не нужно сэмлировать (во всех правилах, которые случайным образом "решили" сэмплировать запрос, не осталось квоты), он не сэмплируется. В противном случае уровень детализации определяется, как максимальный среди правил, в квоту которых попал запрос.

Например, при данной конфигурации `sampling`:

```
tracing_config:
  sampling:
    - scope:
        database: /Root/db1
      fraction: 0.5
      level: 5
      max_traces_per_minute: 100
    - scope:
        database: /Root/db1
      fraction: 0.01
      level: 15
      max_traces_per_minute: 5
```

При достаточно маленьком потоке запросов к базе `/Root/db1`, будут сэмплироваться:

- 1% запросов с уровнем детализации 15
- 49.5% запросов с уровнем детализации 5

При достаточно большом потоке запросов к базе `/Root/db1`, будут сэмплироваться:

- 5 запросов в минуту с уровнем детализации 15
- от 95 до 100 запросов в минуту с уровнем детализации 5

Передача внешнего trace-id в YDB

gRPC API

YDB поддерживает передачу внешних trace-id для построения цельной трассы операции. Передача trace-id производится согласно спецификации trace context — через заголовок `traceparent` gRPC запроса должна передаваться строка вида `00-0af7651916cd43dd8448eb211c80319c-b7ad6b7169203331-01`. Она состоит из 4 частей, разделенных символом `-`:

1. Версия – на момент написания документации спецификация определяет только версию 00.
2. Trace id – идентификатор трассы, частью которой является запрос.
3. Parent id – идентификатор родительского спана.
4. Флаги – набор рекомендаций по работе с переданными в заголовке данными.

Только версия 00 поддерживается, флаги игнорируются. Если заголовок не соответствует спецификации, он игнорируется. Все получаемые таким образом трассы имеют уровень детализации 13 (все компоненты трассируются с уровнем `Detailed`).

> **Важно**
>
> При наличии секции `external_throttling` и потоке запросов, превышающем установленные лимиты, могут трассироваться не все запросы. При отсутствии секции `external_throttling` заголовок `traceparent` **игнорируется** и никакие внешние трассы не продолжаются.

Поддержка SDK

Некоторые SDK поддерживают передачу trace-id, их список, а так же примеры использования вы можете найти на странице Трассировка с OpenTelemetry.

Обзор утилиты YDB DSTool

С помощью утилиты YDB DSTool вы можете управлять дисковой подсистемой кластера YDB. Чтобы установить и настроить утилиту, выполните инструкцию.

Утилита YDB DSTool включает следующие команды:

Команда	Описание
device list	Вывести список устройств хранения.
pdisk add-by-serial	Добавить PDisk в набор по серийному номеру.
pdisk remove-by-serial	Удалить PDisk из набора по серийному номеру.
pdisk set	Задать параметры PDisk'а.
pdisk list	Вывести список PDisk'ов.
pdisk populate	Разместить те же VDisk'и на другом PDisk для тестирования устройства.
vdisk evict	Переместить VDisk'и на другие PDisk'и.
vdisk remove-donor	Удалить VDisk-донор.
vdisk wipe	Очистить VDisk'и
vdisk list	Вывести список VDisk'ов.
group add	Добавить группы хранения в пул.
group check	Проверить группы хранения.
group show blob-info	Вывести информацию о блобе.
group show usage-by-tablets	Вывести информацию об использовании таблеток группами.
group state	Вывести или изменить состояние группы хранения.
group take-snapshot	Сделать снапшот метаданных группы хранения.
group list	Вывести список групп хранения.
pool list	Вывести список пулов.
box list	Вывести список наборов PDisk'ов.
node list	Вывести список узлов.
cluster balance	Переместить VDisk'и с перегруженных PDisk'ов.
cluster get	Вывести параметры кластера.
cluster set	Задать параметры кластера.
cluster workload run	Создать рабочую нагрузку для тестирования модели отказа.
cluster list	Вывести информацию о кластере.

Обзор утилиты YDB DSTool

С помощью утилиты YDB DSTool вы можете управлять дисковой подсистемой кластера YDB. Чтобы установить и настроить утилиту, выполните инструкцию.

Утилита YDB DSTool включает следующие команды:

Команда	Описание
device list	Вывести список устройств хранения.
pdisk add-by-serial	Добавить PDisk в набор по серийному номеру.
pdisk remove-by-serial	Удалить PDisk из набора по серийному номеру.
pdisk set	Задать параметры PDisk'а.
pdisk list	Вывести список PDisk'ов.
pdisk populate	Разместить те же VDisk'и на другом PDisk для тестирования устройства.
vdisk evict	Переместить VDisk'и на другие PDisk'и.
vdisk remove-donor	Удалить VDisk-донор.
vdisk wipe	Очистить VDisk'и
vdisk list	Вывести список VDisk'ов.
group add	Добавить группы хранения в пул.
group check	Проверить группы хранения.
group show blob-info	Вывести информацию о блобе.
group show usage-by-tablets	Вывести информацию об использовании таблеток группами.
group state	Вывести или изменить состояние группы хранения.
group take-snapshot	Сделать снапшот метаданных группы хранения.
group list	Вывести список групп хранения.
pool list	Вывести список пулов.
box list	Вывести список наборов PDisk'ов.
node list	Вывести список узлов.
cluster balance	Переместить VDisk'и с перегруженных PDisk'ов.
cluster get	Вывести параметры кластера.
cluster set	Задать параметры кластера.
cluster workload run	Создать рабочую нагрузку для тестирования модели отказа.
cluster list	Вывести информацию о кластере.

Установка YDB DSTool

Linux

Чтобы установить YDB DSTool:

1. Выполните команду:

   ```
   curl -sSL 'https://install.ydb.tech/dstool' | bash
   ```

   Скрипт установит YDB DSTool. Если скрипт запускается из оболочки `bash` или `zsh`, он также добавит исполняемый файл `ydb-dstool` в переменную окружения `PATH`. В противном случае вы можете запускать его из папки `~/ydb/bin` или добавить в `PATH` вручную.
2. Чтобы обновить переменные окружения, перезапустите командную оболочку.
3. Проверьте работу, выполнив команду вывода информации о кластере:

   ```
   ydb-dstool -e <bs_endpoint> cluster list
   ```

   - `bs_endpoint` — URI HTTP-эндпоинта кластера — это тот же самый эндпоинт, который обслуживает YDB UI. Пример: `http://localhost:8765`.

   Результат:

   ```
   ┌───────┬───────┬───────┬────────┬────────┬───────┬────────┐
   | Hosts | Nodes | Pools | Groups | VDisks | Boxes | PDisks |
   ├───────┼───────┼───────┼────────┼────────┼───────┼────────┤
   | 8     | 16    | 1     | 5      | 40     | 1     | 32     |
   └───────┴───────┴───────┴────────┴────────┴───────┴────────┘
   ```

macOS

Чтобы установить YDB DSTool:

1. Выполните команду:

   ```
   curl -sSL 'https://install.ydb.tech/dstool' | bash
   ```

   Скрипт установит YDB DSTool. Если скрипт запускается из оболочки `bash` или `zsh`, он также добавит исполняемый файл `ydb-dstool` в переменную окружения `PATH`. В противном случае вы можете запускать его из папки `~/ydb/bin` или добавить в `PATH` вручную.
2. Чтобы обновить переменные окружения, перезапустите командную оболочку.
3. Проверьте работу, выполнив команду вывода информации о кластере:

   ```
   ydb-dstool -e <bs_endpoint> cluster list
   ```

   - `bs_endpoint` — URI HTTP-эндпоинта кластера — это тот же самый эндпоинт, который обслуживает YDB UI. Пример: `http://localhost:8765`.

   Результат:

   ```
   ┌───────┬───────┬───────┬────────┬────────┬───────┬────────┐
   | Hosts | Nodes | Pools | Groups | VDisks | Boxes | PDisks |
   ├───────┼───────┼───────┼────────┼────────┼───────┼────────┤
   | 8     | 16    | 1     | 5      | 40     | 1     | 32     |
   └───────┴───────┴───────┴────────┴────────┴───────┴────────┘
   ```

Windows

Чтобы установить YDB DSTool:

1. Выполните команду:
   - **PowerShell**:

     ```
     iex (New-Object System.Net.WebClient).DownloadString('https://install.ydb.tech/dstool-windows')
     ```

   - **CMD**:

     ```
     @"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -Command "iex ((New-Object System.Ne
     t.WebClient).DownloadString('https://install.ydb.tech/dstool-windows'))"
     ```

2. Укажите, нужно ли добавить путь к `ydb-dstool` в переменную окружения `PATH`:

   ```
   Add ydb-dstool installation dir to your PATH? [Y/n]
   ```

3. Чтобы обновить переменные окружения, перезапустите командную оболочку.

Примечание

YDB DSTool использует символы Юникода в выводе некоторых команд. При некорректном отображении таких символов в консоли Windows, переключите кодировку на UTF-8:

```
chcp 65001
```

4. Проверьте работу, выполнив команду вывода информации о кластере:

```
ydb-dstool -e <bs_endpoint> cluster list
```

   - `bs_endpoint` — URI HTTP-эндпоинта кластера — это тот же самый эндпоинт, который обслуживает YDB UI. Пример: `http://localhost:8765`.

Результат:

```
┌───────┬───────┬───────┬────────┬────────┬───────┬────────┐
| Hosts | Nodes | Pools | Groups | VDisks | Boxes | PDisks |
├───────┼───────┼───────┼────────┼────────┼───────┼────────┤
| 8     | 16    | 1     | 5      | 40     | 1     | 32     |
└───────┴───────┴───────┴────────┴────────┴───────┴────────┘
```

Глобальные параметры

Глобальные параметры являются общими для всех подкоманд утилиты YDB DSTool.

Параметр	Описание
`-?`, `-h`, `--help`	Вывести встроенную справку.
`-v`, `--verbose`	Получить подробный вывод при выполнении команды.
`-q`, `--quiet`	Подавить вывод несущественных сообщений при выполнении команды.
`-n`, `--dry-run`	Выполнить пробный запуск команды.
`-e`, `--endpoint`	Эндпоинт для подключения к кластеру YDB в формате `[PROTOCOL://]HOST[:PORT]`. Значения по умолчанию: протокол — `http`, порт — `8765`.
`--grpc-port`	gRPC-порт для вызова процедур.
`--mon-port`	Порт для просмотра данных HTTP-мониторинга в формате JSON.
`--token-file`	Путь к файлу с Access Token.
`--ca-file`	Путь к файлу корневого PEM-сертификата для TLS-соединения.
`--http`	Использовать HTTP для подключения к Blob Storage вместо gRPC.
`--http-timeout`	Тайм-аут на операции ввода-вывода сокета во время HTTP(s)-запросов.
`--insecure`	Разрешить небезопасную передачу данных по HTTPS. В этом режиме не проверяются SSL-сертификат и имя хоста.

device list

С помощью подкоманды `device list` вы можете вывести список устройств хранения, доступных на кластере YDB.

Общий вид команды:

```
ydb-dstool [global options ...] device list [list options ...]
```

- `global options` — глобальные параметры.
- `list options` — параметры подкоманды.

Посмотрите описание команды вывода списка устройств:

```
ydb-dstool device list --help
```

Параметры подкоманды

Параметр	Описание
`-H`, `--human-readable`	Вывести данные в человекочитаемом формате.
`--sort-by`	Колонка сортировки. Одно из значений: `SerialNumber`, `FQDN`, `Path`, `Type`, `StorageStatus`, `NodeId:PDiskId`.
`--reverse`	Использовать обратный порядок сортировки.
`--format`	Формат вывода. Одно из значений: `pretty`, `json`, `tsv`, `csv`.
`--no-header`	Не выводить строку с именами колонок.
`--columns`	Список колонок, которые нужно вывести. Одно или комбинация значений: `SerialNumber`, `FQDN`, `Path`, `Type`, `StorageStatus`, `NodeId:PDiskId`.
`-A`, `--all-columns`	Вывести все колонки.

Примеры

Следующая команда выведет список устройств, доступных на кластере:

```
ydb-dstool -e node-5.example.com device list
```

Результат:

```
┌─────────────────────┬──────────────────────┬──────────────────────────────────┬──────┬──────────────────────────────
─┬─────────────────┐
| SerialNumber       | FQDN                | Path                            | Type | StorageStatus
| NodeId:PDiskId |
├─────────────────────┼──────────────────────┼──────────────────────────────────┼──────┼──────────────────────────────
─┼─────────────────┤
| PHLN123301H41P2BGN | node-1.example.com | /dev/disk/by-partlabel/nvme_04 | NVME | FREE
| NULL           |
| PHLN123301A62P2BGN | node-6.example.com | /dev/disk/by-partlabel/nvme_03 | NVME | PDISK_ADDED_BY_DEFINE_BOX
| [6:1001]       |
...
```

pdisk populate

С помощью подкоманды `pdisk populate` можно переместить выбранный набор VDisk'ов на один PDisk. Команда позволяет сохранить список активных VDisk'ов исходного PDisk в файл-snapshot, а затем использовать этот файл для заполнения целевого PDisk.

> **Важно**
>
> Используйте `pdisk populate` для контролируемого тестирования устройств: команда позволяет разместить на новом устройстве точно такой же набор VDisk'ов и сравнить, как старое и новое устройства справляются с этой нагрузкой.

Blob Storage Controller проверяет все выбранные VDisk'и и планирует их перемещение в рамках одной атомарной транзакции конфигурации. Если хотя бы один VDisk невозможно переместить, конфигурация не изменяется. Сам перенос данных продолжается асинхронно после применения транзакции.

Общий вид команды:

```
ydb-dstool [global options ...] pdisk populate [populate options ...]
```

- `global options` — глобальные параметры.
- `populate options` — параметры подкоманды.

Посмотрите описание команды:

```
ydb-dstool pdisk populate --help
```

Параметры подкоманды

Параметр	Описание
`--snapshot-from-pdisk <NodeId:PDiskId>`	Режим создания snapshot. Собрать активные VDisk'и с указанного PDisk. VDisk'и в режиме донора пропускаются.
`-d`, `--destination-pdisk <NodeId:PDiskId>`	Режим заполнения. Переместить на указанный PDisk VDisk'и, перечисленные в файле `--snapshot-file`.
`--snapshot-file <PATH>`	В режиме создания snapshot записать список VDisk'ов в формате JSON в указанный файл. В режиме заполнения прочитать список VDisk'ов из этого файла. Параметр обязателен в режиме заполнения.
`--suppress-donor-mode`	Не оставлять прежние расположения VDisk'ов в режиме донора после перемещения. Параметр доступен только в режиме заполнения.
`--format <FORMAT>`	Формат вывода: `pretty` (по умолчанию) или `json`.

Необходимо указать ровно один из параметров: `--snapshot-from-pdisk` или `--destination-pdisk`.

Пример

Сначала сохраните список активных VDisk'ов на PDisk `[1:1000]` в файл-snapshot:

```
ydb-dstool -e node-1.example.com pdisk populate \
  --snapshot-from-pdisk '[1:1000]' \
  --snapshot-file /tmp/pdisk-1-1000.json
```

Файл-snapshot имеет формат:

```
{
  "pdisk_id": "[1:1000]",
  "vdisk_ids": [
    "[80000001:_:0:0:0]",
    "[80000002:_:0:1:0]"
  ]
}
```

Проверьте содержимое snapshot, а затем переместите перечисленные VDisk'и на PDisk `[2:1000]`:

```
ydb-dstool -e node-1.example.com pdisk populate \
  --destination-pdisk '[2:1000]' \
  --snapshot-file /tmp/pdisk-1-1000.json
```

Команда пропускает VDisk'и с `GroupId=0`, потому что их нельзя перемещать. Если snapshot содержит только такие VDisk'и, команда завершается с ошибкой и не изменяет конфигурацию.

Обзор утилиты ydbops

> **Примечание**
>
> Утилита `ydbops` находится в активной разработке. В редких случаях обратная совместимость может быть нарушена.

Утилита `ydbops` облегчает выполнение объемных сценариев на кластерах YDB. Утилита поддерживает кластеры, развернутые с помощью Ansible, Kubernetes или вручную.

Смотрите также

- Для установки утилиты следуйте инструкциям.
- Для настройки утилиты смотрите справочник по конфигурации.
- Исходный код `ydbops` доступен на GitHub.

Поддерживаемые сценарии

- Выполнение перезагрузки кластера.

Сценарии в разработке

- Запрос разрешения на вывод узлов YDB на обслуживание без нарушения инвариантов модели отказа YDB.

Обзор утилиты ydbops

> **Примечание**
>
> Утилита `ydbops` находится в активной разработке. В редких случаях обратная совместимость может быть нарушена.

Утилита `ydbops` облегчает выполнение объемных сценариев на кластерах YDB. Утилита поддерживает кластеры, развернутые с помощью Ansible, Kubernetes или вручную.

Смотрите также

- Для установки утилиты следуйте инструкциям.
- Для настройки утилиты смотрите справочник по конфигурации.
- Исходный код `ydbops` доступен на GitHub.

Поддерживаемые сценарии

- Выполнение перезагрузки кластера.

Сценарии в разработке

- Запрос разрешения на вывод узлов YDB на обслуживание без нарушения инвариантов модели отказа YDB.

Установка ydbops

> **Примечание**
>
> Утилита `ydbops` находится в активной разработке. В редких случаях обратная совместимость может быть нарушена.

Загрузка бинарного файла со страницы релизов

Скомпилированные исполняемые файлы можно скачать с YDBOps Releases.

Сборка из исходного кода

1. Склонируйте репозиторий `ydbops` с GitHub:

   ```
   git clone https://github.com/ydb-platform/ydbops.git
   ```

2. Сборка исполняемого файла может быть выполнена одним из двух вариантов:
   2.1. Сборка с использованием компилятора Go
   2.2. Сборка внутри docker-контейнера

Для сборки с помощью Docker используется официальный docker image для Golang v1.22, это гарантирует то, что сборка будет проведена успешно. Описание работы docker-контейнера в `Dockerfile` репозитория. Так же сборка в Docker проводит дополнительные манипуляции: выполняет проверки линтером, подставляет версию для сборки ydbops, чтобы она прописалась в исполняемом файле ydbops.

Сборка с использованием компилятора Go

Требования

Установите Go. Рекомендуемая версия Go — 1.22.

Компиляция

Вызовите `go build` в корневой папке репозитория:

```
go build
```

Исполняемый файл `ydbops` будет доступен в корневой папке репозитория.

Установка

Вручную переместите исполняемый файл в требуемый каталог или используйте утилиту `make` для установки в каталог `install_folder` из текущего каталога:

```
make install INSTALL_DIR=install_folder BUILD_DIR=.
```

Сборка внутри docker-контейнера

Требования

- make;
- docker engine.

Компиляция

Выполните в папке репозитория команду:

```
make build-in-docker
```

Исполняемый файл `ydbops` будет доступен в корневой папке `bin`. Автоматически собираются исполняемые файлы для Linux и MacOS (arm64, amd64).

Исполняемый файл	Платформа
ydbops	Linux(amd64)
ydbops_darwin_amd64	MacOS(amd64)
ydbops_darwin_arm64	MacOS(arm64)

Установка

Для установки исполняемого файла выполните команду `make`.

Дополнительные параметры:

- `INSTALL_DIR`: каталог, в который будет установлен исполняемый файл. По умолчанию: `~/ydb/bin`.
- `BUILD_DIR`: каталог, содержащий скомпилированный исполняемый файл. Этот параметр используется, если файл собран самостоятельно. Например, если файл находится в текущем каталоге, используйте `BUILD_DIR=.`.

```
make install [INSTALL_DIR=<путь_к_каталогу>] [BUILD_DIR=<путь_к_каталогу>]
```

Пример команды для установки в каталог `install_folder` из текущего каталога:

```
make install INSTALL_DIR=install_folder BUILD_DIR=.
```

Настройка ydbops

> **Примечание**
>
> Утилита `ydbops` находится в активной разработке. В редких случаях обратная совместимость может быть нарушена.

`ydbops` можно запустить, указав все необходимые аргументы командной строки при вызове команды.
Однако, есть две функции, которые позволяют избежать повторения часто используемых аргументов:

- Файл конфигурации
- Переменные окружения

Файл конфигурации

Файл конфигурации для `ydbops` — это файл в формате YAML, содержащий несколько профилей. Профили для `ydbops` работают так же, как и профили в YDB CLI, однако они не взаимозаменяемы, поскольку могут содержать параметры, которые актуальны для `ydbops`, но не имеют смысла для YDB CLI, и наоборот.

Расположение файла конфигурации по умолчанию следует той же конвенции, что и YDB CLI, и находится в той же папке в подкаталоге `ydbops`. Для сравнения:

- файл конфигурации по умолчанию для YDB CLI : `$HOME/.ydb/config/config.yaml`
- файл конфигурации по умолчанию для CLI `ydbops` находится в той же папке, в подкаталоге `ydbops` : `$HOME/ydb/ydbops/config/config.yaml`

Некоторые параметры командной строки могут быть записаны в файл конфигурации вместо того, чтобы указывать их каждый раз напрямую при вызове `ydbops`.

Примеры

Вызов команды `ydbops restart` без профиля:

```
ydbops restart \
 -e grpc://<hostname>:2135 \
 --kubeconfig ~/.kube/config \
 --k8s-namespace <k8s-namespace> \
 --user admin \
 --password-file ~/<password-file> \
 --tenant --tenant-list=<tenant-name>
```

Вызов той же команды `ydbops restart` с включенными параметрами профиля делает команду намного короче:

```
ydbops restart \
 --config-file ./config.yaml \
 --tenant --tenant-list=<tenant-name>
```

Для указанного выше вызова предполагается наличие следующего `config.yaml`:

```
current-profile: my-profile
profiles:
  my-profile:
    endpoint: grpcs://<hostname>:2135
    ca-file: ~/<ca-certificate-file>
    user: <username>
    password-file: ~/<password-file>
    k8s-namespace: <k8s-namespace>
    kubeconfig: ~/.kube/config
```

Команды управления профилями

В настоящее время `ydbops` не поддерживает создание, изменение и активацию профилей через команды CLI так, как это делает YDB CLI.

Файл конфигурации необходимо создать и редактировать вручную.

Максимально полный файл конфигурации

Пример файла конфигурации со всеми возможными параметрами и примерами значений (скорее всего, не все опции будут нужны одновременно):

```
# специальный ключ `current-profile` может быть указан для
# использования в качестве активного профиля по умолчанию при вызове CLI
current-profile: my-profile

# Описание профилей помещается в ключ `profiles`
profiles:
  my-profile:
    endpoint: grpcs://your.ydb.cluster.fqdn:2135

    # расположение файла CA при использовании grpcs в endpoint
    ca-file: /path/to/custom/ca/file

    # имя пользователя и файл пароля, если используется аутентификация при помощи логина и пароля:
    user: your-ydb-user-name
    password-file: /path/to/password-file
```

```
# если используется аутентификация при помощи access token
token-file: /path/to/ydb/token

# если идет работа с YDB кластерами под Kubernetes, можно указать путь к kubeconfig:
kubeconfig: /path/to/kube/config
k8s-namespace: <k8s-namespace>
```

Переменные окружения

Кроме того, можно указать несколько переменных окружения вместо передачи аргументов командной строки или использования файлов конфигурации.

Для справки о приоритете при определении опции в нескольких местах, вызовите `ydbops --help`.

- `YDB_TOKEN` может быть передана вместо флага `--token-file` или параметра профиля `token-file`.
- `YDB_PASSWORD` может быть передана вместо флага `--password-file` или параметра профиля `password-file`.
- `YDB_USER` может быть передана вместо флага `--user` или параметра профиля `user`.

Смотрите также

- Обзор утилиты ydbops
- Установка ydbops
- Выполнение перезапуска кластера YDB с помощью ydbops

Выполнение перезапуска кластера YDB с помощью ydbops

> **Примечание**
>
> Утилита `ydbops` находится в активной разработке. В редких случаях обратная совместимость может быть нарушена.

`ydbops` можно использовать для выполнения перезагрузки кластера: перезапуска всех или некоторых узлов кластера YDB при сохранении доступности кластера. Почему это не тривиально и требует специальной утилиты, объясняется в статье про обслуживание кластера без потери доступности.

Подкоманда, ответственная за эту операцию, — `ydbops restart`.

Общий алгоритм

Существует ряд опций для `ydbops restart`, которые действуют как фильтры. Фильтры неявно соединены логическим «и», то есть при указании нескольких фильтров будут выбраны только те узлы, которые удовлетворяют всем из них одновременно. Следовательно, при отсутствии фильтров перезапускаются все узлы.

Существуют два особых фильтра, которые являются исключением из этого правила: `--storage` и `--tenant`:

- Указание только `--storage` фильтрует только storage-узлы.
- Указание только `--tenant` фильтрует только tenant-узлы.
- Указание обоих фильтров выбирает все узлы, что эквивалентно отсутствию этих двух фильтров.

Алгоритм всегда работает в две фазы:

- Сначала он определяет, соответствуют ли какие-либо storage-узлы условиям перезапуска, и перезапускает только эти узлы.
- После того, как все storage-узлы были перезапущены (или ни один из storage-узлов не подошёл под условия фильтрации), процесс повторяется для tenant-узлов.

Примеры

Следующие примеры предполагают, что необходимые параметры подключения указаны (такие как `--endpoint` или аутентификационные опции).

Перезапуск всех узлов в кластере

Следующая команда перезапустит все узлы в кластере: сначала storage-узлы, а затем tenant-узлы.

```
ydbops restart
```

Перезапуск только storage-узлов или tenant-узлов

Можно перезапустить только storage-узлы:

```
ydbops restart --storage
```

Или только tenant-узлы:

```
ydbops restart --tenant
```

Можно перезапустить только узлы определенных tenant-ов, указав `--tenant-list`:

```
ydbops restart --tenant-list=</domain/database_name_1>,</domain/database_name_2>,...
```

Перезапуск только определённых узлов

Можно перезапустить узлы только на определённых хостах, указав FQDN с помощью опции `--hosts`:

```
ydbops restart --hosts=<node1.some.zone>,<node2.some.zone>
```

Или указав идентификаторы узлов напрямую:

```
ydbops restart --hosts=1,2,3
```

Перезапуск по времени работы узла

Можно перезапустить только узлы, которые имеют определённое время работы, используя опцию `--started`.

Аргумент опции должен быть заключён в кавычки, иначе shell может интерпретировать `>` или `<` как перенаправление ввода/вывода. См. пример:

```
ydbops restart --started '>2024-03-13T17:00:00Z'
```

Например, это может быть удобно для перезапуска только тех узлов, которые работают более нескольких минут, так как остальные уже перезапустились.

Перезапуск по версии YDB на узлах

Можно перезапустить только те узлы, версия которых равна (`==`), не равна (`!=`), больше (`>`) или меньше (`<`) желаемой.

Аргумент опции должен быть заключён в кавычки, иначе shell может интерпретировать `>` или `<` как перенаправление ввода/вывода. См. пример:

```
ydbops restart --version '>24.1.2'
ydbops restart --version '<24.1.2'
ydbops restart --version '!=24.1.2'
ydbops restart --version '==24.1.2'
```

Хотфикс-версии (например, `ydb-stable-24-1-14-hotfix-9`) имеют те же основные, минорные и патч-номера, что и соответствующая не-хотфикс версия `24.1.14`.

Например, версия `ydb-stable-24-1-14-hotfix-9` в операциях сравнения будет идентична версии `ydb-stable-24-1-14`.

Справка по работе YDB в Docker

В данном разделе собраны подробности по работе с сервером YDB, поставляемым в виде Docker-контейнера.

Документация по Docker-контейнеру содержит следующие разделы:

- Именование тегов докер-образа `ydbplatform/local-ydb`
- Установка Docker для работы с YDB
- Запуск YDB в Docker
- Настройка Docker-контейнера YDB
- Пользовательские скрипты инициализации
- Остановка Docker

Именование тегов докер-образа `ydbplatform/local-ydb`

Для докер-образа ydbplatform/local-ydb применяются следующие правила именования для тегов:

Тег	Описание
`latest`	Соответствует *stable*-версии YDB, проверенной на известных продакшен-кластерах YDB. Тег **latest** пересобирается, когда в проекте ydb появляется запись о новом релизе YDB.
`edge`	Кандидат на следующий *stable*, то есть та версия, которая в данный момент тестируется. В edge можно попробовать новые фичи YDB, но также не следует ожидать стабильности данной сборки.
`trunk`, `main`, `nightly`	Последняя версия YDB из кода в основной ветке разработки. Содержит самые последние изменения. Образ пересобирается каждую ночь.
`XX.Y`	Соответствует последней версии YDB в мажорном релизе `XX-Y` (со всеми патчами).
`XX.Y.ZZ`	Соответствует версии YDB в релизе `XX-Y-ZZ`.
`XX.Y-slim` и `XX.Y.ZZ-slim`	Соответствует версиям YDB со специальным образом сжатыми бинарными исполняемыми файлами `ydbd` и `ydb` внутри образа. Версии `*-slim` имеют значительно меньший размер образа, но при этом дольше стартуют. Для сжатия используется утилита upx.

Установка Docker для работы с YDB

Прежде чем воспользоваться Docker-образом YDB, необходимо установить и настроить Docker-окружение. Для этого можно воспользоваться официальной документацией:

- Linux
- macOS
- Windows

Альтернативные способы установки, такие как colima, также допустимы.

Установка самого YDB происходит при первом запуске.

Запуск YDB в Docker

Перед началом работы

Создайте каталог для тестирования YDB и используйте его в качестве текущего рабочего каталога:

```
mkdir ~/ydbd && cd ~/ydbd
mkdir ydb_data
mkdir ydb_certs
```

Запуск контейнера с YDB в Docker

Пример команды запуска YDB в Docker с подробными комментариями:

```
docker_args=(
    -d                              # запуск в фоне
    --rm                            # автоматическое удаление после установки
    --name ydb-local                # имя контейнера
    --hostname localhost            # хостейм
    --platform linux/amd64          # платформа
    -p 2135:2135                    # открытие внешнего доступа к grpcs порту
    -p 2136:2136                    # открытие внешнего доступа к grpc порту
    -p 8765:8765                    # открытие внешнего доступа к http порту
    -p 9092:9092                    # открытие внешнего доступа к порту, обеспечивающему Kafka-совместимость
    -v $(pwd)/ydb_certs:/ydb_certs  # директория для TLS сертификатов
    -v $(pwd)/ydb_data:/ydb_data    # рабочая директория
    -e GRPC_TLS_PORT=2135           # grpcs порт должен соответствовать тому, что опубликовано выше
    -e GRPC_PORT=2136               # grpc порт должен соответствовать тому, что опубликовано выше
    -e MON_PORT=8765                # http  порт должен соответствовать тому, что опубликовано выше
    -e YDB_KAFKA_PROXY_PORT=9092    # порт Kafka должен соответствовать тому, что опубликовано выше
    ydbplatform/local-ydb:latest
)

docker run "${docker_args[@]}"
```

Примечание

При использовании Mac с процессором Apple Silicon, набор процессорных инструкций x86_64 можно эмулировать с помощью Rosetta:

- colima с параметрами `colima start --arch aarch64 --vm-type=vz --vz-rosetta`;
- Docker Desktop с установленной и включённой Rosetta 2.

Если не включена Rosetta 2, в команду запуска Docker-контейнера необходимо добавить параметр `-e YDB_USE_IN_MEMORY_PDISKS=true`. См. Настройка Docker-контейнера YDB.

Подробнее про переменные окружения, используемые при запуске Docker-контейнера с YDB, можно узнать в разделе Настройка Docker-контейнера YDB.

При указанных в примере выше параметрах и запуске Docker локально, YDB UI YDB будет доступен по адресу http://localhost:8765.

Подробнее про остановку и удаление Docker-контейнера с YDB можно узнать в разделе Остановка Docker.

Переопределение файла конфигурации

По умолчанию при запуске контейнера Docker для YDB используется встроенный файл конфигурации, который обеспечивает стандартные параметры работы. Чтобы использовать свой конфиг, смонтируйте его в `/ydb_data/cluster/kikimr_configs/config.yaml`:

```
docker run "${docker_args[@]}" \
  -v $(pwd)/ydb_config/my-ydb-config.yaml:/ydb_data/cluster/kikimr_configs/config.yaml
```

Для пользователей, не имеющих опыта работы с Docker, важно понимать, как правильно монтировать файл конфигурации в контейнер. Ниже приведен пошаговый пример:

1. Запустите контейнер без указания файла конфигурации и без монтирования директории данных, чтобы он сгенерировал конфигурацию по умолчанию:

   ```
   docker run -d \
     --rm \
     --name ydb-local \
     --hostname localhost \
     --platform linux/amd64 \
     -v $(pwd)/ydb_certs:/ydb_certs \
     -e GRPC_TLS_PORT=2135 \
     -e GRPC_PORT=2136 \
     -e MON_PORT=8765 \
     ydbplatform/local-ydb:latest
   ```

2. Создайте директорию для файлов конфигурации и скопируйте сгенерированный файл конфигурации из контейнера прямо в нее:

   ```
   mkdir ydb_config
   docker cp ydb-local:/ydb_data/cluster/kikimr_configs/config.yaml ydb_config/my-ydb-config.yaml
   ```

3. Остановите контейнер, если он все еще запущен, и удалите созданную директорию данных:

```
docker stop ydb-local
rm -rf ydb_data
```

4. Отредактируйте скопированный файл конфигурации `ydb_config/my-ydb-config.yaml` по своему усмотрению.
5. При запуске контейнера смонтируйте отредактированный файл конфигурации в `/ydb_data/cluster/kikimr_configs/config.yaml`:

```
docker_args=(
    -d
    --rm
    --name ydb-local
    --hostname localhost
    --platform linux/amd64
    -p 2135:2135
    -p 2136:2136
    -p 8765:8765
    -v $(pwd)/ydb_certs:/ydb_certs
    -v $(pwd)/ydb_data:/ydb_data
    -v $(pwd)/ydb_config/my-ydb-config.yaml:/ydb_data/cluster/kikimr_configs/config.yaml
    -e GRPC_TLS_PORT=2135
    -e GRPC_PORT=2136
    -e MON_PORT=8765
    ydbplatform/local-ydb:latest
)

docker run "${docker_args[@]}"
```

В этом примере локальный файл `ydb_config/my-ydb-config.yaml` подменяет конфигурацию по умолчанию внутри контейнера.

Настройка Docker-контейнера YDB

YDB в Docker-контейнере настраивается через дополнительные переменные окружения, меняющие поведение YDB по умолчанию. Старт Docker-контейнера YDB осуществляется с помощью команды `docker run`, в которой можно указать переменные окружения, используя ключ `-e` для каждой переменной окружения. Ниже приведён список поддерживаемых переменных окружения.

Переменные окружения

Имя	Тип	Значение по умолчанию	Описание
`YDB_GRPC_ENABLE_TLS`	`0`, `1`, `false` или `true`	`true`	Включает использование TLS для gRPC-соединений.
`YDB_GRPC_TLS_DATA_PATH`	`string`	`/ydb_certs`	Путь к директории с TLS-сертификатами для gRPC-соединений.
`MON_PORT`	`int`	`8765`	HTTP-порт YDB UI.
`GRPC_PORT`	`int`	`2136`	gRPC порт.
`IC_PORT`	`int`	`19001`	Порт интерконнекта.
`GRPC_TLS_PORT`	`int`	`2135`	gRPCS порт защищенного соединения.
`YDB_KAFKA_PROXY_PORT`	`int`		Порт для подключения по Kafka API. Если не заполнено, режим совместимости с Kafka отключен.
`YDB_ERASURE`	`string`	`None`	Режим работы распределённого хранилища, см. Топология кластера YDB.
`FQ_CONNECTOR_ENDPOINT`	`string`	`None`	Строка подключения коннектора к внешним источникам `fq-connector-go`.
`YDB_USE_IN_MEMORY_PDISKS`	`0` или `1`	`0`	Делает все данные волатильными, хранящимися только в оперативной памяти. В настоящее время сохранение данных путём её отключения поддерживается только на процессорах x86_64 или эмулирующих их виртуальных машинах.
`YDB_PDISK_SIZE`	`string`	`64GB`	Размер файла для хранения данных в директории `/ydb_data`, используемый запущенным контейнером. Может быть указан числом в байтах или с суффиксом: `KB`, `MB` или `GB` (например, `64GB`).
`YDB_DEFAULT_LOG_LEVEL`	`string`	`NOTICE`	Задаёт уровень логирования по умолчанию. Доступные значения уровней: `CRIT`, `ERROR`, `WARN`, `NOTICE`, `INFO`.
`YDB_ADDITIONAL_LOG_CONFIGS`	`string`	`None`	Задаёт дополнительные уровни логирования в формате: `компонент:значение уровня`. Если необходимо указать несколько компонентов, их следует вводить через запятую.
`YDB_FEATURE_FLAGS`	`string`	`None`	Список фича-флагов YDB, через запятую.
`YDB_ENABLE_COLUMN_TABLES`	`0` или `1`	`0`	Включает поддержку колоночных таблиц.
`YDB_PREINITSCRIPTS_DIR`	`string`	`/preinit.d`	Путь к директории pre-init scripts
`YDB_INITSCRIPTS_DIR`	`string`	`/init.d`	Путь к директории init scripts
`YDB_CHANNEL_BUFFER_SIZE`	`string`	`8388608`	Размер канала передачи данных между компонентами обработки запроса. Значение по умолчанию (в байтах) в промышленном окружении равно `8 МиБ`, в тестовом — `256 КиБ`.

Пользовательские скрипты инициализации

Docker-контейнер YDB поддерживает пользовательские скрипты инициализации, позволяющие автоматизировать настройку базы данных.

Директории для скриптов

Существуют две директории для размещения пользовательских скриптов:

Директория	Описание	Использование
`/preinit.d`	Скрипты в этой директории выполняются при каждом запуске контейнера, **до** запуска сервера YDB.	Выполняются при каждом запуске контейнера до запуска сервера. Полезны для установки переменных окружения, настройки логирования или других подготовительных задач, которые могут выполняться при каждом запуске контейнера.
`/init.d`	Скрипты в этой директории выполняются только **один раз** после успешного запуска сервера YDB. Создаётся файл-маркер для предотвращения повторного выполнения при перезапуске контейнера.	Выполняются только один раз после успешного запуска сервера. Например, для создания структуры базы данных (таблиц, индексов) и вставки начальных данных.

Поддерживаемые типы файлов

Скрипты предварительной инициализации (`/preinit.d`)

Расширение	Описание
`.sh`	Shell-скрипты. Если скрипт исполняемый, он запускается напрямую. В противном случае он подключается (sourced), что позволяет модифицировать переменные окружения для последующих скриптов.

Скрипты инициализации (`/init.d`)

Расширение	Описание
`.sh`	Shell-скрипты. Если скрипт исполняемый, он запускается напрямую. В противном случае он подключается (sourced).
`.sql`	SQL-файлы. Содержимое выполняется через интерфейс командной строки YDB YQL.
`.sql.gz`	Сжатые SQL-файлы в формате gzip. Содержимое распаковывается и выполняется.

Порядок выполнения

Скрипты выполняются в алфавитном порядке внутри каждой директории. Используйте числовые префиксы для управления последовательностью выполнения:

```
/init.d/
├── 01-create-tables.sh
├── 02-create-indexes.sql
└── 03-insert-data.sql.gz
```

Переменные окружения

Вы можете настроить директории скриптов с помощью переменных окружения:

Переменная	Значение по умолчанию	Описание
`YDB_PREINITSCRIPTS_DIR`	`/preinit.d`	Путь к директории скриптов предварительной инициализации
`YDB_INITSCRIPTS_DIR`	`/init.d`	Путь к директории скриптов инициализации

Обработка ошибок

Если какой-либо скрипт завершается с ошибкой (возвращает ненулевой код), контейнер прекращает выполнение и завершается с ошибкой. Это гарантирует немедленное обнаружение ошибок инициализации и предотвращает запуск контейнера с незавершённой настройкой.

Примеры

Использование shell-скриптов

Создайте файл `/init.d/01-setup.sh` для настройки базы данных и заполните следующим содержимым:

```
#!/bin/bash
# /init.d/01-setup.sh
echo "Настройка базы данных..."
/ydb -e grpc://localhost:2136 -d /local --no-discovery sql -s "CREATE TABLE test (id Uint64, PRIMARY KEY (i
d));"
```

Примонтируйте директорию init при запуске контейнера:

```
docker run -d \
    --name ydb-local \
    -v $(pwd)/init.d:/init.d \
    ydbplatform/local-ydb:latest
```

При запуске контейнера автоматически создастся таблица — `test` с использованием YDB CLI. Скрипт выполняется только один раз после успешного старта сервера.

Использование SQL-файлов

Создайте файл `/init.d/01-create-tables.sql` с содержимым:

```
-- /init.d/01-create-tables.sql
CREATE TABLE users (
    id Uint64,
    name Utf8,
    email Utf8,
    PRIMARY KEY (id)
);

CREATE TABLE orders (
    id Uint64,
    user_id Uint64,
    amount Double,
    PRIMARY KEY (id)
);
```

Примонтируйте директорию init при запуске контейнера:

```
docker run -d \
    --name ydb-local \
    -v $(pwd)/init.d:/init.d \
    ydbplatform/local-ydb:latest
```

При запуске контейнера автоматически создадутся две таблицы — `users` и `orders` — с помощью SQL-скрипта. Скрипт выполняется только один раз после успешного старта сервера.

Восстановление из резервной копии с помощью init-скриптов

Создайте файл `/init.d/01-restore-backup.sh` с содержимым:

```
# /init.d/01-restore-backup.sh
#!/bin/bash
if [ -d "/backup" ] && [ -n "$(ls -A /backup)" ]; then
    /ydb -e grpc://localhost:2136 -d /local --no-discovery tools restore -p . -i /backup
fi
```

Смонтируйте директории init, а также том с вашей резервной копией при запуске контейнера:

```
docker run -d \
    --name ydb-local \
    -v $(pwd)/init.d:/init.d \
    -v $(pwd)/backup:/backup \
    ydbplatform/local-ydb:latest
```

При запуске контейнера произойдет восстановление из резервной копии `/backup` в базу данных `/local`. Скрипт выполняется только один раз после успешного старта сервера.

Использование скриптов предварительной инициализации

Создайте файл `/preinit.d/01-set-env.sh` с содержимым:

```
# /preinit.d/01-set-env.sh
# Этот скрипт будет подключён (sourced), поэтому экспортированные переменные будут доступны
export YDB_DEFAULT_LOG_LEVEL=INFO
```

Примонтируйте директорию preinit при запуске контейнера:

```
docker run -d \
    --name ydb-local \
    -v $(pwd)/preinit.d:/preinit.d \
    ydbplatform/local-ydb:latest
```

При запуске контейнера экспортируется переменная окружения `YDB_DEFAULT_LOG_LEVEL` со значением `INFO`. Скрипт выполняется при каждом запуске контейнера.

Остановка Docker

Для остановки YDB в Docker необходимо выполнить следующую команду:

```
docker stop ydb-local
```

Если при запуске был указан флаг `--rm`, то контейнер будет удалён после остановки.

Удаление Docker-контейнера YDB

Для удаления Docker-контейнера, содержащего YDB, необходимо выполнить следующую команду:

```
docker kill ydb-local
```

Для очистки файловую систему удалите рабочую директорию с помощью команды `rm -rf ~/ydbd`, при этом все данные внутри локального кластера YDB будут потеряны.

Рецепты по работе с YDB

Этот раздел документации YDB содержит готовые рецепты для различных аспектов взаимодействия с YDB. Они сгруппированы по следующим категориям:

- Рецепты кода с использованием YDB SDK и фреймворков
- YDB CLI рецепты
- YQL рецепты
- Рецепты работы с трансфером данных из топика в таблицу
- Рецепты резервного копирования
- Рецепты работы с потоковыми запросами

Рецепты резервного копирования

Практические руководства по резервному копированию и восстановлению данных в YDB.

Содержание:

- Рецепты для коллекций резервных копий
- Рецепты для резервного копирования системных таблеток
- Резервное копирование и восстановление через NFS

См. также

- Резервное копирование — концепции и обзор механизмов
- Резервное копирование и восстановление — руководство для администраторов

Рецепты для коллекций резервных копий

Пошаговые руководства для типовых сценариев работы с коллекциями резервных копий.

Начало работы

- Создание первой коллекции резервных копий — создание коллекции, резервное копирование и мониторинг операций

Настройка окружений

- Настройка резервного копирования для разных сред — конфигурация для разработки и продуктовых сред
- Стратегия резервного копирования для микросервисов — организация резервных копий по границам сервисов

Внешнее хранилище

- Экспорт резервных копий во внешнее хранилище — экспорт в S3 или файловую систему для аварийного восстановления
- Импорт и восстановление резервных копий — восстановление из внешнего хранилища

Обслуживание

- Обслуживание и очистка резервных копий — управление жизненным циклом и хранилищем
- Проверка и тестирование резервных копий — проверка целостности резервных копий

См. также

- Коллекции резервных копий
- Резервное копирование и восстановление
- CREATE BACKUP COLLECTION
- BACKUP
- RESTORE
- DROP BACKUP COLLECTION

Создание первой коллекции резервных копий

Это руководство описывает создание коллекции резервных копий, выполнение первых резервных копий и мониторинг операций.

Создание коллекции резервных копий

Коллекция резервных копий — это объект схемы, хранящийся в схеме базы данных. Вы создаёте и управляете коллекциями с помощью SQL-операторов, а просматриваете их с помощью команд навигации по схеме (например, `ydb scheme ls`), поскольку они отображаются как каталоги в структуре базы данных.

Создайте коллекцию, включающую таблицы, для которых нужно обеспечить согласованное резервное копирование:

```
-- Создание коллекции для связанных таблиц
CREATE BACKUP COLLECTION production_backups
    ( TABLE orders
    , TABLE products
    , TABLE customers
    )
WITH ( STORAGE = 'cluster', INCREMENTAL_BACKUP_ENABLED = 'true' );
```

Создание резервных копий

После создания коллекции выполните первоначальную полную резервную копию, затем используйте инкрементальные копии для последующих операций:

```
-- Создание первоначальной полной резервной копии
BACKUP production_backups;

-- Внесите изменения в данные...
-- Операции INSERT, UPDATE или DELETE над таблицами в коллекции

-- Позже создайте инкрементальную резервную копию, чтобы захватить изменения
BACKUP production_backups INCREMENTAL;
```

> **Примечание**
>
> Операции резервного копирования выполняются асинхронно и не идемпотентны — каждая команда `BACKUP` создаёт новую резервную копию. Перед повтором после таймаута проверьте статус операции с помощью `ydb operation list incbackup`.

Мониторинг операций резервного копирования

Отслеживайте прогресс резервного копирования и просматривайте структуру резервных копий:

```
# Проверка статуса операций резервного копирования
ydb operation list incbackup

# Получение подробностей для конкретной операции
ydb operation get <operation-id>

# Просмотр коллекций резервных копий
ydb scheme ls .backups/collections/

# Список резервных копий в коллекции
ydb scheme ls .backups/collections/production_backups/
```

Следующие шаги

- Настройка резервного копирования для разных сред
- Экспорт резервных копий во внешнее хранилище
- Проверка и тестирование резервных копий

Настройка резервного копирования для разных сред

Настройка различных стратегий резервного копирования для сред разработки и продуктовых сред.

Среда разработки

Используйте упрощённые конфигурации с меньшим количеством таблиц для тестирования:

```
-- Создание коллекции с меньшим количеством таблиц для тестирования
CREATE BACKUP COLLECTION dev_test_backups
    ( TABLE users
    , TABLE test_data
    )
WITH ( STORAGE = 'cluster', INCREMENTAL_BACKUP_ENABLED = 'true' );

-- Ежедневные полные резервные копии в среде разработки
BACKUP dev_test_backups;
```

Производственная среда

Создавайте коллекции с более частыми инкрементальными резервными копиями:

```
-- Создание коллекции для продуктовой среды
CREATE BACKUP COLLECTION prod_daily_backups
    ( TABLE orders
    , TABLE products
    , TABLE customers
    , TABLE inventory
    , TABLE transactions
    )
WITH ( STORAGE = 'cluster', INCREMENTAL_BACKUP_ENABLED = 'true' );

-- Еженедельная полная резервная копия
BACKUP prod_daily_backups;

-- Ежедневные инкрементальные резервные копии
BACKUP prod_daily_backups INCREMENTAL;
```

Рекомендуемое расписание резервного копирования

Среда	Полная копия	Инкрементальная копия
Разработка	Ежедневно	Нет
Staging	Еженедельно	Ежедневно
Продуктовая среда	Еженедельно	Ежедневно или ежечасно

Следующие шаги

- Стратегия резервного копирования для микросервисов
- Экспорт резервных копий во внешнее хранилище

Стратегия резервного копирования для микросервисов

Организация коллекций резервных копий по границам сервисов для независимых операций резервного копирования и восстановления.

Коллекции для отдельных сервисов

Создайте отдельные коллекции резервных копий для каждого микросервиса:

```
-- Коллекция резервных копий пользовательского сервиса
CREATE BACKUP COLLECTION user_service_backups
    ( TABLE profiles
    , TABLE preferences
    , TABLE sessions
    )
WITH ( STORAGE = 'cluster', INCREMENTAL_BACKUP_ENABLED = 'true' );

-- Коллекция резервных копий сервиса заказов
CREATE BACKUP COLLECTION order_service_backups
    ( TABLE orders
    , TABLE order_items
    , TABLE payments
    )
WITH ( STORAGE = 'cluster', INCREMENTAL_BACKUP_ENABLED = 'true' );

-- Коллекция резервных копий сервиса инвентаря
CREATE BACKUP COLLECTION inventory_service_backups
    ( TABLE products
    , TABLE stock_levels
    , TABLE warehouses
    )
WITH ( STORAGE = 'cluster', INCREMENTAL_BACKUP_ENABLED = 'true' );
```

Независимое резервное копирование

Выполняйте резервное копирование каждого сервиса по собственному расписанию:

```
-- Создание резервных копий для каждого сервиса независимо
BACKUP user_service_backups;
BACKUP order_service_backups;
BACKUP inventory_service_backups;

-- Позже создание инкрементальных резервных копий
BACKUP user_service_backups INCREMENTAL;
BACKUP order_service_backups INCREMENTAL;
BACKUP inventory_service_backups INCREMENTAL;
```

Преимущества коллекций для отдельных сервисов

- **Независимое восстановление**: восстановление отдельных сервисов без влияния на другие
- **Гибкое расписание**: разная частота резервного копирования для сервисов в зависимости от критичности
- **Уменьшение зоны поражения**: проблемы с одной коллекцией не влияют на другие
- **Владение командами**: каждая команда может управлять своей стратегией резервного копирования

Следующие шаги

- Экспорт резервных копий во внешнее хранилище
- Импорт и восстановление резервных копий

Экспорт резервных копий во внешнее хранилище

Экспорт коллекций резервных копий в S3-совместимое хранилище или файловую систему для аварийного восстановления.

Экспорт в S3

Для больших резервных копий или аварийного восстановления экспортируйте в S3-совместимое хранилище:

```
# Экспорт коллекции резервных копий в S3
ydb export s3 \
  --s3-endpoint storage.yandexcloud.net \
  --bucket my-backup-bucket \
  --item src=.backups/collections/production_backups,dst=backups/production_backups

# Экспорт конкретной резервной копии в S3
ydb export s3 \
  --s3-endpoint storage.yandexcloud.net \
  --bucket my-backup-bucket \
  --item src=.backups/collections/production_backups/20250821141425Z_full,dst=backups/20250821141425Z_full
```

Экспорт в файловую систему

Для небольших резервных копий или локального тестирования экспортируйте в локальную файловую систему:

```
# Экспорт коллекции резервных копий
ydb tools dump -p .backups/collections/production_backups -o /backup/exports/production_backups_export

# Экспорт конкретной резервной копии из коллекции
ydb tools dump -p .backups/collections/production_backups/20250601120000Z_full -o /backup/exports/backup_2025
0601
```

Лучшие практики экспорта

- **Экспортируйте каждую резервную копию отдельно**: экспортируйте полные и инкрементальные копии по отдельности для сохранения целостности цепочки
- **Сохраняйте порядок цепочки**: при экспорте цепочки сначала экспортируйте полную копию, затем инкрементальные по порядку
- **Проверяйте экспорт**: убедитесь, что экспортированные данные полны, прежде чем удалять резервные копии из кластера
- **Планируйте регулярный экспорт**: автоматизируйте экспорт во внешнее хранилище для аварийного восстановления

Следующие шаги

- Импорт и восстановление резервных копий
- Обслуживание и очистка резервных копий

Импорт и восстановление резервных копий

Импорт резервных копий из внешнего хранилища и восстановление данных в базе данных.

Импорт из файловой системы

Импорт ранее экспортированных резервных копий:

```
# Импорт резервной копии в целевую базу данных
ydb tools restore -p .backups/collections/production_backups_restored -i /backup/exports/production_backups_e
xport

# Импорт конкретной резервной копии в коллекцию
ydb tools restore -p .backups/collections/emergency_restore -i /backup/exports/backup_20250601
```

Импорт из S3

```
ydb import s3 \
  --s3-endpoint storage.yandexcloud.net \
  --bucket my-backup-bucket \
  --item src=backups/production_backups,dst=.backups/collections/production_backups
```

Восстановление данных

После импорта резервных копий в кластер восстановите данные:

```
RESTORE production_backups;
```

Процесс аварийного восстановления

1. **Импорт полной резервной копии**: сначала импортируйте базовую полную резервную копию
2. **Импорт инкрементальных копий**: импортируйте каждую инкрементальную резервную копию по порядку
3. **Выполнение RESTORE**: запустите команду RESTORE для применения цепочки резервных копий

```
# Шаг 1: Импорт полной резервной копии
ydb tools restore -p .backups/collections/recovery/20250821141425Z_full -i /backup/full_20250821

# Шаг 2: Импорт инкрементальных копий по порядку
ydb tools restore -p .backups/collections/recovery/20250822070000Z_incremental -i /backup/inc_20250822

# Шаг 3: Восстановление
ydb yql -s "RESTORE recovery;"
```

Следующие шаги

- Обслуживание и очистка резервных копий
- Проверка и тестирование резервных копий

Обслуживание и очистка резервных копий

Управление жизненным циклом резервных копий и очистка старых резервных копий для контроля использования хранилища.

Определение цепочек резервных копий

Каталоги резервных копий именуются с временными метками и суффиксами (`_full` или `_incremental`). Инкрементальные резервные копии принадлежат последней полной резервной копии, которая предшествует им хронологически.

```
# Список резервных копий, отсортированный по времени
ydb scheme ls .backups/collections/production_backups/ | sort
```

Пример вывода:

```
20250601120000Z_full        # Цепочка 1: полная резервная копия
20250601130000Z_incremental # Цепочка 1: инкрементальная (принадлежит полной 20250601)
20250602070000Z_incremental # Цепочка 1: инкрементальная (принадлежит полной 20250601)
20250608120000Z_full        # Цепочка 2: новая полная резервная копия начинает новую цепочку
20250608130000Z_incremental # Цепочка 2: инкрементальная (принадлежит полной 20250608)
```

Когда вы создаёте новую полную резервную копию, она начинает новую цепочку. Все инкрементальные копии, созданные после этой полной копии (до следующей полной), принадлежат этой цепочке.

Ручная очистка

Удаляйте старые цепочки резервных копий, когда они больше не нужны:

```
# Удаление старых каталогов резервных копий
ydb scheme rmdir -r .backups/collections/production_backups/20250601120000Z_full/

# Всегда удаляйте полные цепи, никогда не удаляйте частичные цепи
# Пример: Удаление старой полной резервной копии и всех её инкрементальных копий
ydb scheme rmdir -r .backups/collections/production_backups/20250601120000Z_full/
ydb scheme rmdir -r .backups/collections/production_backups/20250601130000Z_incremental/
ydb scheme rmdir -r .backups/collections/production_backups/20250602070000Z_incremental/
```

> **Важно**
>
> Никогда не удаляйте отдельные резервные копии из середины цепочки. Удаление полной резервной копии делает все её инкрементальные копии невосстановимыми. Всегда удаляйте полные цепочки целиком.

Удаление коллекции

Удалите всю коллекцию резервных копий, когда она больше не нужна:

```
-- Удаление коллекции, когда она больше не нужна (удаляет коллекцию и все резервные копии)
DROP BACKUP COLLECTION old_collection_name;
```

Безопасный процесс очистки

1. Создайте новую полную резервную копию
2. Убедитесь, что новая резервная копия завершена
3. Экспортируйте старые цепочки резервных копий во внешнее хранилище при необходимости
4. Удалите старые цепочки резервных копий (полную копию + все её инкрементальные копии вместе)

Советы по управлению хранилищем

- **Отслеживайте рост хранилища**: регулярно контролируйте использование хранилища резервных копий
- **Установите политики хранения**: определите, как долго хранить цепочки резервных копий
- **Периодически начинайте новые цепочки**: создавайте новые полные резервные копии еженедельно или раз в две недели для ограничения длины цепочки
- **Экспортируйте перед удалением**: всегда экспортируйте во внешнее хранилище перед удалением резервных копий из кластера

Следующие шаги

- Проверка и тестирование резервных копий
- Экспорт резервных копий во внешнее хранилище

Проверка и тестирование резервных копий

Проверка целостности резервных копий и тестирование процедур восстановления.

Проверка завершения резервного копирования

Убедитесь, что операции резервного копирования завершились успешно:

```
# Список операций резервного копирования и проверка статуса
ydb operation list incbackup
```

Ищите следующие значения статуса в выводе:

Статус	Значение
`SUCCESS`	Операция завершена успешно
`PROGRESS`	Операция ещё выполняется
`CANCELLED`	Операция была отменена (в настоящее время неприменимо — отмена пока не поддерживается)
`ERROR`	Операция завершилась с ошибкой

```
# Получение подробной информации о конкретной операции
ydb operation get <operation-id>

# Проверка существования каталога резервной копии
ydb scheme ls .backups/collections/production_backups/ | tail -1
```

Если операция показывает статус `ERROR`, проверьте детали операции для получения сообщения об ошибке и убедитесь, что все таблицы в коллекции доступны.

Проверка экспорта во внешнее хранилище

Регулярно проверяйте, что резервные копии можно экспортировать во внешнее хранилище:

```
# Экспорт в S3 и проверка
ydb export s3 \
  --s3-endpoint storage.yandexcloud.net \
  --bucket my-backup-bucket \
  --item src=.backups/collections/production_backups/20250821141425Z_full,dst=backups/test_export

# Проверка успешного завершения экспорта
ydb operation list export/s3

# Проверка наличия экспортированных данных в S3 (с помощью вашего S3-клиента)
# aws s3 ls s3://my-backup-bucket/backups/test_export/
```

Тестирование восстановления резервных копий

Регулярно проверяйте, что резервные копии можно восстановить:

```
# Экспорт недавней резервной копии для тестирования
ydb tools dump -p .backups/collections/production_backups/20250601120000Z_full -o /tmp/test_restore

# Тестовое восстановление во временное место (в тестовой среде)
ydb tools restore -p /Root/test_restore_verification -i /tmp/test_restore
```

Чек-лист проверки

- [ ] Операции резервного копирования завершаются без ошибок
- [ ] Каталоги резервных копий создаются с ожидаемыми временными метками
- [ ] Экспортированные резервные копии можно успешно импортировать
- [ ] Восстановленные данные соответствуют ожидаемому содержимому
- [ ] Целостность инкрементальной цепочки сохраняется

Рекомендуемое расписание тестирования

Тип теста	Частота	Описание
Статус операций	Ежедневно	Проверка завершения запланированных резервных копий
Проверка экспорта	Еженедельно	Проверка экспорта во внешнее хранилище
Полный тест восстановления	Ежемесячно	Полное восстановление в тестовую среду
Учения по аварийному восстановлению	Ежеквартально	Полная симуляция аварийного восстановления

Настройка мониторинга и оповещений

YDB не предоставляет встроенного мониторинга или оповещений для резервного копирования. Реализуйте внешний мониторинг для обнаружения сбоев резервного копирования:

Пример скрипта мониторинга

```
#!/bin/bash
# Проверка неуспешных операций резервного копирования
FAILED=$(ydb operation list incbackup --format json | jq '[.operations[] | select(.status == "ERROR")] | length')

if [ "$FAILED" -gt 0 ]; then
  echo "ALERT: $FAILED операция(й) резервного копирования завершилась с ошибкой" | mail -s "YDB Backup Alert" admin@example.com
fi

# Проверка, что резервная копия была создана за последние 24 часа
LATEST=$(ydb scheme ls .backups/collections/production_backups/ | sort | tail -1)
# Добавьте свою логику для разбора временной метки и сравнения с текущим временем
```

Рекомендации по мониторингу

- Планируйте запуск скриптов мониторинга через cron после ожидаемого времени резервного копирования
- Проверяйте как статус операции (`ERROR`), так и создание новых резервных копий
- Интегрируйте с вашей существующей системой оповещений (email, Slack, PagerDuty и т.д.)
- Отслеживайте рост хранилища резервных копий, чтобы избежать нехватки места

Следующие шаги

- Создание первой коллекции резервных копий
- Обслуживание и очистка резервных копий

Рецепты для резервного копирования системных таблеток

Пошаговые руководства по включению резервного копирования системных таблеток и их восстановлению.

Включение и восстановление

- Включение и настройка резервного копирования — настройка `system_tablet_backup_config` и подготовка хранилища;
- Восстановление системных таблеток — перевод в Recovery-режим и восстановление из локальной резервной копии.

См. также

- Резервное копирование системных таблеток — концепции и принцип работы
- Резервное копирование и восстановление — обзор способов резервного копирования
- system_tablet_backup_config — параметры конфигурации

Включение и настройка резервного копирования системных таблеток

Примечание

Концептуальную информацию о механизме см. в разделе Резервное копирование системных таблеток.

Важно

В качестве хранилища резервных копий поддерживаются только локальные файловые системы. Сетевые файловые системы, такие как NFS, не поддерживаются и могут привести к повреждению резервных копий или сбоям в работе таблеток.

По умолчанию резервное копирование системных таблеток **выключено**, так как требует настройки хранилища. Узлы YDB обычно не используют локальную файловую систему, поэтому включение требует осознанного решения администратора.

Для включения добавьте секцию `system_tablet_backup_config` в конфигурацию кластера:

```
system_tablet_backup_config:
    filesystem:
        path: "/path/to/backup/directory"
```

Параметр `path` — абсолютный путь к директории на локальной файловой системе хоста для хранения резервных копий.

Важно

Директория для резервных копий должна быть доступна для записи процессом YDB на каждом хосте, где работают системные таблетки. Убедитесь, что на диске каждого хоста достаточно свободного места для хранения нескольких резервных копий всех системных таблеток. Для крупных рабочих сред, состоящих из десятков узлов хранения, рекомендуется как минимум 5 ГБ.

После изменения конфигурации перезапустите узлы кластера, на которых могут работать системные таблетки. Резервное копирование начнётся автоматически.

Важно

По умолчанию системные таблетки могут работать на любых статических узлах кластера, в том числе на тех, диски которых входят в статическую группу. Резервные копии хранятся локально на хостах, где работают системные таблетки. Это означает, что при потере хостов статической группы могут быть потеряны и резервные копии, которые могут потребоваться для восстановления статической группы.

Рекомендуется настроить секцию `bootstrap_config` в конфигурации кластера так, чтобы системные таблетки запускались на узлах, не входящих в статическую группу. В этом случае резервные копии будут храниться отдельно от данных статической группы.

Восстановление системных таблеток

Примечание

Концептуальную информацию о механизме см. в разделе Резервное копирование системных таблеток.

Важно

Восстановление системных таблеток — критическая операция, в результате которой могут быть потеряны данные. Выполняйте её только при наличии чёткого понимания проблемы и после консультации с командой эксплуатации. Перед началом убедитесь, что вы ознакомились со всеми шагами.

Шаг 1. Переведите таблетку в Recovery-режим

Таблетку, которую требуется восстановить, необходимо перевести в Recovery-режим.

Важно

Если восстановление выполняется после полной потери статической группы, все системные таблетки должны быть переведены в Recovery-режим одновременно с пересозданием статической группы на новых хостах. Если пересоздать статическую группу, не переведя таблетки в Recovery-режим, они автоматически запустятся поверх пустой статической группы, что приведёт к неправильной работе кластера.

1. Определите идентификатор системной таблетки, которую требуется восстановить. Идентификатор таблетки можно найти в разделе Tablets в YDB UI.
2. Сохраните текущую конфигурацию кластера в файл `config.yaml`.
   - При использовании конфигурации V1, необходимо сохранить статическую конфигурацию.
   - При использовании конфигурации V2, воспользуйтесь инструкцией.
3. Определите список хостов, на которых может работать восстанавливаемая системная таблетка по конфигурации кластера. Этот список будет использован в дальнейших шагах для перезапуска узлов и обновления конфигурации.

   Для этого удобно воспользоваться скриптом, который принимает на вход идентификатор таблетки и файл конфигурации кластера, в котором хранится список узлов, где могут работать системные таблетки, и отображение из узлов в хосты. Скрипт объединяет эту информацию, и в результате выдает список хостов, на которых может работать переданная системная таблетка, в пригодном для pssh формате.

   Создайте изолированное окружение с PyYAML:

   ```
   python3 -m venv ./recovery && ./recovery/bin/pip install pyyaml
   ```

   Создайте файл скрипта:

   ```
   cat > find-tablet-hosts.py <<'EOF'
   #!/usr/bin/env python3
   import argparse
   import yaml


   def find_nodes(cfg, tid):
       bootstrap = (cfg.get("bootstrap_config") or {}).get("tablet") or []
       for tablet in bootstrap:
           if str(tablet.get("info", {}).get("tablet_id")) == tid:
               return tablet.get("node") or []

       for tablets_by_type in (cfg.get("system_tablets") or {}).values():
           for tablet in tablets_by_type or []:
               if str(tablet.get("info", {}).get("tablet_id")) == tid:
                   return tablet.get("node") or []

       return []


   parser = argparse.ArgumentParser()
   parser.add_argument("--config-path", required=True, help="Path to cluster configuration file")
   parser.add_argument("--tablet-id", required=True, help="System tablet id")
   args = parser.parse_args()

   with open(args.config_path) as f:
       cfg = yaml.safe_load(f) or {}

   cfg = cfg.get("config", cfg)

   hosts_by_id = {h["node_id"]: h["host"] for h in cfg.get("hosts") or []}

   node_ids = find_nodes(cfg, args.tablet_id)
   if not node_ids:
       node_ids = list(hosts_by_id)

   for node_id in sorted(set(node_ids)):
       host = hosts_by_id.get(node_id)
       if host is not None:
           print(host)
   EOF
   ```

Запустите скрипт, подставив путь к своей конфигурации и идентификатор таблетки:

Пример:

```
./recovery/bin/python find-tablet-hosts.py --config-path config.yaml --tablet-id <tablet-id> > hosts.txt
```

4. Измените сохраненную конфигурацию, добавив `boot_type: RECOVERY` в секцию конфигурации запуска восстанавливаемой таблетки и сократите список узлов, где может запускаться восстанавливаемая таблетка (массив `node`), до одного узла.
   Пример для таблетки `Hive` с идентификатором `72057594037968897`:
   - При использовании `bootstrap_config`:

     ```
     bootstrap_config:
         tablet:
         - type: FLAT_HIVE
             node:
             - 1
             info:
                 tablet_id: '72057594037968897'
                 channels:
                 - channel: 0
                 history:
                 - from_generation: 0
                     group_id: 0
                 channel_erasure_name: mirror-3-dc
                 - channel: 1
                 history:
                 - from_generation: 0
                     group_id: 0
                 channel_erasure_name: mirror-3-dc
                 - channel: 2
                 history:
                 - from_generation: 0
                     group_id: 0
                 channel_erasure_name: mirror-3-dc
             boot_type: RECOVERY
     ```

   - При использовании `system_tablets`:

     ```
     flat_hive:
     - info:
         tablet_id: 72057594037968897
       node:
       - 9
       boot_type: RECOVERY
     ```

5. Обновите конфигурацию в кластере.
   - При использовании конфигурации V1 необходимо обновить статическую конфигурацию на всех хостах, полученных на шаге 3, с помощью команды:

     Скопируйте обновлённую конфигурацию во временный файл на каждом хосте:

     ```
     pscp -h hosts.txt config.yaml /tmp/config.yaml.new
     ```

     Сделайте резервную копию текущей конфигурации, замените её новой версией и защитите от случайных изменений:

     ```
     pssh -h hosts.txt 'sudo cp /opt/ydb/cfg/config.yaml /opt/ydb/cfg/config.yaml.bak && sudo mv /tmp/con
     fig.yaml.new /opt/ydb/cfg/config.yaml && sudo chattr +i /opt/ydb/cfg/config.yaml'
     ```

   - При использовании конфигурации V2, воспользуйтесь инструкцией.
6. Перезапустите все узлы, на которых может работать восстанавливаемая таблетка. Если какой-либо узел недоступен и не может быть перезапущен, изолируйте его от кластера по сети — например, с помощью firewall.

   Пример:

   ```
   pssh -h hosts.txt 'sudo systemctl restart ydbd-storage; sleep 10'
   ```

   Важно

   Убедитесь, что все узлы, на которых может работать восстанавливаемая таблетка, перезапущены с обновлённой конфигурацией или изолированы от кластера по сети до начала восстановления. В процессе восстановления старые данные стираются и заменяются данными из резервной копии. Если в этот момент таблетка запустится в обычном режиме на узле со старой конфигурацией, она может начать работу с частично восстановленными данными.

7. Убедитесь, что:
   - С таблеткой нет проблем в HealthCheck.
   - Таблетка не перезапускается.
   - В App таблетки в YDB UI доступна форма восстановления.

Шаг 2. Найдите файлы резервных копий

1. На каждом хосте, полученном на шаге 3, проверьте наличие резервных копий. Путь к резервным копиям определяется параметром `path` в секции конфигурации `system_tablet_backup_config`.

   Имя каждой резервной копии содержит ключевую информацию: `backup_<timestamp>_g<generation>_s<step>`, где:

   - `timestamp` — время создания резервной копии;
   - `generation` — поколение таблетки, увеличивается при каждом перезапуске таблетки;
   - `step` — шаг таблетки в рамках поколения, увеличивается при каждом изменении состояния таблетки.

2. Выберите наиболее актуальную резервную копию, пригодную для восстановления.

   Выберите резервную копию с **максимальным поколением**, а при равном поколении — с **максимальным шагом**.

   Убедитесь, что резервная копия полностью записана. Выбранная резервная копия должна содержать директорию `snapshot`, а **не** `snapshot.tmp`. Наличие `snapshot.tmp` означает, что запись снапшота не была завершена и копия непригодна для восстановления. В этом случае выберите предыдущую по актуальности копию.

   ```
   ls /path/to/backup/directory/hive/72057594037968897/backup_20251007T193502_g214_s1222/snapshot/
   ```

   ```
   manifest.json
   schema.json
   Tablet.json
   TabletFollowerGroup.json
   ...
   ```

   Проверьте, что файлы резервной копии содержат данные:

   ```
   ls -lh /path/to/backup/directory/hive/72057594037968897/backup_20251007T193502_g214_s1222/snapshot
   ```

   ```
   total 128K
   -rw-r--r-- 1 ydb disk  591 May 27 11:34 manifest.json
   -rw-r--r-- 1 ydb disk  12K May 27 11:34 schema.json
   -rw-r--r-- 1 ydb disk  47K May 27 11:34 Tablet.json
   -rw-r--r-- 1 ydb disk  41K May 27 11:34 TabletFollowerGroup.json
   ...
   ```

   Посчитайте чексумму от файла `changelog.json` и сверьте ее с чексуммой, записанной в `changelog.json.sha256`:

   ```
   sha256sum /path/to/backup/directory/hive/72057594037968897/backup_20251007T193502_g214_s1222/changelog.j
   son
   ```

   ```
   ea4bdc2f7afaf7b6d35adbf13b5360e4e4a19046f742effdf1b4f0bb9c449185
   ```

   ```
   cat /path/to/backup/directory/hive/72057594037968897/backup_20251007T193502_g214_s1222/changelog.json.sh
   a256
   ```

   ```
   ea4bdc2f7afaf7b6d35adbf13b5360e4e4a19046f742effdf1b4f0bb9c449185
   ```

   В случае, если чексуммы отличаются, то возможны две ситуации:

   - Последняя запись в `changelog.json` не была полностью записана. В этом случае, чексумма, хранящаяся в `changelog.json.sha256` находится в одной из записей в `changelog.json` в поле `prev_sha256`. Для восстановления необходимо отредактировать файлы: удалить/дополнить не полностью записанные записи в `changelog.json` и обновить чексумму в `changelog.json.sha256`.
   - Произошло повреждение данных, восстановление невозможно.

Шаг 3. Перенесите файлы резервной копии

1. Определите, на каком хосте запущена таблетка в Recovery-режиме. Для этого откройте YDB UI и найдите хост, на котором работает таблетка.
2. Если файлы резервной копии находятся на другом хосте, скопируйте их на хост с таблеткой в Recovery-режиме с помощью `scp`, `rsync` или любого другого доступного инструмента:

   Скопируйте бекап из директории с бекапами в свою домашнюю директорию:

   ```
   cp -R /path/to/backup/directory/hive/72057594037968897/backup_20251007T193502_g214_s1222 ~/backup_202510
   07T193502_g214_s1222
   ```

   Скопируйте бекап на целевой хост:

   ```
   scp -r ~/backup_20251007T193502_g214_s1222 target-host:~/backup_20251007T193502_g214_s1222
   ```

   В случае, если операция копирования между хостами напрямую недоступна, то необходимо копировать через промежуточный хост:

   ```
   scp -r backup-host:~/backup_20251007T193502_g214_s1222 backup_20251007T193502_g214_s1222
   ```

```
scp -r backup_20251007T193502_g214_s1222 target-host:~/backup_20251007T193502_g214_s1222
```

После копирования бекапа на целевой хост, предоставьте доступ к нему процессу ydb:

```
chown -R ydb ~/backup_20260527104029Z_g1911_s
```

Шаг 4. Выполните восстановление

1. Откройте App восстанавливаемой таблетки в YDB UI.
2. В форме восстановления укажите полный путь до директории с файлами резервной копии, например:

```
/path/to/backup/directory/hive/72057594037968897/backup_20251007T193502_g214_s1222
```

3. При необходимости установите флаги:
   - **Dry Run** — выполняет пробное восстановление без внесения изменений в хранилище. Позволяет убедиться в корректности резервной копии. После завершения пробного восстановления необходимо перезапустить таблетку.
   - **Skip Checksum Validation** — пропускает проверку контрольных сумм файлов резервной копии.

> **Важно**
>
> Всегда выполняйте Dry Run перед восстановлением, чтобы убедиться в корректности резервной копии.

> **Важно**
>
> Пропускайте проверку контрольных сумм только в случае ручного редактирования файлов резервной копии. В остальных случаях рекомендуется оставлять проверку включённой для обеспечения целостности восстанавливаемых данных.

4. Нажмите кнопку **Start Restore**. Перед началом восстановления система запросит подтверждение, так как операция перезаписывает существующие данные таблетки. Подтвердите действие для запуска. После начала восстановления форма становится недоступной — повторный запуск невозможен до перезапуска таблетки.

> **Примечание**
>
> Восстановление также можно запустить с помощью `curl`, отправив POST-запрос на страницу App таблетки с параметром `restoreBackup`:
>
> ```
> curl -X POST "http://<host>:<mon_port>/tablets/app?TabletID=72057594037968897&restoreBackup=/tablet/hive/72057594037968897/backup_20251007T193502_g214_s1222"
> ```
>
> Где `<host>` — адрес узла кластера, `<mon_port>` — порт мониторинга этого узла.

5. Отслеживайте прогресс восстановления. Страница **не обновляется автоматически** — для получения актуального статуса обновляйте страницу вручную. Продолжительность восстановления зависит от объёма резервной копии.

> **Примечание**
>
> Восстановление выполняется на стороне сервера и не зависит от браузера. Вы можете закрыть страницу или браузер — это **не прервёт** процесс восстановления. При повторном открытии страницы отобразится актуальный статус.

   Под формой отображается текущий статус операции. Возможные статусы:

   - `Restoring from '<путь>'` — восстановление выполняется, данные из резервной копии считываются и записываются в хранилище. Дополнительно отображается **прогресс-бар** с процентом выполнения операции.
   - `Restore from '<путь>' completed successfully` — восстановление завершено успешно. Можно переходить к следующему шагу.
   - `Restore from '<путь>' completed, but changelog is not fully restored` — основные данные таблетки восстановлены, но хвост журнала изменений в резервной копии повреждён, и часть последних изменений потеряна. Изучите, какие данные были потеряны, и при необходимости восстановите их вручную, либо переходите к следующему шагу.
   - `Restore from '<путь>' failed: <описание ошибки>` — восстановление завершилось с ошибкой. Изучите описание ошибки и при необходимости повторите попытку.

   Для принудительного прерывания восстановления **перезапустите таблетку**.
6. Дождитесь успешного завершения восстановления. Если форма восстановления снова стала доступна (кнопка **Start Restore** активна, статус сброшен), это означает, что таблетка была перезапущена и восстановление прервано. В этом случае начните восстановление заново с пункта 2.

Шаг 5. Верните таблетку к нормальному режиму работы

После успешного восстановления:

1. Верните конфигурацию в исходное состояние, удалив `boot_type: RECOVERY` из секции конфигурации запуска восстанавливаемой таблетки и восстановив изначальный список узлов, где может запускаться восстанавливаемая таблетка.
   - При использовании конфигурации V1 необходимо обновить статическую конфигурацию на всех хостах, полученных на шаге 3, с помощью команды:

     Снимите защиту от случайных изменений и замените новую конфигурацию исходной версией:

```
pssh -h hosts.txt 'sudo chattr -i /opt/ydb/cfg/config.yaml && sudo mv /opt/ydb/cfg/config.yaml.bak /opt/ydb/cfg/config.yaml'
```

- При использовании конфигурации V2, воспользуйтесь инструкцией.

2. Перезапустите все узлы, на которых может работать восстанавливаемая таблетка. Если какие-либо узлы были изолированы от кластера по сети на предыдущих шагах, снимите сетевую изоляцию.

   Пример:

   ```
   pssh -h hosts.txt 'sudo systemctl restart ydbd-storage; sleep 10'
   ```

3. Убедитесь, что:
   - С таблеткой нет проблем в HealthCheck.
   - Таблетка не перезапускается.
   - В App таблетки в YDB UI отсутствует форма восстановления.
4. Поочерёдно перезапустите все узлы кластера для синхронизации состояния внутренних кешей в памяти с состоянием таблетки. После перезапуска каждого узла дождитесь его возвращения в рабочее состояние и убедитесь в отсутствии проблем в HealthCheck; только после этого переходите к следующему узлу.

Резервное копирование и восстановление через NFS

Рецепт описывает подготовку NFS для серверных команд `export nfs` и `import nfs`: требования к кластеру, права доступа, пошаговую настройку NFS-сервера и клиентов, feature flags, производительность и диагностику. Синтаксис команд см. в справочнике: выгрузка в NFS, загрузка из NFS.

Предварительные требования

Операции `export nfs` / `import nfs` выполняются **серверными процессами** YDB, а не CLI-клиентом. Это означает, что файловая система должна быть доступна с каждого узла кластера:

- NFS-директория должна быть смонтирована на **всех узлах** кластера YDB по одному и тому же абсолютному пути.
- Процесс YDB должен иметь права на **запись** при экспорте и на **чтение** при импорте в эту директорию.
- Путь `--fs-path` должен быть абсолютным. Относительные пути, пути содержащие `..` или `.`, а также пути с нулевыми байтами будут отклонены сервером.
- Функциональность должна быть включена с помощью feature flag `enable_fs_backups`.

> **Важно**
>
> Данная функциональность совместима с протоколом сетевой файловой системы NFSv4.

Права доступа

Файловые операции выполняются серверными процессами YDB, а не CLI-клиентом. Поэтому права на NFS-директорию должны быть предоставлены пользователю ОС, от имени которого запущен процесс `ydbd`.

POSIX-права:

- Для **экспорта** (запись файлов): `rwx` на директорию (чтение для обхода, запись для создания файлов и поддиректорий, выполнение для навигации) и `rw` на файлы.
- Для **импорта** (чтение файлов): `rx` на директорию (чтение для обхода, выполнение для навигации) и `r` на файлы.

NFSv4 ACL:

NFSv4 использует расширенную модель прав вместо POSIX `rwx`. Одна и та же буква в NFSv4 ACE имеет разное значение для файлов и директорий:

Буква	Для файлов	Для директорий
`r`	`read-data` — чтение содержимого	`list-directory` — просмотр содержимого
`w`	`write-data` — запись данных	`create-file` — создание файлов
`a`	`append-data` — дополнение данных	`create-subdirectory` — создание поддиректорий
`x`	`execute` — выполнение	`change-directory` — навигация
`D`	—	`delete-child` — удаление вложенных объектов
`t`	`read-attributes`	`read-attributes`

Права на NFS-директорию для **экспорта** (создание файлов и поддиректорий):

- На саму директорию: `w` (create-file), `a` (create-subdirectory), `x` (change-directory), `t` (read-attributes), `D` (delete-child).
- На создаваемые файлы: `w` (write-data), `a` (append-data), `t` (read-attributes).

Права на NFS-директорию для **импорта** (чтение файлов):

- На саму директорию: `r` (list-directory), `x` (change-directory), `t` (read-attributes).
- На файлы: `r` (read-data), `t` (read-attributes).

Поскольку файлы создаются процессом `ydbd` в ходе экспорта, права на них нельзя задать заранее. В NFSv4 для этого используется **наследование ACE** — флаги `f` (file-inherit) и `d` (directory-inherit) на ACE родительской директории. При создании нового файла или поддиректории ACE с этими флагами автоматически копируются на новый объект.

Пошаговая настройка NFS-сервера и клиентов

Ниже приведены инструкции по настройке NFS-сервера и клиентских узлов YDB для двух вариантов управления правами: стандартные POSIX-права и расширенные NFSv4 ACL.

Вариант 1: POSIX-права

Настройка NFS-сервера

1. Установите пакет NFS-сервера:

```
sudo apt install nfs-kernel-server
```

2. Создайте директорию для экспорта и назначьте владельца:

```
sudo mkdir -p /mnt/ydb-backup
sudo chown ydb:ydb /mnt/ydb-backup
sudo chmod 755 /mnt/ydb-backup
```

Опции команды `chmod`:

- `7` (rwx) — владелец (`ydb`) может читать, писать и переходить в директорию.

- `5` (r-x) — группа и остальные могут читать и переходить, но не писать.

3. Добавьте директорию в `/etc/exports`:

```
/mnt/ydb-backup ydb-node-*.example.com(rw,sync,root_squash)
```

Опции экспорта:

- `rw` — разрешить чтение и запись.
- `sync` — синхронная запись (данные записываются на диск до подтверждения клиенту).
- `root_squash` — преобразовать `root` на клиенте в анонимного пользователя.

4. Примените изменения и запустите NFS-сервер:

```
sudo exportfs -arv
sudo systemctl enable --now nfs-server
```

Опции `exportfs`:

- `-a` — экспортировать все директории из `/etc/exports`.
- `-r` — переэкспортировать все директории (синхронизация с `/etc/exports`).
- `-v` — подробный вывод.

Настройка клиентов (узлов YDB)

1. Установите пакет NFS-клиента:

```
sudo apt install nfs-common
```

2. Создайте точку монтирования:

```
sudo mkdir -p /mnt/ydb-backup
```

3. Смонтируйте NFS-директорию:

```
sudo mount -t nfs4 -o rw nfs-server.example.com:/mnt/ydb-backup /mnt/ydb-backup
```

Опции монтирования:

- `-t nfs4` — тип файловой системы NFSv4.
- `-o rw` — монтировать с правом чтения и записи.

В данном примере хост сервер не принадлежит кластеру YDB, при этом резервное копирование и восстановление будут корректно работать в случае, если один из хостов кластера назначен NFS-сервером.

4. Для автоматического монтирования при загрузке добавьте строку в `/etc/fstab`:

```
nfs-server.example.com:/mnt/ydb-backup /mnt/ydb-backup nfs4 rw,hard,timeo=600,retrans=2,_netdev 0 0
```

Опции `fstab`:

- `rw` — чтение и запись.
- `hard` — при недоступности сервера повторять запросы бесконечно (не терять данные).
- `timeo=600` — таймаут запроса в десятых долях секунды (60 секунд).
- `retrans=2` — количество повторных попыток перед сообщением об ошибке.
- `_netdev` — монтировать только после инициализации сети.

Проверка работоспособности

На сервере (или любом клиенте) создайте тестовый файл от имени пользователя `ydb`:

```
sudo -u ydb touch /mnt/ydb-backup/test-file
```

На всех клиентских узлах проверьте, что файл виден:

```
sudo -u ydb ls /mnt/ydb-backup/test-file
```

После проверки удалите тестовый файл:

```
sudo -u ydb rm /mnt/ydb-backup/test-file
```

Вариант 2: NFSv4 ACL

NFSv4 ACL позволяют задать более гранулярные права доступа, а также настроить наследование прав для вновь создаваемых файлов и директорий.

Настройка NFS-сервера

1. Установите пакет NFS-сервера и утилиты для работы с NFSv4 ACL:

```
sudo apt install nfs-kernel-server nfs4-acl-tools
```

2. Создайте директорию для экспорта:

```
sudo mkdir -p /mnt/ydb-backup-server
```

3. Добавьте директорию в `/etc/exports`:

```
/mnt/ydb-backup-server ydb-node-*.example.com(rw,sync,no_subtree_check,root_squash)
```

4. Примените изменения:

```
sudo exportfs -arv
sudo systemctl enable --now nfs-server
```

Настройка клиентов (узлов YDB)

В отличие от POSIX-прав, которые задаются на NFS-сервере, NFSv4 ACL настраиваются на каждом клиенте после монтирования директории. Если NFS-сервер одновременно является узлом кластера YDB, на нём необходимо выполнить все описанные ниже шаги.

1. Установите пакеты NFS-клиента и утилиты для работы с NFSv4 ACL:

```
sudo apt install nfs-common nfs4-acl-tools
```

2. Создайте точку монтирования:

```
sudo mkdir -p /mnt/ydb-backup
```

3. Смонтируйте NFS-директорию по протоколу NFSv4:

```
sudo mount -t nfs4 -o rw nfs-server.example.com:/mnt/ydb-backup /mnt/ydb-backup
```

Опции:

- `-t nfs4` — явно указать протокол NFSv4 (необходимо для корректной работы ACL).
- `-o rw` — монтировать с правом чтения и записи.

4. Для автоматического монтирования добавьте строку в `/etc/fstab`:

```
nfs-server.example.com:/mnt/ydb-backup-server /mnt/ydb-backup nfs4 rw,hard,timeo=600,retrans=2,_netde
v 0 0
```

5. Определите UID пользователя `ydb`:

```
id -u ydb
```

Пример вывода: `1871`

6. Настройте NFSv4 ACL на директории:

```
sudo nfs4_setfacl -s A::<uid>:rwaxtD,A:fi:OWNER@:rwat,A:di:OWNER@:rwaxtD /mnt/ydb-backup
```

Структура ACE (`A:flags:who:permissions`):

- `A` — тип ACE: Allow (разрешить).
- Флаги наследования:
  - `f` (file-inherit) — наследовать на создаваемые файлы.
  - `d` (directory-inherit) — наследовать на создаваемые поддиректории.
  - `i` (inherit-only) — применять только к потомкам, не к самой директории.
- `who` — кому предоставляются права:
  - `1871` — UID пользователя `ydb`.
  - `OWNER@` — владелец объекта.
- Разрешения:
  - `r` — read-data / list-directory.
  - `w` — write-data / create-file.
  - `a` — append-data / create-subdirectory.
  - `t` — read-attributes.
  - `D` - delete-child.

Данная команда устанавливает:

- Права для пользователя `ydb` на саму директорию.
- Наследуемые права для файлов и поддиректорий (`fi`, `di`), которые будут применяться только к потомкам.
- Права владельца (`OWNER@`) с наследованием на файлы и директории.

Проверка работоспособности

Проверьте установленные ACL:

```
nfs4_getfacl /mnt/ydb-backup
```

Создайте тестовый файл от имени пользователя `ydb`:

```
sudo -u ydb touch /mnt/ydb-backup/test-file
```

Проверьте, что файл унаследовал ACL:

```
nfs4_getfacl /mnt/ydb-backup/test-file
```

На всех клиентских узлах проверьте доступ:

```
sudo -u ydb ls /mnt/ydb-backup/test-file
```

После проверки удалите тестовый файл:

```
sudo -u ydb rm /mnt/ydb-backup/test-file
```

Feature flags

Для использования экспорта и импорта через NFS необходимо включить feature flag `enable_fs_backups` в конфигурации кластера. По умолчанию этот флаг **выключен**.

Конфигурация производительности

На производительность операций экспорта и импорта через NFS влияют два механизма: **брокер ресурсов** (управляет параллельностью задач сканирования и загрузки данных) и **IO-пул акторной системы** (обслуживает файловые операции записи и чтения).

Брокер ресурсов

Экспорт и импорт через NFS используют те же очереди брокера ресурсов, что и S3: `queue_backup` для экспорта и `queue_restore` для импорта. Увеличение лимитов ускоряет операции, но увеличивает нагрузку на CPU. Значения по умолчанию и способ настройки описаны в разделе resource_broker_config.

IO-пул акторной системы

Файловые операции (запись при экспорте, чтение при импорте) выполняются акторами в пуле `IO` акторной системы. По умолчанию этот пул содержит `1` поток. Поскольку операции с файловой системой являются блокирующими, при интенсивном экспорте или импорте IO-пул может стать узким местом.

Для увеличения количества потоков в IO-пуле измените параметр `threads` в конфигурации акторной системы:

```yaml
actor_system_config:
  executor:
  - name: IO
    threads: 3
    time_per_mailbox_micro_secs: 100
    type: IO
```

> **Совет**
>
> Рекомендации по выбору количества потоков IO-пула при использовании NFS-экспорта/импорта:
>
> - Начните с `2` потоков и наблюдайте за нагрузкой.
> - Учитывайте, что IO-пул обслуживает не только NFS-операции, но и другие блокирующие задачи (например, запись логов).
> - Мониторьте загрузку пула через YDB UI.

Логирование

Ход выполнения операций экспорта и импорта отражается в серверных логах YDB. Основные компоненты логирования:

- `EXPORT` — логирование процесса экспорта на SchemeShard: создание директорий, копирование таблиц, запуск задач резервного копирования, загрузка схемы.
- `IMPORT` — логирование процесса импорта на SchemeShard: чтение метаданных, создание таблиц, восстановление данных.
- `FS_WRAPPER` — логирование файловых операций на каждом узле. Этот компонент фиксирует ошибки ввода-вывода (недоступность файла, нехватка места, ошибки блокировки), что позволяет выявить проблемы с NFS-монтированием на конкретных узлах.

При диагностике проблем рекомендуется:

- Проверять статус операции через `ydb operation get`.
- Искать в серверных логах записи с компонентами `EXPORT` или `IMPORT` и идентификатором операции.
- При ошибках записи или чтения файлов искать в логах записи с компонентом `FS_WRAPPER` — они содержат путь к файлу и описание ошибки, что позволяет определить, на каком хосте и с каким файлом возникла проблема.
- Проверять, что NFS-директория доступна и имеет достаточно свободного места на **всех** узлах кластера.

Ограничения

- Экспорт и импорт через NFS поддерживают те же типы объектов, что и экспорт в S3. Подробный перечень поддерживаемых объектов приведён в документации команды экспорта.
- Импорт всегда создаёт объекты заново — нельзя импортировать в существующие таблицы.
- Параметр `--list` для перечисления объектов выгрузки в текущей версии доступен только для импорта из S3.
- Операция не поддерживается на Windows-платформе.

Рецепты кода с использованием YDB SDK и фреймворков

В данном разделе содержатся рецепты кода на разных языках программирования для решения различных задач, часто встречающихся на практике, с использованием YDB SDK.

Содержание:

- Инициализация драйвера
- Аутентификация
  - С помощью токена
  - Анонимная
  - Файл сервисного аккаунта
  - Сервис метаданных
  - С помощью переменных окружения
  - С помощью логина и пароля
- Балансировка
  - Равномерный случайный выбор
  - Предпочитать ближайший дата-центр
  - Предпочитать зону доступности
- Выполнение повторных запросов
- Установить размер пула сессий
- Вставка данных
- Пакетная вставка данных
- Установка режима выполнения транзакции
- Настройка времени жизни строк (TTL) таблиц
- Векторный поиск
- Координация
  - Распределённая блокировка
  - Обнаружение сервисов
  - Публикация конфигурации
  - Выбор лидера

Подключение средств диагностики YDB SDK — логирования, метрик и распределённой трассировки — описано в разделе Наблюдаемость в YDB SDK.

Смотрите также:

- YDB для разработчиков приложений
- Тестовое приложение
- Справка по YDB SDK

Инициализация драйвера

Для подключения к YDB требуется указать обязательные параметры (подробнее читайте в разделе Подключение к серверу YDB) и дополнительные, которые определяют поведение драйвера при работе.

Ниже приведены примеры кода подключения к YDB (создания драйвера) в разных YDB SDK.

C++

Native SDK

```
#include <ydb-cpp-sdk/client/driver/driver.h>

int main() {
  auto driverConfig = NYdb::TDriverConfig("grpc://localhost:2136/local");

  NYdb::TDriver driver(driverConfig);

  // ...

  driver.Stop();

  return 0;
}
```

userver

static config

```
ydb:
    databases:
        db:
            endpoint: grpc://localhost:2136
            database: /local
```

```
#include <userver/components/component_base.hpp>
#include <userver/components/minimal_server_component_list.hpp>
#include <userver/storages/secdist/component.hpp>
#include <userver/storages/secdist/provider_component.hpp>
#include <userver/utils/daemon_run.hpp>
#include <userver/ydb/component.hpp>
#include <userver/ydb/table.hpp>

class MyYdbWorker final : public components::ComponentBase {
public:
    static constexpr std::string_view kName = "my-ydb-worker";

    MyYdbWorker(const components::ComponentConfig& config, const components::ComponentContext& context)
        : components::ComponentBase(config, context),
          table_client_(context.FindComponent<ydb::YdbComponent>().GetTableClient("db"))
    {
        // ...
    }

private:
    std::shared_ptr<ydb::TableClient> table_client_;
};

int main(int argc, char* argv[]) {
    auto component_list = components::MinimalServerComponentList()
        .Append<components::DefaultSecdistProvider>()
        .Append<components::Secdist>()
        .Append<ydb::YdbComponent>()
        .Append<MyYdbWorker>();
    return utils::DaemonMain(argc, argv, component_list);
}
```

Go

Native SDK

```
package main

import (
  "context"

  "github.com/ydb-platform/ydb-go-sdk/v3"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()

  db, err := ydb.Open(ctx, "grpc://localhost:2136/local")
  if err != nil {
      panic(err)
```

```
  }
  defer db.Close()

  // ...
}
```

database/sql

С помощью коннектора (рекомендуемый способ)

```
package main

import (
  "context"
  "database/sql"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()

  nativeDriver, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
  )
  if err != nil {
    panic(err)
  }
  defer nativeDriver.Close(ctx)

  connector, err := ydb.Connector(nativeDriver)
  if err != nil {
    panic(err)
  }
  defer connector.Close()

  db := sql.OpenDB(connector)
  defer db.Close()

  // ...
}
```

С помощью строки подключения

Регистрация драйвера `database/sql` реализуется в момент импорта пакета конкретного драйвера через символ подчеркивания:

```
package main

import (
  "database/sql"

  _ "github.com/ydb-platform/ydb-go-sdk/v3"
)

func main() {
  db, err := sql.Open("ydb", "grpc://localhost:2136/local")
  if err != nil {
    panic(err)
  }
  defer db.Close()

  // ...
}
```

Java

Для подключения укажите строку подключения и при необходимости настройте аутентификацию. Запросы рекомендуется выполнять через `QueryClient` и `SessionRetryContext` (см. повторные попытки).

Native SDK

```
import tech.ydb.common.transaction.TxMode;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;

public class InitExample {
    public static void main(String[] args) {
        // Строка подключения: endpoint и путь к базе данных
        String connectionString = System.getenv().getOrDefault(
```

```
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString).build();
             QueryClient queryClient = QueryClient.newClient(transport).build()) {

            // Контекст ретраев для устойчивого выполнения запросов
            SessionRetryContext retryCtx = SessionRetryContext.create(queryClient).build();

            // Проверка подключения простым запросом
            QueryReader reader = retryCtx.supplyResult(session -> QueryReader.readFrom(
                    session.createQuery("SELECT 1 AS value", TxMode.NONE, Params.empty())
            )).join().getValue();

            ResultSetReader rs = reader.getResultSet(0);
            if (rs.next()) {
                System.out.println("Подключение успешно, SELECT 1 = " + rs.getColumn("value").getInt32());
            }
        }
    }
}
```

JDBC

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class JdbcInitExample {
    public static void main(String[] args) throws SQLException {
        // Драйвер tech.ydb.jdbc.YdbDriver должен быть в classpath для автозагрузки через DriverManager
        String url = System.getenv().getOrDefault(
                "YDB_JDBC_URL", "jdbc:ydb:grpc://localhost:2136/local");

        try (Connection connection = DriverManager.getConnection(url);
             Statement statement = connection.createStatement();
             ResultSet rs = statement.executeQuery("SELECT 1 AS value")) {

            if (rs.next()) {
                System.out.println("Подключение успешно, SELECT 1 = " + rs.getInt("value"));
            }
        }
    }
}
```

Для Spring Boot укажите URL и класс драйвера в `application.properties` или `application.yml` (`spring.datasource.url`, `spring.datasource.driver-class-name`).

Spring Boot, а также ORM и прочие инструменты вокруг JDBC (Hibernate, JOOQ, MyBatis и т.п.) инициализируют транспорт к YDB так же, как обычный JDBC: достаточно подключить зависимость с YDB JDBC-драйвером и задать URL подключения — отдельная настройка нативного `GrpcTransport` не требуется.

Python

Native SDK

```
import ydb

with ydb.Driver(connection_string="grpc://localhost:2136?database=/local") as driver:
  driver.wait(timeout=5)
  ...
```

Native SDK (Asyncio)

```
import ydb
import asyncio

async def ydb_init():
  async with ydb.aio.Driver(endpoint="grpc://localhost:2136", database="/local") as driver:
    await driver.wait()
    ...

asyncio.run(ydb_init())
```

C#

```
using Ydb.Sdk.Ado;

await using var dataSource = new YdbDataSource("Host=localhost;Port=2136;Database=/local");
await using var connection = await dataSource.OpenConnectionAsync();
// ...
```

JavaScript

```
import { Driver } from '@ydbjs/core'

const driver = new Driver('grpc://localhost:2136/local')
await driver.ready()
```

Rust

```
use ydb::{AccessTokenCredentials, ClientBuilder, YdbResult};

#[tokio::main]
async fn main() -> YdbResult<()> {
    let client = ClientBuilder::new_from_connection_string("grpc://localhost:2136/local")?
        .with_credentials(AccessTokenCredentials::from("..."))
        .client()?;
    client.wait().await?;
    Ok(())
}
```

PHP

```
<?php

use YdbPlatform\Ydb\Ydb;

$config = [
    // Database path
    'database'    => '/ru-central1/b1glxxxxxxxxxxxxxxx/etn0xxxxxxxxxxxxxxx',

    // Database endpoint
    'endpoint'    => 'ydb.serverless.yandexcloud.net:2135',

    // Auto discovery (dedicated server only)
    'discovery'   => false,

    // IAM config
    'iam_config'  => [
        // 'root_cert_file' => './CA.pem', // Root CA file (uncomment for dedicated server)
    ],

    'credentials' => new \YdbPlatform\Ydb\Auth\Implement\AccessTokenAuthentication('AAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAA') // use from reference/ydb-sdk/auth
];

$ydb = new Ydb($config);
```

Аутентификация

YDB поддерживает несколько способов аутентификации подключения к серверной стороне. Каждый из них, как правило, специфичен для конкретной пары окружений: где находится клиентское приложение (в доверенной зоне YDB или вне ее) и серверная часть YDB (докер-контейнер, Yandex.Cloud, data cloud, установка на отдельном кластере)

В данном разделе содержатся рецепты кода с настройкой аутентификации в разных YDB SDK. Общее описание принципов аутентификации в SDK можно прочитать в статье Аутентификация в SDK.

Содержание:

- С помощью токена
- Анонимная
- Файл сервисного аккаунта
- Сервис метаданных
- С помощью переменных окружения
- С помощью логина и пароля

Аутентификация при помощи токена

Аутентификация с токеном доступа передаёт заранее полученный IAM-токен в каждый запрос к YDB. Этот способ подходит, когда токен уже есть в приложении — например, получен через OAuth или утилиту `yc`. Типичные шаги: получить токен, создать провайдер аутентификации, открыть транспорт и выполнить запрос. Подробности о режимах аутентификации — в разделе Аутентификация; базовое подключение — в рецепте инициализации драйвера. Альтернативы: анонимная, через переменные окружения, метаданные, сервисный аккаунт, логин и пароль.

Ниже приведены примеры кода аутентификации при помощи токена в разных YDB SDK.

C++

Native SDK

```
#include <ydb-cpp-sdk/client/driver/driver.h>

NYdb::TDriver CreateDriverWithAccessToken(const std::string& accessToken) {
    auto config = NYdb::TDriverConfig("grpcs://localhost:2135/?database=/local")
        .SetAuthToken(accessToken);

    return NYdb::TDriver(config);
}
```

userver

secdist

```
{
  "ydb_settings": {
    "db": {
      "token": "<access token>"
    }
  }
}
```

Код инициализации `ydb::YdbComponent`, получения `ydb::TableClient` и запуска `components::MinimalServerComponentList` — как в примере из init.md.

Go

Native SDK

```
package main

import (
  "context"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithAccessTokenCredentials(os.Getenv("YDB_TOKEN")),
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
  ...
}
```

database/sql

Если используется коннектор для создания подключения к YDB

```
package main

import (
  "context"
  "database/sql"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  nativeDriver, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithAccessTokenCredentials(os.Getenv("YDB_TOKEN")),
  )
  if err != nil {
```

```
    panic(err)
  }
  defer nativeDriver.Close(ctx)
  connector, err := ydb.Connector(nativeDriver)
  if err != nil {
    panic(err)
  }
  db := sql.OpenDB(connector)
  defer db.Close()
  ...
}
```

Если используется строка подключения

```
package main

import (
  "context"
  "database/sql"
  "os"

  _ "github.com/ydb-platform/ydb-go-sdk/v3"
)

func main() {
  db, err := sql.Open("ydb", "grpcs://localhost:2135/local?token="+os.Getenv("YDB_TOKEN"))
  if err != nil {
    panic(err)
  }
  defer db.Close()
  ...
}
```

Java

Native SDK

```
import tech.ydb.common.transaction.TxMode;
import tech.ydb.auth.TokenAuthProvider;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;

public class AccessTokenExample {
    public static void main(String[] args) throws Exception {
        // Строка подключения из переменной окружения или локальный YDB по умолчанию
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        // IAM-токен из переменной окружения YDB_ACCESS_TOKEN_CREDENTIALS
        String token = System.getenv("YDB_ACCESS_TOKEN_CREDENTIALS");
        if (token == null || token.isEmpty()) {
            throw new IllegalStateException("Задайте переменную окружения YDB_ACCESS_TOKEN_CREDENTIALS");
        }

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString)
                .withAuthProvider(new TokenAuthProvider(token))
                .build();
             QueryClient queryClient = QueryClient.newClient(transport).build()) {

            SessionRetryContext retryCtx = SessionRetryContext.create(queryClient).build();
            QueryReader reader = retryCtx.supplyResult(
                    session -> QueryReader.readFrom(session.createQuery("SELECT 1", TxMode.NONE))
            ).join().getValue();

            // Проверка подключения: выводим результат SELECT 1
            ResultSetReader rs = reader.getResultSet(0);
            if (rs.next()) {
                System.out.println("SELECT 1 = " + rs.getColumn(0).getInt32());
            }
        }
    }
}
```

JDBC

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Properties;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class AccessTokenJdbcExample {
```

```
    public static void main(String[] args) throws SQLException {
        String jdbcUrl = System.getenv().getOrDefault(
                "YDB_JDBC_URL", "jdbc:ydb:grpc://localhost:2136/local");

        // Вариант 1: токен задаётся свойством token
        Properties propsWithToken = new Properties();
        String token = System.getenv("YDB_ACCESS_TOKEN_CREDENTIALS");
        if (token == null || token.isEmpty()) {
            throw new IllegalStateException("Задайте переменную окружения YDB_ACCESS_TOKEN_CREDENTIALS");
        }
        propsWithToken.setProperty("token", token);

        try (Connection connection = DriverManager.getConnection(jdbcUrl, propsWithToken);
             Statement statement = connection.createStatement();
             ResultSet rs = statement.executeQuery("SELECT 1")) {
            if (rs.next()) {
                System.out.println("SELECT 1 = " + rs.getInt(1));
            }
        }

        // Вариант 2: токен читается из файла — свойство tokenFile
        Properties propsWithTokenFile = new Properties();
        propsWithTokenFile.setProperty("tokenFile", System.getenv().getOrDefault("YDB_TOKEN_FILE", "~/.ydb_to
ken"));

        try (Connection connection = DriverManager.getConnection(jdbcUrl, propsWithTokenFile);
             Statement statement = connection.createStatement();
             ResultSet rs = statement.executeQuery("SELECT 1")) {
            if (rs.next()) {
                System.out.println("SELECT 1 = " + rs.getInt(1));
            }
        }
    }
}
```

В Spring Boot, ORM и прочих сторонних фреймворках вокруг JDBC используйте ту же JDBC-строку подключения и те же параметры аутентификации, что и выше (например, `spring.datasource.url` с query-параметрами или `spring.datasource.*` для токена и файла токена).

JavaScript

```
import { Driver } from "@ydbjs/core";
import { AccessTokenCredentialsProvider } from "@ydbjs/auth/access-token";

const driver = new Driver("grpc://localhost:2136/local", {
  credentialsProvider: new AccessTokenCredentialsProvider({
    token: "accessToken",
  }),
});

await driver.ready();
```

Python

Native SDK

```
import os
import ydb

with ydb.Driver(
    connection_string=os.environ["YDB_CONNECTION_STRING"],
    credentials=ydb.credentials.AccessTokenCredentials(os.environ["YDB_TOKEN"]),
) as driver:
    driver.wait(timeout=5)
    ...
```

Native SDK (Asyncio)

```
import os
import ydb
import asyncio

async def ydb_init():
    async with ydb.aio.Driver(
        endpoint=os.environ["YDB_ENDPOINT"],
        database=os.environ["YDB_DATABASE"],
        credentials=ydb.credentials.AccessTokenCredentials(os.environ["YDB_TOKEN"]),
    ) as driver:
        await driver.wait()
        ...

asyncio.run(ydb_init())
```

SQLAlchemy

```
import os
import sqlalchemy as sa
```

```
engine = sa.create_engine(
    "yql+ydb://localhost:2136/local",
    connect_args={
        "credentials": {"token": os.environ["YDB_TOKEN"]}
    }
)
with engine.connect() as connection:
    result = connection.execute(sa.text("SELECT 1"))
```

C#

```
using Ydb.Sdk.Ado;
using Ydb.Sdk.Auth;

var builder = new YdbConnectionStringBuilder("Host=localhost;Port=2136;Database=/local")
{
    CredentialsProvider = new TokenProvider("MY_VERY_SECURE_TOKEN")
};

await using var dataSource = new YdbDataSource(builder);
await using var connection = await dataSource.OpenConnectionAsync();
```

Для Entity Framework и linq2db не поддерживается.

Rust

```
use ydb::{AccessTokenCredentials, ClientBuilder, YdbResult};

let client = ClientBuilder::new_from_connection_string("grpc://localhost:2136?database=local")?
    .with_credentials(AccessTokenCredentials::from(std::env::var("YDB_TOKEN")?))
    .client()?;
```

PHP

```
<?php

use YdbPlatform\Ydb\Ydb;
use YdbPlatform\Ydb\Auth\Implement\AccessTokenAuthentication;

$config = [

    // Database path
    'database'    => '/local',

    // Database endpoint
    'endpoint'    => 'localhost:2136',

    // Auto discovery (dedicated server only)
    'discovery'   => false,

    // IAM config
    'iam_config'  => [
        'insecure' => true,
        // 'root_cert_file' => './CA.pem', // Root CA file (uncomment for dedicated server)
    ],

    'credentials' => new AccessTokenAuthentication('AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA')
];

$ydb = new Ydb($config);
```

Анонимная аутентификация

Анонимная аутентификация не передаёт учётные данные — YDB SDK подключается к базе без токена, логина или пароля. Это режим по умолчанию для локальной разработки и тестирования против Docker-контейнера с YDB. Типичные шаги: указать строку подключения, явно или неявно выбрать анонимный провайдер и выполнить запрос. Подробности — в разделе Аутентификация; базовое подключение — в рецепте инициализации драйвера. Другие способы: токен, переменные окружения, метаданные, сервисный аккаунт, логин и пароль.

Ниже приведены примеры кода анонимной аутентификации в разных YDB SDK.

C++

Native SDK

Анонимная аутентификация является аутентификацией по умолчанию.
Явным образом анонимную аутентификацию можно включить так:

```
#include <ydb-cpp-sdk/client/driver/driver.h>
#include <ydb-cpp-sdk/client/types/credentials/credentials.h>

NYdb::TDriver CreateDriverAnonymous() {
    auto config = NYdb::TDriverConfig("grpc://localhost:2136/local")
        .SetCredentialsProviderFactory(NYdb::CreateInsecureCredentialsProviderFactory());

    return NYdb::TDriver(config);
}
```

userver

Если в статическом конфиге не задавать `credentials-provider`, не указывать `databases.*.credentials` и не класть в secdist для этой базы `token`, `iam_jwt_params` и пару `user` / `password`, драйвер будет использовать анонимный режим по умолчанию.

Код инициализации `ydb::YdbComponent`, получения `ydb::TableClient` и запуска `components::MinimalServerComponentList` — как в примере из init.md.

Go

Native SDK

Анонимная аутентификация является аутентификацией по умолчанию.
Явным образом анонимную аутентификацию можно включить так:

```
package main

import (
  "context"

  "github.com/ydb-platform/ydb-go-sdk/v3"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithAnonymousCredentials(),
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
  ...
}
```

database/sql

Анонимная аутентификация является аутентификацией по умолчанию.
Явным образом анонимную аутентификацию можно включить так:

```
package main

import (
  "context"
  "database/sql"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  nativeDriver, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithAnonymousCredentials(),
  )
  if err != nil {
    panic(err)
  }
```

```
  defer nativeDriver.Close(ctx)
  connector, err := ydb.Connector(nativeDriver)
  if err != nil {
    panic(err)
  }
  db := sql.OpenDB(connector)
  defer db.Close()
  ...
}
```

Java

Native SDK

```
import tech.ydb.common.transaction.TxMode;
import tech.ydb.auth.NopAuthProvider;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;

public class AnonymousAuthExample {
    public static void main(String[] args) throws Exception {
        // Строка подключения из переменной окружения или локальный YDB по умолчанию
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString)
                // Явно указываем анонимную аутентификацию (режим по умолчанию)
                .withAuthProvider(NopAuthProvider.INSTANCE)
                .build();
             QueryClient queryClient = QueryClient.newClient(transport).build()) {

            SessionRetryContext retryCtx = SessionRetryContext.create(queryClient).build();
            QueryReader reader = retryCtx.supplyResult(
                    session -> QueryReader.readFrom(session.createQuery("SELECT 1", TxMode.NONE))
            ).join().getValue();

            // Проверка подключения: выводим результат SELECT 1
            ResultSetReader rs = reader.getResultSet(0);
            if (rs.next()) {
                System.out.println("SELECT 1 = " + rs.getColumn(0).getInt32());
            }
        }
    }
}
```

JDBC

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class AnonymousAuthJdbcExample {
    public static void main(String[] args) throws SQLException {
        String jdbcUrl = System.getenv().getOrDefault(
                "YDB_JDBC_URL", "jdbc:ydb:grpc://localhost:2136/local");

        // Подключение без дополнительных свойств — анонимная аутентификация
        try (Connection connection = DriverManager.getConnection(jdbcUrl);
             Statement statement = connection.createStatement();
             ResultSet rs = statement.executeQuery("SELECT 1")) {
            if (rs.next()) {
                System.out.println("SELECT 1 = " + rs.getInt(1));
            }
        }
    }
}
```

В Spring Boot, ORM и прочих сторонних фреймворках вокруг JDBC подключение задаётся той же JDBC-строкой подключения, что и выше (например, `spring.datasource.url`).

JavaScript

```
import { Driver } from "@ydbjs/core";
import { AnonymousCredentialsProvider } from "@ydbjs/auth/anonymous";

const driver = new Driver("grpc://localhost:2136/local", {
  credentialsProvider: new AnonymousCredentialsProvider(),
});

await driver.ready();
```

Python

Native SDK

```
import os
import ydb

with ydb.Driver(
    connection_string=os.environ["YDB_CONNECTION_STRING"],
    credentials=ydb.credentials.AnonymousCredentials(),
) as driver:
    driver.wait(timeout=5)
    ...
```

Native SDK (Asyncio)

```
import os
import ydb
import asyncio

async def ydb_init():
    async with ydb.aio.Driver(
        endpoint=os.environ["YDB_ENDPOINT"],
        database=os.environ["YDB_DATABASE"],
        credentials=ydb.credentials.AnonymousCredentials(),
    ) as driver:
        await driver.wait()
        ...

asyncio.run(ydb_init())
```

SQLAlchemy

```
import sqlalchemy as sa

engine = sa.create_engine("yql+ydb://localhost:2136/local")
with engine.connect() as connection:
    result = connection.execute(sa.text("SELECT 1"))
```

C#

```
using Ydb.Sdk.Ado;

await using var dataSource = new YdbDataSource("Host=localhost;Port=2136;Database=/local");
await using var connection = await dataSource.OpenConnectionAsync();
```

Для Entity Framework и linq2db используйте тот же connectionString.

Rust

```
use ydb::{AnonymousCredentials, ClientBuilder, YdbResult};

let client = ClientBuilder::new_from_connection_string("grpc://localhost:2136?database=local")?
    .with_credentials(AnonymousCredentials::new())
    .client()?;
```

PHP

```
<?php

use YdbPlatform\Ydb\Ydb;
use YdbPlatform\Ydb\Auth\Implement\AnonymousAuthentication;

$config = [

    // Database path
    'database'    => '/local',

    // Database endpoint
    'endpoint'    => 'localhost:2136',

    // Auto discovery (dedicated server only)
    'discovery'   => false,

    // IAM config
    'iam_config'  => [
        'insecure' => true,
        // 'root_cert_file' => './CA.pem', // Root CA file (uncomment for dedicated server)
    ],

    'credentials' => new AnonymousAuthentication()
];

$ydb = new Ydb($config);
```

Аутентификация при помощи файла сервисного аккаунта

Аутентификация с файлом сервисного аккаунта использует JSON-ключ для формирования JWT и получения IAM-токена. Этот способ подходит для серверных приложений вне облака, CI/CD и локальной разработки с доступом к YDB в Яндекс Облаке. Типичные шаги: сохранить ключ сервисного аккаунта в файл, передать путь к нему в провайдер аутентификации, открыть транспорт и выполнить запрос. Подробности — в разделе Аутентификация; базовое подключение — в рецепте инициализации драйвера. Другие способы: токен, анонимная, переменные окружения, метаданные, логин и пароль.

Ниже приведены примеры кода аутентификации при помощи файла сервисного аккаунта в разных YDB SDK.

C++

Native SDK

```
#include <ydb-cpp-sdk/client/driver/driver.h>
#include <ydb-cpp-sdk/client/iam/iam.h>

NYdb::TDriver CreateDriverWithServiceAccountKeyFile(
    const std::string& connectionString,
    const std::string& saKeyFilePath,
    const std::string& internalCA)
{
    auto config = NYdb::TDriverConfig(connectionString)
        .UseSecureConnection(internalCA)
        .SetCredentialsProviderFactory(NYdb::CreateIamJwtFileCredentialsProviderFactory({
            .JwtFilename = saKeyFilePath,
        }));

    return NYdb::TDriver(config);
}
```

userver

secdist

`<PEM>` - сертификаты Yandex Cloud.

```
{
  "ydb_settings": {
    "db": {
      "iam_jwt_params": {
        "id": "...",
        "service_account_id": "...",
        "private_key": "..."
      },
      "secure_connection_cert": "<PEM>"
    }
  }
}
```

Код инициализации `ydb::YdbComponent`, получения `ydb::TableClient` и запуска `components::MinimalServerComponentList` — как в примере из init.md.

Go

Native SDK

```
package main

import (
  "context"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  yc "github.com/ydb-platform/ydb-go-yc"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    yc.WithServiceAccountKeyFileCredentials(
      os.Getenv("YDB_SERVICE_ACCOUNT_KEY_FILE_CREDENTIALS"),
    ),
    yc.WithInternalCA(), // append Yandex Cloud certificates
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
  ...
}
```

database/sql

```
package main

import (
  "context"
  "database/sql"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  yc "github.com/ydb-platform/ydb-go-yc"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  nativeDriver, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    yc.WithServiceAccountKeyFileCredentials(
      os.Getenv("YDB_SERVICE_ACCOUNT_KEY_FILE_CREDENTIALS"),
    ),
    yc.WithInternalCA(), // append Yandex Cloud certificates
  )
  if err != nil {
    panic(err)
  }
  defer nativeDriver.Close(ctx)
  connector, err := ydb.Connector(nativeDriver)
  if err != nil {
    panic(err)
  }
  db := sql.OpenDB(connector)
  defer db.Close()
  ...
}
```

Java

Native SDK

```
import tech.ydb.auth.iam.CloudAuthHelper;
import tech.ydb.common.transaction.TxMode;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;

public class ServiceAccountAuthExample {
    public static void main(String[] args) throws Exception {
        // Строка подключения из переменной окружения или локальный YDB по умолчанию
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        // Путь к JSON-ключу сервисного аккаунта
        String saKeyFile = System.getenv("YDB_SERVICE_ACCOUNT_KEY_FILE_CREDENTIALS");
        if (saKeyFile == null || saKeyFile.isEmpty()) {
            throw new IllegalStateException(
                    "Задайте переменную окружения YDB_SERVICE_ACCOUNT_KEY_FILE_CREDENTIALS");
        }

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString)
                .withAuthProvider(CloudAuthHelper.getServiceAccountFileAuthProvider(saKeyFile))
                .build();
             QueryClient queryClient = QueryClient.newClient(transport).build()) {

            SessionRetryContext retryCtx = SessionRetryContext.create(queryClient).build();
            QueryReader reader = retryCtx.supplyResult(
                    session -> QueryReader.readFrom(session.createQuery("SELECT 1", TxMode.NONE))
            ).join().getValue();

            // Проверка подключения: выводим результат SELECT 1
            ResultSetReader rs = reader.getResultSet(0);
            if (rs.next()) {
                System.out.println("SELECT 1 = " + rs.getColumn(0).getInt32());
            }
        }
    }
}
```

JDBC

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Properties;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```

```java
public class ServiceAccountAuthJdbcExample {
    public static void main(String[] args) throws SQLException {
        String jdbcUrl = System.getenv().getOrDefault(
                "YDB_JDBC_URL", "jdbc:ydb:grpc://localhost:2136/local");

        String saKeyFile = System.getenv("YDB_SERVICE_ACCOUNT_KEY_FILE_CREDENTIALS");
        if (saKeyFile == null || saKeyFile.isEmpty()) {
            throw new IllegalStateException(
                    "Задайте переменную окружения YDB_SERVICE_ACCOUNT_KEY_FILE_CREDENTIALS");
        }

        Properties props = new Properties();
        props.setProperty("saKeyFile", saKeyFile);

        try (Connection connection = DriverManager.getConnection(jdbcUrl, props);
             Statement statement = connection.createStatement();
             ResultSet rs = statement.executeQuery("SELECT 1")) {
            if (rs.next()) {
                System.out.println("SELECT 1 = " + rs.getInt(1));
            }
        }
    }
}
```

Опцию `saKeyFile` также можно указать прямо в JDBC URL: `jdbc:ydb:grpc://localhost:2136/local?saKeyFile=~/keys/sa_key.json`.

В Spring Boot, ORM и прочих сторонних фреймворках вокруг JDBC укажите ту же JDBC-строку подключения и параметр `saKeyFile` (в URL или в свойствах `DataSource`), что и в примере выше.

JavaScript

Загрузка данных сервисного аккаунта из файла:

```javascript
import { Driver } from "@ydbjs/core";
import { ServiceAccountCredentialsProvider } from "@ydbjs/auth-yandex-cloud";

const driver = new Driver("grpc://localhost:2136/local", {
  credentialsProvider: ServiceAccountCredentialsProvider.fromFile("./authorized_key.json"),
});

await driver.ready();
```

Загрузка данных сервисного аккаунта из стороннего источника (например, из хранилища секретов):

```javascript
import { Driver } from "@ydbjs/core";
import { ServiceAccountCredentialsProvider } from "@ydbjs/auth-yandex-cloud";

const driver = new Driver("grpc://localhost:2136/local", {
  credentialsProvider: new ServiceAccountCredentialsProvider({
    id: "serviceAccountId",
    keyId: "accessKeyId",
    privateKey: "-----BEGIN PRIVATE KEY-----\n...",
  }),
});

await driver.ready();
```

Python

Native SDK

```python
import os
import ydb
import ydb.iam

with ydb.Driver(
    connection_string=os.environ["YDB_CONNECTION_STRING"],
    # service account key should be in the local file,
    # and SA_KEY_FILE environment variable should point to it
    credentials=ydb.iam.ServiceAccountCredentials.from_file(os.environ["SA_KEY_FILE"]),
) as driver:
    driver.wait(timeout=5)
    ...
```

Native SDK (Asyncio)

```python
import os
import asyncio
import ydb
import ydb.iam

async def ydb_init():
    async with ydb.aio.Driver(
        endpoint=os.environ["YDB_ENDPOINT"],
        database=os.environ["YDB_DATABASE"],
        # service account key should be in the local file,
```

```
        # and SA_KEY_FILE environment variable should point to it
        credentials=ydb.iam.ServiceAccountCredentials.from_file(os.environ["SA_KEY_FILE"]),
    ) as driver:
        await driver.wait()
        ...

asyncio.run(ydb_init())
```

SQLAlchemy

```
import os
import sqlalchemy as sa
import ydb.iam

engine = sa.create_engine(
    "yql+ydb://localhost:2136/local",
    connect_args={
        "credentials": ydb.iam.ServiceAccountCredentials.from_file(
            os.environ["YDB_SERVICE_ACCOUNT_KEY_FILE_CREDENTIALS"]
        )
    }
)
with engine.connect() as connection:
    result = connection.execute(sa.text("SELECT 1"))
```

C#

```
using Ydb.Sdk.Ado;

await using var dataSource = new YdbDataSource(
    "Host=ydb.serverless.yandexcloud.net;Port=2135;Database=/ru-central1/<folder-id>/<database-id>;ServiceAcc
ountKeyFilePath=path/to/sa_file.json");
await using var connection = await dataSource.OpenConnectionAsync();
```

Для Entity Framework и linq2db используйте тот же connectionString.

Rust

```
use ydb::{ClientBuilder, ServiceAccountCredentials, YdbResult};

let client = ClientBuilder::new_from_connection_string(std::env::var("YDB_CONNECTION_STRING")?)?
    .with_credentials(ServiceAccountCredentials::from_env()?)
    .client()?;
```

PHP

```
<?php

use YdbPlatform\Ydb\Ydb;
use YdbPlatform\Ydb\Auth\JwtWithJsonAuthentication;

$config = [
    'database'    => '/ru-central1/b1glxxxxxxxxxxxxxxx/etn0xxxxxxxxxxxxxxx',
    'endpoint'    => 'ydb.serverless.yandexcloud.net:2135',
    'discovery'   => false,
    'iam_config'  => [
        'temp_dir'       => './tmp', // Temp directory
        // 'root_cert_file' => './CA.pem', // Root CA file (uncomment for dedicated server)ы
    ],

    'credentials' => new JwtWithJsonAuthentication('./jwtjson.json')
];

$ydb = new Ydb($config);
```

или

```
<?php

use YdbPlatform\Ydb\Ydb;
use YdbPlatform\Ydb\Auth\JwtWithPrivateKeyAuthentication;

$config = [
    'database'    => '/ru-central1/b1glxxxxxxxxxxxxxxx/etn0xxxxxxxxxxxxxxx',
    'endpoint'    => 'ydb.serverless.yandexcloud.net:2135',
    'discovery'   => false,
    'iam_config'  => [
        'temp_dir'           => './tmp', // Temp directory
        // 'root_cert_file' => './CA.pem', // Root CA file (uncomment for dedicated server)

    ],

    'credentials' => new JwtWithPrivateKeyAuthentication(
        "ajexxxxxxxxx","ajeyyyyyyyyy",'./private.key')

];
```

```
$ydb = new Ydb($config);
```

Аутентификация при помощи сервиса метаданных

Аутентификация через сервис метаданных получает IAM-токен из HTTP-эндпоинта виртуальной машины или облачной функции Яндекс Облака. Этот способ предназначен для приложений, которые работают внутри Яндекс Облака и не хранят ключи в коде. Типичные шаги: создать провайдер метаданных, открыть транспорт с защищённым соединением к YDB в облаке и выполнить запрос. Подробности — в разделе Аутентификация; базовое подключение — в рецепте инициализации драйвера. Другие способы: токен, анонимная, переменные окружения, сервисный аккаунт, логин и пароль.

Ниже приведены примеры кода аутентификации при помощи сервиса метаданных в разных YDB SDK.

C++

Native SDK

```
#include <ydb-cpp-sdk/client/driver/driver.h>
#include <ydb-cpp-sdk/client/iam/iam.h>

NYdb::TDriver CreateDriverWithMetadataCredentials(
    const std::string& connectionString,
    const std::string& internalCA)
{
    auto config = NYdb::TDriverConfig(connectionString)
        .UseSecureConnection(internalCA)
        .SetCredentialsProviderFactory(NYdb::CreateIamCredentialsProviderFactory());

    return NYdb::TDriver(config);
}
```

userver

Готовой настройки для сервиса метаданных в `ydb::YdbComponent` нет — нужен свой `ydb::CredentialsProviderComponent` с `NYdb::CreateIamCredentialsProviderFactory()`.

static config

```
ydb:
    credentials-provider: ydb-metadata-credentials
    databases:
        db:
            endpoint: grpcs://localhost:2135
            database: /local
            credentials: {}
```

secdist

`<PEM>` - сертификаты Yandex Cloud.

```
{
  "ydb_settings": {
    "db": {
      "secure_connection_cert": "<PEM>"
    }
  }
}
```

```
#include <userver/components/component_base.hpp>
#include <userver/components/minimal_server_component_list.hpp>
#include <userver/storages/secdist/component.hpp>
#include <userver/storages/secdist/provider_component.hpp>
#include <userver/utils/daemon_run.hpp>
#include <userver/ydb/component.hpp>
#include <userver/ydb/credentials.hpp>
#include <userver/ydb/table.hpp>

#include <ydb-cpp-sdk/client/iam/iam.h>

class YdbMetadataCredentials final : public ydb::CredentialsProviderComponent {
public:
    static constexpr std::string_view kName = "ydb-metadata-credentials";

    using ydb::CredentialsProviderComponent::CredentialsProviderComponent;

    std::shared_ptr<NYdb::ICredentialsProviderFactory> CreateCredentialsProviderFactory(
        const yaml_config::YamlConfig&) const override
    {
        return NYdb::CreateIamCredentialsProviderFactory();
    }
};

class MyYdbWorker final : public components::ComponentBase {
public:
    static constexpr std::string_view kName = "my-ydb-worker";

    MyYdbWorker(const components::ComponentConfig& config, const components::ComponentContext& context)
        : components::ComponentBase(config, context),
          table_client_(context.FindComponent<ydb::YdbComponent>().GetTableClient("db"))
```

```
    {
        // ...
    }

private:
    std::shared_ptr<ydb::TableClient> table_client_;
};

int main(int argc, char* argv[]) {
    auto component_list = components::MinimalServerComponentList()
        .Append<components::DefaultSecdistProvider>()
        .Append<components::Secdist>()
        .Append<YdbMetadataCredentials>()
        .Append<ydb::YdbComponent>()
        .Append<MyYdbWorker>();
    return utils::DaemonMain(argc, argv, component_list);
}
```

Go

Native SDK

```
package main

import (
  "context"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  yc "github.com/ydb-platform/ydb-go-yc-metadata"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    yc.WithCredentials(),
    yc.WithInternalCA(), // append Yandex Cloud certificates
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
  ...
}
```

database/sql

```
package main

import (
  "context"
  "database/sql"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  yc "github.com/ydb-platform/ydb-go-yc-metadata"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  nativeDriver, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    yc.WithCredentials(),
    yc.WithInternalCA(), // append Yandex Cloud certificates
  )
  if err != nil {
    panic(err)
  }
  defer nativeDriver.Close(ctx)
  connector, err := ydb.Connector(nativeDriver)
  if err != nil {
    panic(err)
  }
  db := sql.OpenDB(connector)
  defer db.Close()
  ...
}
```

Java

Native SDK

```
import tech.ydb.auth.iam.CloudAuthHelper;
import tech.ydb.common.transaction.TxMode;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
```

```java
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;

public class MetadataAuthExample {
    public static void main(String[] args) throws Exception {
        // Строка подключения из переменной окружения или локальный YDB по умолчанию
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        // Токен запрашивается у сервиса метаданных виртуальной машины или функции в Яндекс Облаке
        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString)
                .withAuthProvider(CloudAuthHelper.getMetadataAuthProvider())
                .build();
             QueryClient queryClient = QueryClient.newClient(transport).build()) {

            SessionRetryContext retryCtx = SessionRetryContext.create(queryClient).build();
            QueryReader reader = retryCtx.supplyResult(
                    session -> QueryReader.readFrom(session.createQuery("SELECT 1", TxMode.NONE))
            ).join().getValue();

            // Проверка подключения: выводим результат SELECT 1
            ResultSetReader rs = reader.getResultSet(0);
            if (rs.next()) {
                System.out.println("SELECT 1 = " + rs.getColumn(0).getInt32());
            }
        }
    }
}
```

JDBC

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Properties;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class MetadataAuthJdbcExample {
    public static void main(String[] args) throws SQLException {
        String jdbcUrl = System.getenv().getOrDefault(
                "YDB_JDBC_URL", "jdbc:ydb:grpc://localhost:2136/local");

        // Аутентификация через сервис метаданных Яндекс Облака
        Properties props = new Properties();
        props.setProperty("useMetadata", "true");

        try (Connection connection = DriverManager.getConnection(jdbcUrl, props);
             Statement statement = connection.createStatement();
             ResultSet rs = statement.executeQuery("SELECT 1")) {
            if (rs.next()) {
                System.out.println("SELECT 1 = " + rs.getInt(1));
            }
        }
    }
}
```

Опцию `useMetadata` также можно указать прямо в JDBC URL: `jdbc:ydb:grpc://localhost:2136/local?useMetadata=true`.

В Spring Boot, ORM и прочих сторонних фреймворках вокруг JDBC передайте те же JDBC URL и параметры (`useMetadata` в URL или в свойствах источника данных), что и в примере выше.

JavaScript

```javascript
import { Driver } from "@ydbjs/core";
import { MetadataCredentialsProvider } from "@ydbjs/auth/metadata";

const driver = new Driver("grpc://localhost:2136/local", {
  credentialsProvider: new MetadataCredentialsProvider(),
});

await driver.ready();
```

Python

Native SDK

```python
import os
import ydb
import ydb.iam

with ydb.Driver(
    connection_string=os.environ["YDB_CONNECTION_STRING"],
    credentials=ydb.iam.MetadataUrlCredentials(),
) as driver:
    driver.wait(timeout=5)
    ...
```

Native SDK (Asyncio)

```
import os
import ydb
import ydb.iam
import asyncio

async def ydb_init():
    async with ydb.aio.Driver(
        endpoint=os.environ["YDB_ENDPOINT"],
        database=os.environ["YDB_DATABASE"],
        credentials=ydb.iam.MetadataUrlCredentials(),
    ) as driver:
        await driver.wait()
        ...

asyncio.run(ydb_init())
```

SQLAlchemy

```
import sqlalchemy as sa
import ydb.iam

engine = sa.create_engine(
    "yql+ydb://localhost:2136/local",
    connect_args={
        "credentials": ydb.iam.MetadataUrlCredentials()
    }
)
with engine.connect() as connection:
    result = connection.execute(sa.text("SELECT 1"))
```

C#

```
using Ydb.Sdk.Ado;

await using var dataSource = new YdbDataSource(
    "Host=ydb.serverless.yandexcloud.net;Port=2135;Database=/ru-central1/<folder-id>/<database-id>;EnableMeta
dataCredentials=True");
await using var connection = await dataSource.OpenConnectionAsync();
```

Для Entity Framework и linq2db используйте тот же connectionString.

Rust

```
use ydb::{ClientBuilder, MetadataUrlCredentials, YdbResult};

let client = ClientBuilder::new_from_connection_string("grpc://localhost:2136?database=local")?
    .with_credentials(MetadataUrlCredentials::new())
    .client()?;
```

PHP

```
<?php

use YdbPlatform\Ydb\Ydb;
use YdbPlatform\Ydb\Auth\Implement\MetadataAuthentication;

$config = [

    // Database path
    'database'    => '/local',

    // Database endpoint
    'endpoint'    => 'localhost:2136',

    // Auto discovery (dedicated server only)
    'discovery'   => false,

    // IAM config
    'iam_config'  => [
        'insecure' => true,
        // 'root_cert_file' => './CA.pem', // Root CA file (uncomment for dedicated server)
    ],

    'credentials' => new MetadataAuthentication()
];

$ydb = new Ydb($config);
```

Аутентификация при помощи переменных окружения

Аутентификация через переменные окружения позволяет не зашивать учётные данные в код: SDK или JDBC-драйвер сам определяет режим по переменным `YDB_*` в окружении процесса. Этот способ удобен для контейнеров, CI/CD и деплоя в облако, где секреты передаются через окружение. Типичные шаги: задать нужные переменные окружения, создать провайдер аутентификации (или подключиться через JDBC без явных свойств) и выполнить запрос. Подробности о порядке выбора режима — в разделе Аутентификация; базовое подключение — в рецепте инициализации драйвера. Другие способы: токен, анонимная, метаданные, сервисный аккаунт, логин и пароль.

При использовании данного метода режим аутентификации и его параметры будут определены окружением, в котором запускается приложение, в описанном здесь порядке.

Установив одну из следующих переменных окружения, можно управлять способом аутентификации:

- `YDB_SERVICE_ACCOUNT_KEY_FILE_CREDENTIALS=<path/to/sa_key_file>` — использовать файл сервисного аккаунта в Яндекс Облаке.
- `YDB_ANONYMOUS_CREDENTIALS="1"` — Использовать анонимную аутентификацию. Актуально для тестирования против докер-контейнера с YDB.
- `YDB_METADATA_CREDENTIALS="1"` — использовать сервис метаданных внутри Яндекс Облака (Яндекс функция или виртуальная машина).
- `YDB_ACCESS_TOKEN_CREDENTIALS=<access_token>` — использовать аутентификацию с токеном.

Ниже приведены примеры кода аутентификации при помощи переменных окружения в разных YDB SDK.

C++

Native SDK

```
#include <ydb-cpp-sdk/client/driver/driver.h>
#include <ydb-cpp-sdk/client/helpers/helpers.h>

NYdb::TDriver CreateDriverFromEnvironment(const std::string& connectionString) {
    return NYdb::TDriver(NYdb::CreateFromEnvironment(connectionString));
}
```

userver

Функциональность на данный момент не поддерживается.

Go

Native SDK

```
package main

import (
  "context"
  "os"

  environ "github.com/ydb-platform/ydb-go-sdk-auth-environ"
  "github.com/ydb-platform/ydb-go-sdk/v3"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    environ.WithEnvironCredentials(ctx),
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
  ...
}
```

database/sql

```
package main

import (
  "context"
  "database/sql"
  "os"

  environ "github.com/ydb-platform/ydb-go-sdk-auth-environ"
  "github.com/ydb-platform/ydb-go-sdk/v3"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  nativeDriver, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    environ.WithEnvironCredentials(ctx),
  )
  if err != nil {
    panic(err)
```

```
  }
  defer nativeDriver.Close(ctx)
  connector, err := ydb.Connector(nativeDriver)
  if err != nil {
    panic(err)
  }
  db := sql.OpenDB(connector)
  defer db.Close()
  ...
}
```

Java

Native SDK

```
import tech.ydb.auth.iam.CloudAuthHelper;
import tech.ydb.common.transaction.TxMode;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;

public class EnvironAuthExample {
    public static void main(String[] args) throws Exception {
        // Строка подключения из переменной окружения или локальный YDB по умолчанию
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        // Режим аутентификации определяется переменными окружения YDB_*
        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString)
                .withAuthProvider(CloudAuthHelper.getAuthProviderFromEnviron())
                .build();
             QueryClient queryClient = QueryClient.newClient(transport).build()) {

            SessionRetryContext retryCtx = SessionRetryContext.create(queryClient).build();
            QueryReader reader = retryCtx.supplyResult(
                    session -> QueryReader.readFrom(session.createQuery("SELECT 1", TxMode.NONE))
            ).join().getValue();

            // Проверка подключения: выводим результат SELECT 1
            ResultSetReader rs = reader.getResultSet(0);
            if (rs.next()) {
                System.out.println("SELECT 1 = " + rs.getColumn(0).getInt32());
            }
        }
    }
}
```

JDBC

JDBC-драйвер читает переменные окружения `YDB_*` в порядке, описанном в разделе Аутентификация. Явно передавать учётные данные не нужно — достаточно пустого объекта `Properties`.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Properties;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class EnvironAuthJdbcExample {
    public static void main(String[] args) throws SQLException {
        String jdbcUrl = System.getenv().getOrDefault(
                "YDB_JDBC_URL", "jdbc:ydb:grpc://localhost:2136/local");

        // Пустые свойства: драйвер сам выберет способ аутентификации по переменным окружения
        try (Connection connection = DriverManager.getConnection(jdbcUrl, new Properties());
             Statement statement = connection.createStatement();
             ResultSet rs = statement.executeQuery("SELECT 1")) {
            if (rs.next()) {
                System.out.println("SELECT 1 = " + rs.getInt(1));
            }
        }
    }
}
```

В Spring Boot, ORM и прочих сторонних фреймворках вокруг JDBC укажите ту же JDBC-строку подключения; учётные данные из переменных окружения подхватываются драйвером так же, как в примере выше (например, через `spring.datasource.url`).

JavaScript

```
import { Driver, getCredentialsFromEnv } from 'ydb-sdk';

export async function connect(endpoint: string, database: string) {
    const authService = getCredentialsFromEnv();
    const driver = new Driver({endpoint, database, authService});
```

```
    const timeout = 10000;
    if (!await driver.ready(timeout)) {
        console.log(`Driver has not become ready in ${timeout}ms!`);
        process.exit(1);
    }
    console.log('Driver connected')
    return driver
}
```

Python

Native SDK

```
import os
import ydb

with ydb.Driver(
    connection_string=os.environ["YDB_CONNECTION_STRING"],
    credentials=ydb.credentials_from_env_variables(),
) as driver:
    driver.wait(timeout=5)
    ...
```

Native SDK (Asyncio)

```
import os
import ydb
import asyncio

async def ydb_init():
    async with ydb.aio.Driver(
        endpoint=os.environ["YDB_ENDPOINT"],
        database=os.environ["YDB_DATABASE"],
        credentials=ydb.credentials_from_env_variables(),
    ) as driver:
        await driver.wait()
        ...

asyncio.run(ydb_init())
```

SQLAlchemy

```
import os
import sqlalchemy as sa
import ydb

engine = sa.create_engine(
    "yql+ydb://localhost:2136/local",
    connect_args={
        "credentials": ydb.credentials_from_env_variables()
    }
)
with engine.connect() as connection:
    result = connection.execute(sa.text("SELECT 1"))
```

C#

Функциональность на данный момент не поддерживается.

Rust

```
use ydb::{ClientBuilder, FromEnvCredentials, YdbResult};

let client = ClientBuilder::new_from_connection_string(std::env::var("YDB_CONNECTION_STRING")?)?
    .with_credentials(FromEnvCredentials::new()?)
    .client()?;
```

PHP

```
<?php

use YdbPlatform\Ydb\Ydb;
use YdbPlatform\Ydb\Auth\EnvironCredentials;

$config = [

    // Database path
    'database'    => '/local',

    // Database endpoint
    'endpoint'    => 'localhost:2136',

    // Auto discovery (dedicated server only)
    'discovery'   => false,

    // IAM config
    'iam_config'  => [
```

```
        'insecure' => true,
        // 'root_cert_file' => './CA.pem', // Root CA file (uncomment for dedicated server)
    ],

    'credentials' => new EnvironCredentials()
];

$ydb = new Ydb($config);
```

Аутентификация при помощи логина и пароля

Статическая аутентификация (логин и пароль) передаёт пару `user` / `password` при подключении к YDB. Этот способ используется на выделенных (dedicated) инсталляциях YDB, где включена аутентификация по логину и паролю. Типичные шаги: задать логин и пароль, создать провайдер статических учётных данных, открыть транспорт и выполнить запрос. Подробности — в разделе Аутентификация; базовое подключение — в рецепте инициализации драйвера. Другие способы: токен, анонимная, переменные окружения, метаданные, сервисный аккаунт.

Ниже приведены примеры кода аутентификации при помощи логина и пароля в разных YDB SDK.

C++

Native SDK

```
#include <ydb-cpp-sdk/client/driver/driver.h>
#include <ydb-cpp-sdk/client/types/credentials/credentials.h>

NYdb::TDriver CreateDriverWithStaticCredentials(
    const std::string& connectionString,
    const std::string& user,
    const std::string& password)
{
    auto config = NYdb::TDriverConfig(connectionString)
        .SetCredentialsProviderFactory(NYdb::CreateLoginCredentialsProviderFactory({
            .User = user,
            .Password = password,
        }));

    return NYdb::TDriver(config);
}
```

userver

secdist

```
{
  "ydb_settings": {
    "db": {
      "user": "user",
      "password": "password"
    }
  }
}
```

Код инициализации `ydb::YdbComponent`, получения `ydb::TableClient` и запуска `components::MinimalServerComponentList` — как в примере из init.md.

Go

Native SDK

Передать логин и пароль можно в составе строки подключения. Например, так:

```
"grpcs://login:password@localhost:2135/local"
```

Также можно передать логин и пароль явно через опцию `ydb.WithStaticCredentials`:

```
package main

import (
  "context"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  db, err := ydb.Open(ctx,
      os.Getenv("YDB_CONNECTION_STRING"),
      ydb.WithStaticCredentials("user", "password"),
  )
  if err != nil {
      panic(err)
  }
  defer db.Close(ctx)
  ...
}
```

database/sql

Передать логин и пароль можно в составе строки подключения. Например, так:

```
package main
```

```
import (
  "context"

  _ "github.com/ydb-platform/ydb-go-sdk/v3"
)

func main() {
  db, err := sql.Open("ydb", "grpcs://login:password@localohost:2135/local")
  if err != nil {
      panic(err)
  }
  defer db.Close()
  ...
}
```

Также можно передать логин и пароль явно при инициализации драйвера через коннектор с помощью специальной опции `ydb.WithStaticCredentials`:

```
package main

import (
  "context"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  nativeDriver, err := ydb.Open(ctx,
      os.Getenv("YDB_CONNECTION_STRING"),
      ydb.WithStaticCredentials("user", "password"),
  )
  if err != nil {
      panic(err)
  }
  defer nativeDriver.Close(ctx)
  connector, err := ydb.Connector(nativeDriver)
  if err != nil {
    panic(err)
  }
  db := sql.OpenDB(connector)
  defer db.Close()
  ...
}
```

Java

Native SDK

```
import tech.ydb.common.transaction.TxMode;
import tech.ydb.auth.StaticCredentials;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;

public class StaticAuthExample {
    public static void main(String[] args) throws Exception {
        // Строка подключения из переменной окружения или локальный YDB по умолчанию
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        String username = System.getenv("YDB_USER");
        String password = System.getenv("YDB_PASSWORD");
        if (username == null || username.isEmpty() || password == null) {
            throw new IllegalStateException("Задайте переменные окружения YDB_USER и YDB_PASSWORD");
        }

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString)
                .withAuthProvider(new StaticCredentials(username, password))
                .build();
             QueryClient queryClient = QueryClient.newClient(transport).build()) {

            SessionRetryContext retryCtx = SessionRetryContext.create(queryClient).build();
            QueryReader reader = retryCtx.supplyResult(
                    session -> QueryReader.readFrom(session.createQuery("SELECT 1", TxMode.NONE))
            ).join().getValue();

            // Проверка подключения: выводим результат SELECT 1
            ResultSetReader rs = reader.getResultSet(0);
            if (rs.next()) {
                System.out.println("SELECT 1 = " + rs.getColumn(0).getInt32());
            }
        }
    }
}
```

JDBC

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Properties;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class StaticAuthJdbcExample {
    public static void main(String[] args) throws SQLException {
        String jdbcUrl = System.getenv().getOrDefault(
                "YDB_JDBC_URL", "jdbc:ydb:grpc://localhost:2136/local");

        String username = System.getenv("YDB_USER");
        String password = System.getenv("YDB_PASSWORD");
        if (username == null || username.isEmpty() || password == null || password.isEmpty()) {
            throw new IllegalStateException("Задайте переменные окружения YDB_USER и YDB_PASSWORD");
        }

        Properties props = new Properties();
        props.setProperty("username", username);
        props.setProperty("password", password);

        try (Connection connection = DriverManager.getConnection(jdbcUrl, props);
             Statement statement = connection.createStatement();
             ResultSet rs = statement.executeQuery("SELECT 1")) {
            if (rs.next()) {
                System.out.println("SELECT 1 = " + rs.getInt(1));
            }
        }
    }
}
```

Логин и пароль также можно передать вторым и третьим аргументами метода `DriverManager.getConnection(jdbcUrl, username, password)`.

В Spring Boot, ORM и прочих сторонних фреймворках вокруг JDBC задайте те же JDBC URL, логин и пароль, что и в примере выше (например, `spring.datasource.url`, `spring.datasource.username`, `spring.datasource.password` или эквивалент в конфигурации пула).

JavaScript

```
import { Driver } from "@ydbjs/core";
import { StaticCredentialsProvider } from "@ydbjs/auth/static";

const driver = new Driver("grpc://localhost:2136/local", {
  credentialsProvider: new StaticCredentialsProvider(
    { username: user, password: password },
    "grpc://localhost:2136",
  ),
});

await driver.ready();
```

Python

Native SDK

```
import os
import ydb

config = ydb.DriverConfig(
    endpoint=os.environ["YDB_ENDPOINT"],
    database=os.environ["YDB_DATABASE"],
)

credentials = ydb.StaticCredentials(
    driver_config=config,
    user=os.environ["YDB_USER"],
    password=os.environ["YDB_PASSWORD"]
)

with ydb.Driver(driver_config=config, credentials=credentials) as driver:
    driver.wait(timeout=5)
    ...
```

Native SDK (Asyncio)

```
import os
import ydb
import asyncio

config = ydb.DriverConfig(
    endpoint=os.environ["YDB_ENDPOINT"],
    database=os.environ["YDB_DATABASE"],
)
```

```
credentials = ydb.StaticCredentials(
    driver_config=config,
    user=os.environ["YDB_USER"],
    password=os.environ["YDB_PASSWORD"],
)

async def ydb_init():
    async with ydb.aio.Driver(driver_config=config, credentials=credentials) as driver:
        await driver.wait()
        ...

asyncio.run(ydb_init())
```

SQLAlchemy

```
import os
import sqlalchemy as sa

engine = sa.create_engine(
    "yql+ydb://localhost:2136/local",
    connect_args={
        "credentials": {
            "username": os.environ["YDB_USER"],
            "password": os.environ["YDB_PASSWORD"]
        }
    }
)
with engine.connect() as connection:
    result = connection.execute(sa.text("SELECT 1"))
```

C#

```
using Ydb.Sdk.Ado;

await using var dataSource = new YdbDataSource(
    "Host=localhost;Port=2136;Database=/local;User=user;Password=password");
await using var connection = await dataSource.OpenConnectionAsync();
```

Rust

```
use ydb::{ClientBuilder, StaticCredentials, YdbResult};

let client = ClientBuilder::new_from_connection_string("grpc://localhost:2136?database=local")?
    .with_credentials(StaticCredentials::new(
        std::env::var("YDB_USER")?,
        std::env::var("YDB_PASSWORD")?,
        http::Uri::from_static("grpc://localhost:2136"),
        "local".into(),
    ))
    .client()?;
```

PHP

```
<?php

use YdbPlatform\Ydb\Ydb;
use YdbPlatform\Ydb\Auth\Implement\StaticAuthentication;

$config = [

    // Database path
    'database'    => '/local',

    // Database endpoint
    'endpoint'    => 'localhost:2136',

    // Auto discovery (dedicated server only)
    'discovery'   => false,

    // IAM config
    'iam_config'  => [
        'insecure' => true,
        // 'root_cert_file' => './CA.pem', // Root CA file (uncomment for dedicated server)
    ],

    'credentials' => new StaticAuthentication($user, $password)
];

$ydb = new Ydb($config);
```

Балансировка

YDB использует клиентскую балансировку, потому что клиентская балансировка эффективнее, когда на базу данных приходит большой трафик от множества клиентских приложений.
В большинстве случаев она просто работает в YDB SDK. Однако иногда нужны специфичные настройки клиентской балансировки, например, для уменьшения серверных хопов, сокращения времени запроса или распределения нагрузки по зонам доступности.

Следует отметить, что клиентская балансировка работает в ограниченном режиме в отношении сессий YDB. Клиентская балансировка в YDB SDK осуществляется только в момент создания новой сессии YDB на конкретной ноде. После того, как сессия создана, все запросы на этой сессии направляются на ту ноду, на которой была создана сессия. Балансировка запросов на одной и той же сессии YDB между разными нодами YDB не происходит.

В данном разделе содержатся рецепты кода с настройкой клиентской балансировки в разных YDB SDK

Содержание:

- Равномерный случайный выбор
- Предпочитать ближайший дата-центр
- Предпочитать конкретную зону доступности
- Предпочитать пайл с конкретным состоянием

Равномерный случайный выбор

YDB SDK использует алгоритм `random_choice` (равномерную случайную балансировку) по умолчанию, кроме C++ SDK, который использует алгоритм "предпочитать ближайший дата-центр" по умолчанию.

Ниже приведены примеры кода принудительной установки алгоритма балансировки "равномерный случайный выбор" в разных YDB SDK.

C++

Native SDK

```
#include <ydb-cpp-sdk/client/driver/driver.h>

int main() {
  auto connectionString = std::string(std::getenv("YDB_CONNECTION_STRING"));

  auto driverConfig = NYdb::TDriverConfig(connectionString)
    .SetBalancingPolicy(NYdb::TBalancingPolicy::UseAllNodes());

  NYdb::TDriver driver(driverConfig);
  // ...
  driver.Stop(true);
  return 0;
}
```

userver

static config

```
ydb:
    databases:
        db:
            endpoint: grpc://localhost:2136
            database: /local
            prefer_local_dc: false
```

Код инициализации `ydb::YdbComponent`, получения `ydb::TableClient` и запуска `components::MinimalServerComponentList` — как в примере из init.md.

Go

Native SDK

```
package main

import (
  "context"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/balancers"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithBalancer(
      balancers.RandomChoice(),
    ),
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
  // ...
}
```

database/sql

Клиентская балансировка в `database/sql` драйвере для YDB осуществляется только в момент установления нового соединения (в терминах `database/sql`), которое представляет собой сессию YDB на конкретной ноде. После того, как сессия создана, все запросы на этой сессии направляются на ту ноду, на которой была создана сессия. Балансировка запросов на одной и той же сессии YDB между разными нодами YDB не происходит.

Пример кода установки алгоритма балансировки "равномерный случайный выбор":

```
package main

import (
  "context"
  "database/sql"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
```

```
  "github.com/ydb-platform/ydb-go-sdk/v3/balancers"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  nativeDriver, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithBalancer(
      balancers.RandomChoice(),
    ),
  )
  if err != nil {
    panic(err)
  }
  defer nativeDriver.Close(ctx)

  connector, err := ydb.Connector(nativeDriver)
  if err != nil {
    panic(err)
  }

  db := sql.OpenDB(connector)
  defer db.Close()
  // ...
}
```

Python

Native SDK

```
import os
import ydb

driver_config = ydb.DriverConfig(
    endpoint=os.environ["YDB_ENDPOINT"],
    database=os.environ["YDB_DATABASE"],
    credentials=ydb.credentials_from_env_variables(),
    use_all_nodes=True,  # равномерный случайный выбор
)

with ydb.Driver(driver_config) as driver:
    driver.wait(timeout=5)
    # ...
```

Native SDK (Asyncio)

```
import os
import ydb
import asyncio

async def ydb_init():
    driver_config = ydb.DriverConfig(
        endpoint=os.environ["YDB_ENDPOINT"],
        database=os.environ["YDB_DATABASE"],
        credentials=ydb.credentials_from_env_variables(),
        use_all_nodes=True,  # равномерный случайный выбор
    )
    async with ydb.aio.Driver(driver_config) as driver:
        await driver.wait()
        # ...

asyncio.run(ydb_init())
```

SQLAlchemy

```
import os
import sqlalchemy as sa

engine = sa.create_engine(
    os.environ["YDB_SQLALCHEMY_URL"],
    connect_args={
        "driver_config_kwargs": {
            "use_all_nodes": True,  # равномерный случайный выбор
        }
    },
)
```

C#

Этот алгоритм используется по умолчанию.

JavaScript

Функциональность на данный момент не поддерживается.

Java

Native SDK

Алгоритм «равномерный случайный выбор» в Java SDK задаётся политикой `USE_ALL_NODES` в `BalancingSettings` (это поведение по умолчанию, если настройки не переопределять).

```
import tech.ydb.common.transaction.TxMode;
import tech.ydb.core.grpc.BalancingSettings;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;

public class RandomChoiceExample {
    public static void main(String[] args) {
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString)
                // Явная установка политики random_choice (USE_ALL_NODES)
                .withBalancingSettings(BalancingSettings.fromPolicy(BalancingSettings.Policy.USE_ALL_NODES))
                .build();
             QueryClient queryClient = QueryClient.newClient(transport).build()) {

            SessionRetryContext retryCtx = SessionRetryContext.create(queryClient).build();

            QueryReader reader = retryCtx.supplyResult(session -> QueryReader.readFrom(
                    session.createQuery("SELECT 1 AS value", TxMode.NONE, Params.empty())
            )).join().getValue();

            ResultSetReader rs = reader.getResultSet(0);
            if (rs.next()) {
                System.out.println("SELECT 1 = " + rs.getColumn("value").getInt32());
            }
        }
    }
}
```

JDBC

По умолчанию JDBC-драйвер использует политику `USE_ALL_NODES` (равномерный случайный выбор). Дополнительные параметры балансировки можно передать через `Properties` или query-параметры JDBC URL — см. свойства JDBC-драйвера.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class JdbcRandomChoiceExample {
    public static void main(String[] args) throws SQLException {
        // USE_ALL_NODES — поведение по умолчанию, дополнительные параметры не нужны
        try (Connection connection = DriverManager.getConnection("jdbc:ydb:grpc://localhost:2136/local");
             Statement statement = connection.createStatement();
             ResultSet rs = statement.executeQuery("SELECT 1 AS value")) {

            if (rs.next()) {
                System.out.println("SELECT 1 = " + rs.getInt("value"));
            }
        }
    }
}
```

В Spring Boot, ORM и прочих сторонних фреймворках вокруг JDBC укажите ту же JDBC-строку подключения и параметры балансировки, что и при прямом использовании драйвера (например, `spring.datasource.url` с нужными query-параметрами или свойства `DataSource`).

Rust

Политика RandomChoice (случайный выбор endpoint среди нод discovery) используется **по умолчанию** — дополнительная настройка не требуется.

PHP

Функциональность на данный момент не поддерживается.

Предпочитать ближайший дата-центр

Ниже приведены примеры кода установки опции алгоритма балансировки "предпочитать ближайший дата-центр" в разных YDB SDK.

C++

Native SDK

YDB C++ SDK использует алгоритм `prefer_local_dc` (предпочитать ближайший дата-центр) по умолчанию.

```
#include <ydb-cpp-sdk/client/driver/driver.h>

int main() {
  auto connectionString = std::string(std::getenv("YDB_CONNECTION_STRING"));

  auto driverConfig = NYdb::TDriverConfig(connectionString)
    .SetBalancingPolicy(NYdb::TBalancingPolicy::UsePreferableLocation());

  NYdb::TDriver driver(driverConfig);
  // ...
  driver.Stop(true);
  return 0;
}
```

userver

static config

```
ydb:
    databases:
        db:
            endpoint: grpc://localhost:2136
            database: /local
            prefer_local_dc: true
```

Код инициализации `ydb::YdbComponent`, получения `ydb::TableClient` и запуска `components::MinimalServerComponentList` — как в примере из init.md.

Go

Native SDK

```
package main

import (
  "context"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/balancers"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithBalancer(
      balancers.PreferLocalDC(
        balancers.RandomChoice(),
      ),
    ),
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
  // ...
}
```

database/sql

Клиентская балансировка в `database/sql` драйвере для YDB осуществляется только в момент установления нового соединения (в терминах `database/sql`), которое представляет собой сессию YDB на конкретной ноде. После того, как сессия создана, все запросы на этой сессии направляются на ту ноду, на которой была создана сессия. Балансировка запросов на одной и той же сессии YDB между разными нодами YDB не происходит.

Пример кода установки алгоритма балансировки "предпочитать ближайший дата-центр":

```
package main

import (
  "context"
  "database/sql"
  "os"
```

```
  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/balancers"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  nativeDriver, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithBalancer(
      balancers.PreferLocalDC(
        balancers.RandomChoice(),
      ),
    ),
  )
  if err != nil {
    panic(err)
  }
  defer nativeDriver.Close(ctx)

  connector, err := ydb.Connector(nativeDriver)
  if err != nil {
    panic(err)
  }

  db := sql.OpenDB(connector)
  defer db.Close()
  // ...
}
```

Python

Native SDK

```
import os
import ydb

driver_config = ydb.DriverConfig(
    endpoint=os.environ["YDB_ENDPOINT"],
    database=os.environ["YDB_DATABASE"],
    credentials=ydb.credentials_from_env_variables(),
    use_all_nodes=False,  # предпочитать ближайший дата-центр
)

with ydb.Driver(driver_config) as driver:
    driver.wait(timeout=5)
    # ...
```

Native SDK (Asyncio)

```
import os
import ydb
import asyncio

async def ydb_init():
    driver_config = ydb.DriverConfig(
        endpoint=os.environ["YDB_ENDPOINT"],
        database=os.environ["YDB_DATABASE"],
        credentials=ydb.credentials_from_env_variables(),
        use_all_nodes=False,  # предпочитать ближайший дата-центр
    )
    async with ydb.aio.Driver(driver_config) as driver:
        await driver.wait()
        # ...

asyncio.run(ydb_init())
```

SQLAlchemy

```
import os
import sqlalchemy as sa

engine = sa.create_engine(
    os.environ["YDB_SQLALCHEMY_URL"],
    connect_args={
        "driver_config_kwargs": {
            "use_all_nodes": False,  # предпочитать ближайший дата-центр
        }
    },
)
```

C#

Функциональность на данный момент не поддерживается.

JavaScript

Функциональность на данный момент не поддерживается.

Java

Native SDK

```
import tech.ydb.common.transaction.TxMode;
import tech.ydb.core.grpc.BalancingSettings;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;

public class PreferLocalExample {
    public static void main(String[] args) {
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString)
                // Автоопределение ближайшего дата-центра по задержке до нод
                .withBalancingSettings(BalancingSettings.detectLocalDs())
                .build();
             QueryClient queryClient = QueryClient.newClient(transport).build()) {

            SessionRetryContext retryCtx = SessionRetryContext.create(queryClient).build();

            QueryReader reader = retryCtx.supplyResult(session -> QueryReader.readFrom(
                    session.createQuery("SELECT 1 AS value", TxMode.NONE, Params.empty())
            )).join().getValue();

            ResultSetReader rs = reader.getResultSet(0);
            if (rs.next()) {
                System.out.println("SELECT 1 = " + rs.getColumn("value").getInt32());
            }
        }
    }
}
```

JDBC

Укажите локальный дата-центр через `localDatacenter` в JDBC URL — см. свойства JDBC-драйвера.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class JdbcPreferLocalExample {
    public static void main(String[] args) throws SQLException {
        // JDBC-драйвер не поддерживает автоматическое определение ближайшего DC.
        // Явно укажите зону доступности через параметр localDatacenter —
        // см. рецепт [Балансировка нагрузки — Предпочитать зону доступности](./balancing-prefer-location.m
d).
        try (Connection connection = DriverManager.getConnection(
                    "jdbc:ydb:grpc://localhost:2136/local?localDatacenter=VLA");
             Statement statement = connection.createStatement();
             ResultSet rs = statement.executeQuery("SELECT 1 AS value")) {

            if (rs.next()) {
                System.out.println("SELECT 1 = " + rs.getInt("value"));
            }
        }
    }
}
```

В Spring Boot, ORM и прочих сторонних фреймворках вокруг JDBC укажите ту же JDBC-строку подключения и параметры балансировки, что и при прямом использовании драйвера (например, `spring.datasource.url` или свойства `DataSource`).

Rust

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#239

PHP

Функциональность на данный момент не поддерживается.

Предпочитать конкретную зону доступности

Ниже приведены примеры кода установки опции алгоритма балансировки "предпочитать зону доступности" в разных YDB SDK.

C++

Native SDK

В C++ SDK можно выбрать только одну зону доступности в качестве предпочитаемой.

```
#include <ydb-cpp-sdk/client/driver/driver.h>

int main() {
  auto connectionString = std::string(std::getenv("YDB_CONNECTION_STRING"));

  auto driverConfig = NYdb::TDriverConfig(connectionString)
    .SetBalancingPolicy(NYdb::TBalancingPolicy::UsePreferableLocation("datacenter1"));

  NYdb::TDriver driver(driverConfig);
  // ...
  driver.Stop(true);
  return 0;
}
```

userver

Функциональность на данный момент не поддерживается.

Go

Native SDK

```
package main

import (
  "context"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/balancers"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithBalancer(
      balancers.PreferLocations(
        balancers.RandomChoice(),
        "a",
        "b",
      ),
    ),
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
  // ...
}
```

database/sql

Клиентская балансировка в `database/sql` драйвере для YDB осуществляется только в момент установления нового соединения (в терминах `database/sql`), которое представляет собой сессию YDB на конкретной ноде. После того, как сессия создана, все запросы на этой сессии направляются на ту ноду, на которой была создана сессия. Балансировка запросов на одной и той же сессии YDB между разными нодами YDB не происходит.

Пример кода установки алгоритма балансировки "предпочитать зону доступности":

```
package main

import (
  "context"
  "database/sql"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/balancers"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  nativeDriver, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithBalancer(
      balancers.PreferLocations(
```

```
        balancers.RandomChoice(),
        "a",
        "b",
      ),
    ),
  )
  if err != nil {
    panic(err)
  }
  defer nativeDriver.Close(ctx)

  connector, err := ydb.Connector(nativeDriver)
  if err != nil {
    panic(err)
  }

  db := sql.OpenDB(connector)
  defer db.Close()
  // ...
}
```

Python

Функциональность на данный момент не поддерживается.

C#

Функциональность на данный момент не поддерживается.

JavaScript

Функциональность на данный момент не поддерживается.

Java

В **Java SDK** предпочтение зоны доступности задаётся в настройках gRPC-транспорта.

Native SDK

```
import tech.ydb.common.transaction.TxMode;
import tech.ydb.core.grpc.BalancingSettings;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;

public class PreferLocationExample {
    public static void main(String[] args) {
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");
        // Имя предпочитаемой зоны доступности (например, VLA, SAS, MYT)
        String zone = System.getenv().getOrDefault("YDB_PREFER_LOCATION", "VLA");

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString)
                .withBalancingSettings(BalancingSettings.fromLocation(zone))
                .build();
             QueryClient queryClient = QueryClient.newClient(transport).build()) {

            SessionRetryContext retryCtx = SessionRetryContext.create(queryClient).build();

            QueryReader reader = retryCtx.supplyResult(session -> QueryReader.readFrom(
                    session.createQuery("SELECT 1 AS value", TxMode.NONE, Params.empty())
            )).join().getValue();

            ResultSetReader rs = reader.getResultSet(0);
            if (rs.next()) {
                System.out.println("SELECT 1 = " + rs.getColumn("value").getInt32());
            }
        }
    }
}
```

JDBC

Аналог `BalancingSettings.fromLocation()` — свойство `localDatacenter` в JDBC URL (см. свойства JDBC-драйвера).

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class JdbcPreferLocationExample {
    public static void main(String[] args) throws SQLException {
        String jdbcUrl = System.getenv().getOrDefault(
                "YDB_JDBC_URL", "jdbc:ydb:grpc://localhost:2136/local");
        // Имя предпочитаемой зоны доступности (например, VLA, SAS, MYT)
        String zone = System.getenv().getOrDefault("YDB_PREFER_LOCATION", "VLA");

        try (Connection connection = DriverManager.getConnection(
```

```
                    jdbcUrl + (jdbcUrl.contains("?") ? "&" : "?") + "localDatacenter=" + zone);
             Statement statement = connection.createStatement();
             ResultSet rs = statement.executeQuery("SELECT 1 AS value")) {

            if (rs.next()) {
                System.out.println("SELECT 1 = " + rs.getInt("value"));
            }
        }
    }
}
```

В Spring Boot, ORM и прочих сторонних фреймворках вокруг JDBC передайте те же JDBC URL и параметры зоны доступности, что и при прямом подключении (например, в `spring.datasource.url` или свойствах пула).

Rust

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#238

PHP

Функциональность на данный момент не поддерживается.

Предпочитать пайл с конкретным состоянием

Ниже приведен пример кода установки опции алгоритма балансировки "предпочитать пайл с конкретным состоянием" в YDB SDK.

Если при установке опции состояние не задано, SDK предпочитает PRIMARY пайл.

Данная опция имеет смысл, только если кластер находится в bridge режиме. Если это неверно, SDK будет использовать равномерную случайную балансировку.

C++

Native SDK

```
#include <ydb-cpp-sdk/client/driver/driver.h>

int main() {
  auto connectionString = std::string(std::getenv("YDB_CONNECTION_STRING"));

  auto driverConfig = NYdb::TDriverConfig(connectionString)
    .SetBalancingPolicy(NYdb::TBalancingPolicy::UsePreferablePileState(NYdb::EPileState::PRIMARY));

  NYdb::TDriver driver(driverConfig);
  // ...
  driver.Stop(true);
  return 0;
}
```

userver

Функциональность на данный момент не поддерживается.

Go

Функциональность на данный момент не поддерживается.

Python

Функциональность на данный момент не поддерживается.

C#

Функциональность на данный момент не поддерживается.

JavaScript

Функциональность на данный момент не поддерживается.

Java

Функциональность на данный момент не поддерживается.

Rust

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку в Rust SDK: ydb-rs-sdk#491

PHP

Функциональность на данный момент не поддерживается.

Выполнение повторных попыток

YDB является распределенной СУБД с автоматическим масштабированием под нагрузку.
На серверной стороне могут проводиться работы, серверные стойки или целые дата-центры могут быть временно отключены.
В связи с этим допускаются некоторые ошибки при работе с YDB.
В зависимости от типа ошибки следует по разному реагировать на них.
YDB SDK для обеспечения высокой доступности предоставляют встроенные средства выполнения повторных попыток,
в которых учтены типы ошибок и закреплена реакция на них.

Ниже приведены примеры кода использования встроенных в YDB SDK средств выполнения повторных попыток:

C++

Native SDK

В YDB C++ SDK выполнение повторных попыток с корректной обработкой ошибок реализовано в нескольких программных интерфейсах:

Синхронное выполнение повторных попыток

Для выполнения запросов с автоматическими повторными попытками используется метод `RetryQuerySync`.
Метод принимает лямбда-функцию, которая получает объект сессии и возвращает результат запроса.
YDB C++ SDK автоматически анализирует ошибки и выполняет повторные попытки в соответствии с их типом.

Пример кода, использующего `RetryQuerySync`:

```
#include <ydb-cpp-sdk/client/query/client.h>

void ExecuteQueryWithRetry(NYdb::NQuery::TQueryClient client) {
    auto result = client.RetryQuerySync([](NYdb::NQuery::TSession session) -> NYdb::TStatus {
        auto query = R"(
            SELECT series_id, title
            FROM series
            WHERE series_id = 1;
        )";

        auto result = session.ExecuteQuery(
            query,
            NYdb::NQuery::TTxControl::BeginTx(NYdb::NQuery::TTxSettings::SerializableRW()).CommitTx()
        ).GetValueSync();

        if (!result.IsSuccess()) {
            return result;
        }

        // Обработка результата запроса
        auto resultSet = result.GetResultSet(0);
        NYdb::TResultSetParser parser(resultSet);
        while (parser.TryNextRow()) {
            std::cout << "Series"
                << ", Id: " << parser.ColumnParser("series_id").GetOptionalUint64().value()
                << ", Title: " << parser.ColumnParser("title").GetOptionalUtf8().value()
                << std::endl;
        }

        return result;
    });

    if (!result.IsSuccess()) {
        // Обработка ошибки после всех попыток
        std::cerr << "Query failed: " << result.GetIssues().ToString() << std::endl;
    }
}
```

Асинхронное выполнение повторных попыток

Для асинхронного выполнения запросов с автоматическими повторными попытками используется метод `RetryQuery`.
Метод возвращает `NThreading::TFuture`, что позволяет выполнять операции асинхронно.

Пример кода, использующего `RetryQuery`:

```
#include <ydb-cpp-sdk/client/query/client.h>

void ExecuteQueryWithRetryAsync(NYdb::NQuery::TQueryClient client) {
    auto future = client.RetryQuery([](NYdb::NQuery::TSession session) -> NYdb::TAsyncStatus {
        auto query = R"(
            SELECT series_id, title, release_date
            FROM series
            WHERE series_id = 1;
        )";

        return session.ExecuteQuery(
            query,
            NYdb::NQuery::TTxControl::BeginTx(NYdb::NQuery::TTxSettings::SerializableRW()).CommitTx()
        ).Apply([](const NYdb::NQuery::TAsyncExecuteQueryResult& asyncResult) -> NYdb::TStatus {
            auto result = asyncResult.GetValue();
            if (!result.IsSuccess()) {
                return result;
            }
```

```
            // Обработка результата запроса
            auto resultSet = result.GetResultSet(0);
            NYdb::TResultSetParser parser(resultSet);
            while (parser.TryNextRow()) {
                std::cout << "Series"
                    << ", Id: " << parser.ColumnParser("series_id").GetOptionalUint64().value()
                    << ", Title: " << parser.ColumnParser("title").GetOptionalUtf8().value()
                    << std::endl;
            }

            return result;
        });
    });

    // Ожидание завершения
    auto status = future.GetValueSync();
    if (!status.IsSuccess()) {
        std::cerr << "Query failed: " << status.GetIssues().ToString() << std::endl;
    }
}
```

Выполнение повторных попыток при работе со стриминговыми запросами

Для выполнения стриминговых запросов с автоматическими повторными попытками используется метод `StreamExecuteQuery`.
Стриминговые запросы позволяют обрабатывать большие объемы данных, получая результаты частями.

Пример кода, использующего `RetryQuerySync` со `StreamExecuteQuery`:

```
#include <ydb-cpp-sdk/client/query/client.h>

void StreamQueryWithRetry(NYdb::NQuery::TQueryClient client) {
    auto result = client.RetryQuerySync([](NYdb::NQuery::TSession session) -> NYdb::TStatus {
        auto query = R"(
            SELECT series_id, title, release_date
            FROM series
            WHERE series_id > 0;
        )";

        auto resultStreamQuery = session.StreamExecuteQuery(
            query,
            NYdb::NQuery::TTxControl::NoTx()
        ).GetValueSync();

        if (!resultStreamQuery.IsSuccess()) {
            return resultStreamQuery;
        }

        // Обработка результатов по частям
        bool eos = false;
        while (!eos) {
            auto streamPart = resultStreamQuery.ReadNext().ExtractValueSync();

            if (!streamPart.IsSuccess()) {
                eos = true;
                if (!streamPart.EOS()) {
                    return streamPart;
                }
                continue;
            }

            if (streamPart.HasResultSet()) {
                auto rs = streamPart.ExtractResultSet();
                NYdb::TResultSetParser parser(rs);
                while (parser.TryNextRow()) {
                    std::cout << "Series"
                        << ", Id: " << parser.ColumnParser("series_id").GetOptionalUint64().value()
                        << ", Title: " << parser.ColumnParser("title").GetOptionalUtf8().value()
                        << std::endl;
                }
            }
        }

        return resultStreamQuery;
    });

    if (!result.IsSuccess()) {
        std::cerr << "Stream query failed: " << result.GetIssues().ToString() << std::endl;
    }
}
```

Настройка параметров повторных попыток

Пользователь может настраивать поведение механизма повторных попыток с помощью класса `TRetryOperationSettings`:

- `MaxRetries(uint32_t)` - максимальное количество повторных попыток (по умолчанию 10)

- `Idempotent(bool)` - признак идемпотентности операции. Идемпотентные операции повторяются для более широкого списка ошибок
- `RetryNotFound(bool)` - повторять ли операции, вернувшие статус `NOT_FOUND` (по умолчанию true)
- `MaxTimeout(TDuration)` - максимальное время выполнения всех попыток
- `FastBackoffSettings(TBackoffSettings)` - настройки быстрых повторов
- `SlowBackoffSettings(TBackoffSettings)` - настройки медленных повторов

Пример использования настроек повторных попыток:

```
#include <ydb-cpp-sdk/client/query/client.h>
#include <ydb-cpp-sdk/client/retry/retry.h>

void ExecuteWithCustomRetry(NYdb::NQuery::TQueryClient client) {
    auto retrySettings = NYdb::NRetry::TRetryOperationSettings()
        .Idempotent(true)
        .MaxRetries(20)
        .MaxTimeout(TDuration::Seconds(30));

    auto result = client.RetryQuerySync([](NYdb::NQuery::TSession session) -> NYdb::TStatus {
        auto query = R"(
            UPSERT INTO series (series_id, title)
            VALUES (10, "New Series");
        )";

        auto result = session.ExecuteQuery(
            query,
            NYdb::NQuery::TTxControl::BeginTx(NYdb::NQuery::TTxSettings::SerializableRW()).CommitTx()
        ).GetValueSync();

        if (!result.IsSuccess()) {
            return result;
        }

        // Обработка результата запроса
        std::cout << "Query executed successfully" << std::endl;
        return result;
    }, retrySettings);

    if (!result.IsSuccess()) {
        std::cerr << "Operation failed: " << result.GetIssues().ToString() << std::endl;
    }
}
```

userver

В `ydb::TableClient` выполнение повторных попыток с корректной обработкой ошибок реализовано во всех методах. userver автоматически анализирует ошибки и выполняет повторные попытки в соответствии с их типом.

Пользователь может настраивать поведение механизма повторных попыток с помощью `ydb::OperationSettings` и `ydb::RetryTxSettings`:

- `retries` - максимальное количество повторных попыток
- `is_idempotent` - признак идемпотентности операции. Идемпотентные операции повторяются для более широкого списка ошибок
- `client_timeout_ms` или `timeout_ms` соответственно - максимальное время выполнения всех попыток
- `get_session_timeout` (актуально только для `ydb::OperationSettings`) - таймаут на получение сессии
- `get_session_settings`, `commit_settings` и `rollback_settings` (актуально только для `ydb::RetryTxSettings`) - настройки для запросов получения сессии, коммита или отката транзакции

`ydb::RetryTxSettings` используется только для метода `ydb::TableClient::RetryTx`, который выполняет интерактивную транзакцию с выполнением повторных попыток при ошибках для всей транзакции.

Секция `ydb.operation-settings` в static config задаёт значения по умолчанию: если при вызове в коде поле не задано (`std::nullopt` или ноль там, где это означает "не задано"), используется значение из конфига, иначе из кода.

static config

```
ydb:
    operation-settings:
        retries: 5
        client-timeout: 2s
        get-session-timeout: 10s
```

```
#include <userver/ydb/table.hpp>

void RetryExamples(ydb::TableClient& client) {
    client.ExecuteQuery(
        ydb::OperationSettings{
            .retries = 7,
            .is_idempotent = true,
        },
        ydb::Query{R"(
            UPSERT INTO series (series_id, title)
            VALUES (10, "New Series");
```

```
        )"}
    );

    client.RetryTx(
        ydb::RetryTxSettings{
            .retries = 3,
            .is_idempotent = true,
        },
        [](ydb::TxActor& tx) {
            tx.Execute(ydb::Query{R"(
                UPSERT INTO series (series_id, title)
                VALUES (11, "Other Series");
            )"});
        }
    );
}
```

Go

Native SDK

В YDB Go SDK корректная обработка ошибок закреплена в нескольких программных интерфейсах:

Функция повторов общего назначения

Основная логика обработки ошибок реализуется функцией-помощником `retry.Retry`
Подробности выполнения повторных запросов максимально скрыты.
Пользователь может влиять на логику работы функции `retry.Retry` двумя способами:

- через контекст (можно устанавливать deadline и cancel);
- через флаг идемпотентности операции `retry.WithIdempotent()`. По умолчанию операция считается неидемпотентной.

Пользователь передает свою функцию в `retry.Retry`, которая по своей сигнатуре должна возвращать ошибку.
В случае, если из пользовательской функции вернулся `nil`, то повторные запросы прекращаются.
В случае, если из пользовательской функции вернулась ошибка, YDB Go SDK пытается идентифицировать эту ошибку и в зависимости от нее выполняет повторные попытки.

Пример кода, использующего функцию `retry.Retry`:

```
package main

import (
    "context"
    "time"

    "github.com/ydb-platform/ydb-go-sdk/v3"
    "github.com/ydb-platform/ydb-go-sdk/v3/retry"
)

func main() {
    db, err := ydb.Open(ctx,
        os.Getenv("YDB_CONNECTION_STRING"),
    )
    if err != nil {
        panic(err)
    }
    defer db.Close(ctx)
    var cancel context.CancelFunc
    // fix deadline for retries
    ctx, cancel := context.WithTimeout(ctx, time.Second)
    err = retry.Retry(
        ctx,
        func(ctx context.Context) error {
            whoAmI, err := db.Discovery().WhoAmI(ctx)
            if err != nil {
                return err
            }
            fmt.Println(whoAmI)
            return nil
        },
        retry.WithIdempotent(true),
    )
    if err != nil {
        panic(err)
    }
}
```

Выполнение повторных попыток при ошибках на объекте сессии YDB

Для повторной обработки ошибок на уровне сессии сервиса таблиц YDB есть функция `db.Table().Do(ctx, op)`, которая предоставляет подготовленную сессию для выполнения запросов.
Функция `db.Table().Do(ctx, op)` использует пакет `retry`, а также следит за временем жизни сессий YDB.
Из пользовательской операции `op` согласно ее сигнатуре требуется возвращать ошибку или `nil`, чтобы драйвер смог по типу ошибки "понять" что нужно делать: потерять операцию или нет, с задержкой или нет, на этой же сессии или новой.
Пользователь может влиять на логику выполнения повторных запросов через контекст и признак идемпотентности, а YDB Go SDK интерпретирует возвращаемые из `op` ошибки.

Пример кода, использующего функцию `db.Table().Do(ctx, op)`:

```
err := db.Table().Do(ctx, func(ctx context.Context, s table.Session) (err error) {
    desc, err = s.DescribeTableOptions(ctx)
    return
}, table.WithIdempotent())
if err != nil {
    return err
}
```

Выполнение повторных попыток при ошибках на объекте интерактивной транзакции YDB

Для повторной обработки ошибок на уровне интерактивной транзакции сервиса таблиц YDB есть функция `db.Table().DoTx(ctx, txOp)`, которая предоставляет подготовленную транзакцию YDB на сессии для выполнения запросов.
Функция `db.Table().DoTx(ctx, txOp)` использует пакет `retry`, а также следит за временем жизни сессий YDB.
Из пользовательской операции `txOp` согласно ее сигнатуре требуется возвращать ошибку или `nil`, чтобы драйвер смог по типу ошибки "понять" что нужно делать: поторять операцию или нет, с задержкой или нет, на этой же транзакции или новой.
Пользователь может влиять на логику выполнения повторных запросов через контекст и признак идемпотентности, а YDB Go SDK интерпретирует возвращаемые из `op` ошибки.

Пример кода, использующего функцию `db.Table().DoTx(ctx, op)`:

```
err := db.Table().DoTx(ctx, func(ctx context.Context, tx table.TransactionActor) error {
    _, err := tx.Execute(ctx,
        "DECLARE $id AS Int64; INSERT INTO test (id, val) VALUES($id, 'asd')",
        table.NewQueryParameters(table.ValueParam("$id", types.Int64Value(100500))),
    )
    return err
}, table.WithIdempotent())
if err != nil {
    return err
}
```

Запросы к остальным сервисам YDB

(`db.Scripting()`, `db.Scheme()`, `db.Coordination()`, `db.Ratelimiter()`, `db.Discovery()`) также используют внутри себя функцию `retry.Retry` для выполнения повторных запросов и не требуют использования внешних вспомогательных функций для повторов.

database/sql

Стандартный пакет `database/sql` использует внутреннюю логику выполнения повторов на основе тех ошибок, что возвращает конкретная реализация драйвера.
Так, в коде пакета `database/sql` во многих местах можно встретить политику повторов из трех попыток:

- 2 попытки выполнить на существующем соединении или новом (если пул соединений `database/sql` пуст)
- 1 попытка выполнить на новом соединении.

В большинстве случаев такой политики повторов достаточно, чтобы пережить временную недоступность нод YDB или проблемы с сессией YDB.

YDB Go SDK предоставляет специальные функции для гарантированного выполнения пользовательской операции:

Выполнение повторных попыток при ошибках на объекте соединения `*sql.Conn`:

Для повторной обработки ошибок на объекте соединения `*sql.Conn` есть вспомогательная функция `retry.Do(ctx, db, op)`, которая предоставляет подготовленное соединение `*sql.Conn` для выполнения запросов.
В функцию `retry.Do` требуется передать контекст, объект базы данных, а также пользовательскую операцию, которую требуется выполнить.
Из клиентского кода можно влиять на логику выполнения повторных запросов через контекст и признак идемпотентности, а YDB Go SDK в свою очередь интерпретирует возвращаемые из `op` ошибки.

Пользовательская операция `op` должна возвращать ошибку или `nil`:

- в случае, если из пользовательской функции вернулся `nil`, то повторные запросы прекращаются;
- в случае, если из пользовательской функции вернулась ошибка, YDB Go SDK пытается идентифицировать эту ошибку и в зависимости от нее предпринимает повторные попытки.

Пример кода, использующего функцию `retry.Do`:

```
import (
    "context"
    "database/sql"
    "fmt"
    "log"

    "github.com/ydb-platform/ydb-go-sdk/v3/retry"
)

func main() {
    ...
    err = retry.Do(ctx, db, func(ctx context.Context, cc *sql.Conn) (err error) {
        row = cc.QueryRowContext(ctx, `
                PRAGMA TablePathPrefix("/local");
                DECLARE $seriesID AS Uint64;
                DECLARE $seasonID AS Uint64;
```

```
                DECLARE $episodeID AS Uint64;
                SELECT views FROM episodes WHERE series_id = $seriesID AND season_id = $seasonID AND epis
ode_id = $episodeID;
            `,
            sql.Named("seriesID", uint64(1)),
            sql.Named("seasonID", uint64(1)),
            sql.Named("episodeID", uint64(1)),
        )
        var views sql.NullFloat64
        if err = row.Scan(&views); err != nil {
            return fmt.Errorf("cannot scan views: %w", err)
        }
        if views.Valid {
            return fmt.Errorf("unexpected valid views: %v", views.Float64)
        }
        log.Printf("views = %v", views)
        return row.Err()
    }, retry.WithDoRetryOptions(retry.WithIdempotent(true)))
    if err != nil {
        log.Printf("retry.Do failed: %v\n", err)
    }
}
```

Выполнение повторных попыток при ошибках на объекте интерактивной транзакции `*sql.Tx`:

Для повторной обработки ошибок на объекте интерактивной транзакции `*sql.Tx` есть вспомогательная функция `retry.DoTx(ctx, db, op)`, которая предоставляет подготовленную транзакцию `*sql.Tx` для выполнения запросов.
В функцию `retry.DoTx` требуется передать контекст, объект базы данных, а также пользовательскую операцию, которую требуется выполнить.
В функцию приходит подготовленная транзакция `*sql.Tx`, на которой следует выполнять запросы к YDB.
Из клиентского кода можно влиять на логику выполнения повторных запросов через контекст и признак идемпотентности операции, а YDB Go SDK в свою очередь интерпретирует возвращаемые из `op` ошибки.

Пользовательская операция `op` должна возвращать ошибку или `nil`:

- в случае, если из пользовательской функции вернулся `nil`, то повторные запросы прекращаются;
- в случае, если из пользовательской функции вернулась ошибка, YDB Go SDK пытается идентифицировать эту ошибку и в зависимости от нее предпринимает повторные попытки.

Функция `retry.DoTx` использует режим изоляции read-write транзакции `sql.LevelDefault` по умолчанию, который можно изменить через опцию `retry.WithTxOptions`.

Пример кода, использующего функцию `retry.Do`:

```
import (
    "context"
    "database/sql"
    "fmt"
    "log"

    "github.com/ydb-platform/ydb-go-sdk/v3/retry"
)

func main() {
    ...
    err = retry.DoTx(ctx, db, func(ctx context.Context, tx *sql.Tx) error {
        row := tx.QueryRowContext(ctx,`
                PRAGMA TablePathPrefix("/local");
                DECLARE $seriesID AS Uint64;
                DECLARE $seasonID AS Uint64;
                DECLARE $episodeID AS Uint64;
                SELECT views FROM episodes WHERE series_id = $seriesID AND season_id = $seasonID AND epis
ode_id = $episodeID;
            `,
            sql.Named("seriesID", uint64(1)),
            sql.Named("seasonID", uint64(1)),
            sql.Named("episodeID", uint64(1)),
        )
        var views sql.NullFloat64
        if err = row.Scan(&views); err != nil {
            return fmt.Errorf("cannot select current views: %w", err)
        }
        if !views.Valid {
            return fmt.Errorf("unexpected invalid views: %v", views)
        }
        t.Logf("views = %v", views)
        if views.Float64 != 1 {
            return fmt.Errorf("unexpected views value: %v", views)
        }
        return nil
    }, retry.WithDoTxRetryOptions(retry.WithIdempotent(true)), retry.WithTxOptions(&sql.TxOptions{
        Isolation: sql.LevelSnapshot,
        ReadOnly:  true,
    }))
    if err != nil {
        log.Printf("do tx failed: %v\n", err)
```

```
    }
}
```

Java

Native SDK

В YDB Java SDK повторные попытки реализованы классом-хелпером `SessionRetryContext`. Он создаётся через `SessionRetryContext.create`, куда передаётся `SessionSupplier` — как правило `TableClient` или `QueryClient`. Какие ошибки считаются временными и требуют повтора, описано в Обработка ошибок.

Настройки повторов:

- `maxRetries(int)` — максимальное число повторов (без учёта первой попытки; по умолчанию `10`)
- `retryNotFound(boolean)` — повторять ли операции со статусом `NOT_FOUND` (по умолчанию `true`)
- `idempotent(boolean)` — идемпотентность операции; расширяет список повторяемых ошибок (по умолчанию `false`)

Методы запуска:

- `supplyStatus` — операция, возвращающая `Status` (DDL, `createTable` и т.п.)
- `supplyResult` — операция, возвращающая данные (`executeDataQuery`, `QueryReader.readFrom` и т.п.)

Повтор выполняется, если лямбда вернула retryable статус или выбросила `UnexpectedResultException` с таким статусом.

```
import tech.ydb.common.transaction.TxMode;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;

public class RetryExample {

    public static void main(String[] args) {
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString).build();
             QueryClient queryClient = QueryClient.newClient(transport).build()) {

            // Настройка политики повторов
            SessionRetryContext retryCtx = SessionRetryContext.create(queryClient)
                    .maxRetries(5)
                    .retryNotFound(true)
                    .idempotent(true)
                    .build();

            // supplyResult — запрос с автоматическими повторами
            QueryReader reader = retryCtx.supplyResult(session -> QueryReader.readFrom(
                    session.createQuery("SELECT 1 AS value", TxMode.NONE, Params.empty())
            )).join().getValue();

            ResultSetReader rs = reader.getResultSet(0);
            if (rs.next()) {
                System.out.println("SELECT 1 => " + rs.getColumn("value").getInt32());
            }

            // supplyStatus — операции без результата (DDL, createTable и т.п.)
            // retryCtx.supplyStatus(session -> session.executeSchemeQuery("CREATE TABLE ..."))
            //         .join().expectSuccess("DDL failed");
        }
    }
}
```

JDBC

`SessionRetryContext` относится к нативному API (`TableClient` / `QueryClient`). При работе через JDBC драйвер выполняет ограниченные встроенные повторы (например, при `BAD_SESSION` вне транзакции); для остальных временных сбоев реализуйте цикл повторов на уровне приложения. Классы `YdbRetryableException` и `YdbConditionallyRetryableException` помечают ошибки, которые имеет смысл повторить — см. Обработка ошибок. Подключение — в Инициализация драйвера.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.SQLTransientException;
import java.sql.Statement;

import tech.ydb.jdbc.exception.YdbConditionallyRetryableException;
import tech.ydb.jdbc.exception.YdbRetryableException;

public class JdbcRetryExample {

    private static final int MAX_RETRIES = 3;

    public static void main(String[] args) throws SQLException {
        String connectionUrl = System.getenv().getOrDefault(
```

```
                "YDB_JDBC_URL", "jdbc:ydb:grpc://localhost:2136/local");

        for (int attempt = 0; attempt <= MAX_RETRIES; attempt++) {
            try (Connection connection = DriverManager.getConnection(connectionUrl);
                 Statement statement = connection.createStatement();
                 ResultSet rs = statement.executeQuery("SELECT 1 AS value")) {
                rs.next();
                System.out.println("SELECT 1 => " + rs.getInt("value"));
                return;
            } catch (SQLException e) {
                if (attempt >= MAX_RETRIES || !isRetryable(e)) {
                    throw new RuntimeException("query failed after retries", e);
                }
                sleepBeforeRetry(attempt);
            }
        }
    }

    private static boolean isRetryable(SQLException e) {
        if (e instanceof YdbRetryableException
                || e instanceof YdbConditionallyRetryableException
                || e instanceof SQLTransientException) {
            return true;
        }
        return e.getCause() instanceof SQLException && isRetryable((SQLException) e.getCause());
    }

    private static void sleepBeforeRetry(int attempt) {
        try {
            Thread.sleep(50L * (attempt + 1));
        } catch (InterruptedException ie) {
            Thread.currentThread().interrupt();
            throw new RuntimeException(ie);
        }
    }
}
```

Python

Native SDK

В YDB Python SDK выполнение повторных попыток реализовано в `QuerySessionPool` с использованием класса `RetrySettings` для настройки параметров повторов. Класс `RetrySettings` поддерживает следующие опции:

- `max_retries` - максимальное количество повторных попыток (по умолчанию 10)
- `idempotent` - признак идемпотентности операции. Идемпотентные операции повторяются для более широкого списка ошибок (по умолчанию False)
- `backoff_ceiling`, `backoff_slot_duration` - параметры алгоритма экспоненциальной задержки
- `fast_backoff_settings`, `slow_backoff_settings` - настройки быстрых и медленных повторов

Для выполнения запросов с повторными попытками `QuerySessionPool` предоставляет методы `retry_operation_sync` и `execute_with_retries`. Метод `execute_with_retries` предназначен для разовых запросов с неявным режимом транзакции (implicit). Для остальных случаев (явные транзакции, несколько операций в одной транзакции) используйте `retry_operation_sync`.

Пример кода, использующего execute_with_retries:

```
import ydb

def execute_query(pool: ydb.QuerySessionPool):
    result_sets = pool.execute_with_retries(
        "SELECT series_id, title FROM series WHERE series_id = 1;",
        retry_settings=ydb.RetrySettings(idempotent=True),
    )
    # ...
```

Пример кода, использующего retry_operation_sync:

```
import ydb

def execute_query(pool: ydb.QuerySessionPool):
    def callee(session: ydb.QuerySession):
          with session.transaction().execute(
              "SELECT 1",
              commit_tx=True,
          ) as result_sets:
              pass

    result = pool.retry_operation_sync(
        callee,
        retry_settings=ydb.RetrySettings(max_retries=20, idempotent=True),
    )
    # ...
```

Native SDK (Asyncio)

Пример кода, использующего execute_with_retries:

```python
import ydb

async def execute_query(pool: ydb.aio.QuerySessionPool):
    result_sets = await pool.execute_with_retries(
        "SELECT series_id, title FROM series WHERE series_id = 1;",
        retry_settings=ydb.RetrySettings(idempotent=True),
    )
    # ...
```

Пример кода, использующего retry_operation_sync:

```python
import ydb

async def execute_query(pool: ydb.aio.QuerySessionPool):
    async def callee(session):
        async with session.transaction(tx_mode=ydb.QuerySerializableReadWrite()) as tx:
            async with await tx.execute("SELECT 1", commit_tx=True) as result_sets:
                pass

    await pool.retry_operation_async(
        callee,
        retry_settings=ydb.RetrySettings(max_retries=20, idempotent=True),
    )
    # ...
```

SQLAlchemy

При использовании YDB через SQLAlchemy выполнение повторных попыток происходит под капотом и не регулируется снаружи.

C#

В YDB C# SDK повторные попытки реализованы на двух уровнях.

OpenRetryableConnectionAsync

Метод `OpenRetryableConnectionAsync` создаёт соединение с автоматическими повторными попытками при transient-ошибках. Соединение, полученное таким образом, не поддерживает интерактивные транзакции — для работы с транзакциями используйте `ExecuteInTransactionAsync`.

```csharp
using Ydb.Sdk.Ado;

await using var dataSource = new YdbDataSource("Host=localhost;Port=2136;Database=/local");

await using var connection = await dataSource.OpenRetryableConnectionAsync();
var command = new YdbCommand("SELECT series_id, title FROM series WHERE series_id = $series_id", connection);
command.Parameters.Add(new YdbParameter("$series_id", YdbDbType.Uint64, 1U));

await using var reader = await command.ExecuteReaderAsync();
while (await reader.ReadAsync())
{
    Console.WriteLine($"series_id: {reader.GetUint64(0)}, title: {reader.GetString(1)}");
}
```

ExecuteInTransactionAsync

Метод `ExecuteInTransactionAsync` выполняет несколько операций в рамках одной транзакции с автоматическим повтором при конфликтах:

```csharp
using Ydb.Sdk.Ado;

await using var dataSource = new YdbDataSource("Host=localhost;Port=2136;Database=/local");

await dataSource.ExecuteInTransactionAsync(async connection =>
{
    var command = connection.CreateCommand();
    command.CommandText = "UPSERT INTO series (series_id, title) VALUES (1, \"IT Crowd\")";
    await command.ExecuteNonQueryAsync();
});
```

JavaScript

Повторные попытки и переподключения сделаны внутри sdk, отдельно ничего пользователю настраивать не нужно.

Сам ретраер доступен в отдельном пакете `@ydbjs/retry`.

```javascript
import { retry } from '@ydbjs/retry'

let attempts = 0
const result = retry({ retry: isError, budget: 3 }, async () => {
  if (attempts >= 2) {
    return 'success'
  }

  attempts++
```

```
  throw new Error('test error')
})
```

Rust

Повторные попытки для запросов через Query Service выполняет `QueryClient`: вспомогательные методы для выполнения одного транзакционного SQL-запроса (`query_row`, `exec` и т.д.) ретраятся автоматически; для нескольких операций в одной транзакции — `retry_tx`.

```
use ydb::{AccessTokenCredentials, ClientBuilder, YdbResult};

#[tokio::main]
async fn main() -> YdbResult<()> {
    let client = ClientBuilder::new_from_connection_string(
        "grpc://localhost:2136?database=local",
    )?
    .with_credentials(AccessTokenCredentials::from("..."))
    .client()?;

    client.wait().await?;

    let mut qc = client.query_client();

    // один SQL-запрос на query-клиенте: внутренние повторные попытки
    let mut row = qc
        .query_row("SELECT series_id, title FROM series WHERE series_id = 1")
        .idempotent(true)
        .await?;

    // несколько операций в одной транзакции с ретраями
    let title: String = qc
        .retry_tx(async |tx| {
            let mut row = tx
                .query_row("SELECT series_id, title FROM series WHERE series_id = 1")
                .await?;
            Ok(row.remove_field_by_name("title")?.try_into()?)
        })
        .idempotent(true)
        .await?;

    Ok(())
}
```

PHP

В YDB PHP SDK повторные попытки для запросов к Table API задаются через `Table::retryTransaction()` (транзакция + коммит + ретраи при поддерживаемых ошибках) или `Table::retrySession()` (одна сессия без обёртки «транзакция целиком»). Второй аргумент `retryTransaction` — признак идемпотентности (`true` расширяет набор ошибок, при которых выполняется повтор).

Пример с `retryTransaction`:

```
<?php

use YdbPlatform\Ydb\Session;
use YdbPlatform\Ydb\Ydb;

$ydb = new Ydb($config);

$result = $ydb->table()->retryTransaction(
    function (Session $session) {
        return $session->query(
            'SELECT series_id, title FROM series WHERE series_id = 1;'
        );
    },
    true
);

// $result->rows(), $result->rowCount(), ...
```

Установить размер пула сессий

YDB создаёт актор для каждой сессии. В результате, размер пула сессий на клиенте влияет на потребление ресурсов (память, процессор) на серверной стороне YDB.

Например, если 1000 клиентов одной базы данных открывают по 1000 сессий, то на серверной стороне создаётся 1000000 акторов. Такое количество акторов потребляет значительные объёмы памяти и ресурсов процессора. При отсутствии ограничения на число сессий на клиенте это может привести к медленной работе кластера и его полуаварийному состоянию.

По умолчанию в YDB SDK при использовании нативных драйверов установлен лимит в 50 сессий. При использовании сторонних библиотек, например, Go `database/sql`, лимит не задан.

Рекомендуется устанавливать лимит на количество сессий на клиенте в минимально необходимый для штатной работы клиентского приложения. Следует учитывать, что сессия однопоточная как на серверной, так и на клиентской стороне. Соответственно, если для расчётной нагрузки приложению требуется выполнять 1000 одновременных запросов (inflight) в YDB, то лимит следует установить на уровне 1000 сессий.

Важно различать расчётный RPS (requests per second, запросов в секунду) и inflight. В первом случае речь идёт об общем количестве запросов, выполняемых к YDB за 1 секунду. Например, при RPS = 10000 и средней задержке исполнения запроса (latency) в 100 мс достаточно установить лимит в 1000 сессий. Это означает, что каждая сессия за расчётную секунду выполнит в среднем 10 последовательных запросов.

Ниже приведены примеры кода установки лимита на пул сессий в разных YDB SDK.

C++

Native SDK

- `MaxActiveSessions` — максимальный размер пула (по умолчанию 50).
- `MinPoolSize` — минимальное число сессий (по умолчанию 10). SDK перестанет закрывать сессии по таймауту по достижении лимита, поэтому число не гарантированное.

```
#include <ydb-cpp-sdk/client/driver/driver.h>
#include <ydb-cpp-sdk/client/query/client.h>

NYdb::NQuery::TQueryClient CreateQueryClient(const NYdb::TDriver& driver) {
    NYdb::NQuery::TClientSettings settings;
    settings.SessionPoolSettings(
        NYdb::NQuery::TSessionPoolSettings()
            .MaxActiveSessions(500)
            .MinPoolSize(10));
    return NYdb::NQuery::TQueryClient(driver, settings);
}
```

userver

static config

```
ydb:
    databases:
        db:
            endpoint: grpc://localhost:2136
            database: /local
            max_pool_size: 500
            min_pool_size: 10
```

Код инициализации `ydb::YdbComponent`, получения `ydb::TableClient` и запуска `components::MinimalServerComponentList` — как в примере из init.md.

Go

Native SDK

```
package main

import (
  "context"

  "github.com/ydb-platform/ydb-go-sdk/v3"
)

func main() {
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithSessionPoolSizeLimit(500),
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
  ...
}
```

database/sql

Библиотека `database/sql` имеет свой пул соединений. Каждое соединение в `database/sql` соответствует конкретной сессии YDB. Управлением пулом соединений в `database/sql` осуществляется с помощью функций `sql.DB.SetMaxOpenConns` и `sql.DB.SetMaxIdleConns`. Подробнее об этом написано в документации `database/sql`.

Пример кода, использующего размер пула соединений `database/sql`:

```
package main

import (
  "context"
  "database/sql"

  _ "github.com/ydb-platform/ydb-go-sdk/v3"
)

func main() {
  db, err := sql.Open("ydb", os.Getenv("YDB_CONNECTION_STRING"))
  if err != nil {
    panic(err)
  }
  defer db.Close()
  db.SetMaxOpenConns(100)
  db.SetMaxIdleConns(100)
  db.SetConnMaxIdleTime(time.Second) // workaround for background keep-aliving of YDB sessions
  ...
}
```

Java

Native SDK

Размер пула задаётся при создании `TableClient` или `QueryClient`. Каждая сессия в пуле — отдельный актор на сервере, поэтому лимит следует выбирать по расчётному inflight, а не только по RPS (см. вводный раздел выше).

```
import tech.ydb.common.transaction.TxMode;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;

public class SessionPoolLimitExample {

    public static void main(String[] args) {
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString).build();
             QueryClient queryClient = QueryClient.newClient(transport)
                     // 10 — минимальное число активных сессий, удерживаемых в пуле при очистке
                     // 500 — максимальный размер пула сессий
                     .sessionPoolMinSize(10)
                     .sessionPoolMaxSize(500)
                     .build()) {

            SessionRetryContext retryCtx = SessionRetryContext.create(queryClient).build();

            QueryReader reader = retryCtx.supplyResult(session -> QueryReader.readFrom(
                    session.createQuery("SELECT 1 AS value", TxMode.NONE, Params.empty())
            )).join().getValue();

            ResultSetReader rs = reader.getResultSet(0);
            if (rs.next()) {
                System.out.println("Пул настроен, SELECT 1 = " + rs.getColumn("value").getInt32());
            }
        }
    }
}
```

JDBC

При работе с JDBC, как правило, используются внешние пулы соединений, такие как HikariCP. YDB JDBC-драйвер подстраивает внутренний пул сессий под число открытых JDBC-соединений, поэтому лимит задаётся на стороне внешнего пула. Свойства подключения — в документации JDBC-драйвера.

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import com.zaxxer.hikari.HikariConfig;
import com.zaxxer.hikari.HikariDataSource;

public class JdbcSessionPoolLimitExample {

    public static void main(String[] args) throws SQLException {
        String jdbcUrl = System.getenv().getOrDefault(
                "YDB_JDBC_URL", "jdbc:ydb:grpc://localhost:2136/local");

        HikariConfig config = new HikariConfig();
        config.setJdbcUrl(jdbcUrl);
        config.setDriverClassName("tech.ydb.jdbc.YdbDriver");
```

```
        config.setMaximumPoolSize(100); // максимум JDBC-соединений (= сессий YDB)

        try (HikariDataSource dataSource = new HikariDataSource(config);
             Connection connection = dataSource.getConnection();
             Statement statement = connection.createStatement();
             ResultSet rs = statement.executeQuery("SELECT 1 AS value")) {

            rs.next();
            System.out.println("HikariCP pool size = 100, SELECT 1 = " + rs.getInt("value"));
        }
    }
}
```

В Spring Boot те же параметры задаются через `spring.datasource.hikari.maximum-pool-size` (см. пример в вводном разделе).

Python

Native SDK

```
import os
import ydb

with ydb.Driver(
    connection_string=os.environ["YDB_CONNECTION_STRING"],
    credentials=ydb.credentials_from_env_variables(),
) as driver:
    driver.wait(timeout=5)
    with ydb.QuerySessionPool(driver, size=500) as pool:
        # ...
```

Native SDK (Asyncio)

```
import os
import ydb
import asyncio

async def ydb_init():
    async with ydb.aio.Driver(
        connection_string=os.environ["YDB_CONNECTION_STRING"],
        credentials=ydb.credentials_from_env_variables(),
    ) as driver:
        await driver.wait()
        async with ydb.aio.QuerySessionPool(driver, size=500) as pool:
            # ...

asyncio.run(ydb_init())
```

SQLAlchemy

Установка размера пула на данный момент не поддержана.

C#

В YDB C# SDK параметры пула сессий задаются через строку подключения:

```
using Ydb.Sdk.Ado;

await using var dataSource = new YdbDataSource(
    "Host=localhost;Port=2136;Database=/local;MaxPoolSize=500;MinPoolSize=10;SessionIdleTimeout=60");
```

- `MaxPoolSize` — максимальный размер пула сессий (по умолчанию 100)
- `MinPoolSize` — минимальное число сессий, удерживаемых в пуле (по умолчанию 0)
- `SessionIdleTimeout` — время простоя сессии в секундах до её закрытия (по умолчанию 300)

Для Entity Framework и linq2db используйте тот же connectionString.

JavaScript

Функциональность на данный момент не поддерживается.

Rust

Размер пула сессий задаётся на драйвере через `Client::with_session_pool` и `SessionPoolSettings::with_limit`. Пул общий для table- и query-клиентов:

```
use ydb::{ClientBuilder, SessionPoolSettings, YdbResult};

#[tokio::main]
async fn main() -> YdbResult<()> {
    let client = ClientBuilder::new_from_connection_string(
        std::env::var("YDB_CONNECTION_STRING")?,
    )?
    .client()?;
    client.wait().await?;

    let client = client
        .with_session_pool(SessionPoolSettings::new().with_limit(500))
```

```
        .await?;

    let mut qc = client.query_client();
    let mut row = qc.query_row("SELECT 1 AS one").await?;
    Ok(())
}
```

PHP

Функциональность на данный момент не поддерживается.

Вставка данных

Ниже приведены примеры кода использования встроенных в YDB SDK средств выполнения вставки:

C++

Native SDK

```
#include <ydb-cpp-sdk/client/query/client.h>

void UpsertSeries(NYdb::NQuery::TQueryClient& client) {
  NYdb::NStatusHelpers::ThrowOnError(client.RetryQuerySync(
      [](NYdb::NQuery::TSession session) {
          constexpr auto query = R"(
              DECLARE $seriesData AS List<Struct<
                  series_id: Uint64,
                  title: Utf8,
                  series_info: Utf8,
                  comment: Optional<Utf8>
              >>;

              UPSERT INTO series
              (
                  series_id,
                  title,
                  series_info,
                  comment
              )
              SELECT
                  series_id,
                  title,
                  series_info,
                  comment
              FROM AS_TABLE($seriesData);
          )";

          auto params = NYdb::TParamsBuilder()
              .AddParam("$seriesData")
                  .BeginList()
                  .AddListItem()
                      .BeginStruct()
                      .AddMember("series_id").Uint64(1)
                      .AddMember("title").Utf8("IT Crowd")
                      .AddMember("series_info").Utf8(
                          "The IT Crowd is a British sitcom produced by Channel 4, written by Graham Linehan,
produced by "
                          "Ash Atalla and starring Chris O'Dowd, Richard Ayoade, Katherine Parkinson, and Mat
t Berry.")
                      .AddMember("comment").OptionalUtf8(std::nullopt)
                      .EndStruct()
                  .AddListItem()
                      .BeginStruct()
                      .AddMember("series_id").Uint64(2)
                      .AddMember("title").Utf8("Silicon Valley")
                      .AddMember("series_info").Utf8(
                          "Silicon Valley is an American comedy television series created by Mike Judge, John
Altschuler and "
                          "Dave Krinsky. The series focuses on five young men who founded a startup company i
n Silicon Valley.")
                      .AddMember("comment").OptionalUtf8("lorem ipsum")
                      .EndStruct()
                  .EndList()
                  .Build()
              .Build();

          return session.ExecuteQuery(
              query,
              NYdb::NQuery::TTxControl::BeginTx(NYdb::NQuery::TTxSettings::SerializableRW()).CommitTx(),
              params).GetValueSync();
      },
      NYdb::NQuery::TRetryOperationSettings()
          .Idempotent(true)
  ));
}
```

userver

```
#include <userver/ydb/io/supported_types.hpp>
#include <userver/ydb/table.hpp>

struct SeriesData final {
    std::uint64_t series_id;
    ydb::Utf8 title;
    ydb::Utf8 series_info;
    std::optional<ydb::Utf8> comment;
};

void UpsertSeries(ydb::TableClient& client) {
    auto builder = client.GetBuilder();
    builder.Add(
```

```
        "$seriesData",
        std::vector<SeriesData>{
            {
                .series_id = 1,
                .title = ydb::Utf8{"IT Crowd"},
                .series_info = ydb::Utf8{
                    "The IT Crowd is a British sitcom produced by Channel 4, written by Graham Linehan, produ
ced by "
                    "Ash Atalla and starring Chris O'Dowd, Richard Ayoade, Katherine Parkinson, and Matt Berr
y."
                },
                .comment = std::nullopt,
            },
            {
                .series_id = 2,
                .title = ydb::Utf8{"Silicon Valley"},
                .series_info = ydb::Utf8{
                    "Silicon Valley is an American comedy television series created by Mike Judge, John Altsc
huler and "
                    "Dave Krinsky. The series focuses on five young men who founded a startup company in Sili
con Valley."
                },
                .comment = ydb::Utf8{"lorem ipsum"},
            },
        }
    );

    client.ExecuteQuery(
        ydb::OperationSettings{
            .tx_mode = ydb::TransactionMode::kSerializableRW,
            .is_idempotent = true,
        },
        ydb::Query{R"(
            DECLARE $seriesData AS List<Struct<
                series_id: Uint64,
                title: Utf8,
                series_info: Utf8,
                comment: Optional<Utf8>
            >>;

            UPSERT INTO series
            (
                series_id,
                title,
                series_info,
                comment
            )
            SELECT
                series_id,
                title,
                series_info,
                comment
            FROM AS_TABLE($seriesData);
        )"},
        std::move(builder)
    );
}
```

Go

Native SDK

```
package main

import (
  "context"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/table"
  "github.com/ydb-platform/ydb-go-sdk/v3/table/types"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithAccessTokenCredentials(os.Getenv("YDB_TOKEN")),
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
  // execute upsert with native ydb data
  err = db.Table().DoTx( // Do retry operation on errors with best effort
    ctx, // context manages exiting from Do
    func(ctx context.Context, tx table.TransactionActor) (err error) { // retry operation
      res, err := tx.Execute(ctx, `
          PRAGMA TablePathPrefix("/path/to/table");
          DECLARE $seriesID AS Uint64;
```

```
          DECLARE $seasonID AS Uint64;
          DECLARE $episodeID AS Uint64;
          DECLARE $views AS Uint64;
          UPSERT INTO episodes ( series_id, season_id, episode_id, views )
          VALUES ( $seriesID, $seasonID, $episodeID, $views );
        `,
        table.NewQueryParameters(
          table.ValueParam("$seriesID", types.Uint64Value(1)),
          table.ValueParam("$seasonID", types.Uint64Value(1)),
          table.ValueParam("$episodeID", types.Uint64Value(1)),
          table.ValueParam("$views", types.Uint64Value(1)), // increment views
        ),
      )
      if err != nil {
        return err
      }
      if err = res.Err(); err != nil {
        return err
      }
      return res.Close()
    }, table.WithIdempotent(),
  )
  if err != nil {
    fmt.Printf("unexpected error: %v", err)
  }
}
```

database/sql

```
package main

import (
  "context"
  "database/sql"
  "os"

  _ "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/retry"
  "github.com/ydb-platform/ydb-go-sdk/v3/types"
)

func main() {
  db, err := sql.Open("ydb", os.Getenv("YDB_CONNECTION_STRING"))
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
  // execute upsert with native ydb data
  err = retry.DoTx(ctx, db, func(ctx context.Context, tx *sql.Tx) error {
    if _, err = tx.ExecContext(ctx,`
        PRAGMA TablePathPrefix("/local");
        REPLACE INTO series
        SELECT
          series_id,
          title,
          series_info,
          comment
        FROM AS_TABLE($seriesData);
      `,
      sql.Named("seriesData", types.ListValue(
        types.StructValue(
          types.StructFieldValue("series_id", types.Uint64Value(1)),
          types.StructFieldValue("title", types.TextValue("IT Crowd")),
          types.StructFieldValue("series_info", types.TextValue(
            "The IT Crowd is a British sitcom produced by Channel 4, written by Graham Linehan, produced by
"+
            "Ash Atalla and starring Chris O'Dowd, Richard Ayoade, Katherine Parkinson, and Matt Berry.",
          )),
          types.StructFieldValue("comment", types.NullValue(types.TypeText)),
        ),
        types.StructValue(
          types.StructFieldValue("series_id", types.Uint64Value(2)),
          types.StructFieldValue("title", types.TextValue("Silicon Valley")),
          types.StructFieldValue("series_info", types.TextValue(
            "Silicon Valley is an American comedy television series created by Mike Judge, John Altschuler an
d "+
            "Dave Krinsky. The series focuses on five young men who founded a startup company in Silicon Vall
ey.",
          )),
          types.StructFieldValue("comment", types.TextValue("lorem ipsum")),
        ),
      )),
    ); err != nil {
      return err
    }
    return nil
  }, retry.WithDoTxRetryOptions(retry.WithIdempotent(true)))
  if err != nil {
    fmt.Printf("unexpected error: %v", err)
```

```
  }
}
```

Java

Операция UPSERT INTO вставляет строку или обновляет существующую по первичному ключу без предварительного чтения. Для массовой неатомарной загрузки см. пакетную вставку. Структура таблицы описана в разделе Таблицы.

Native SDK

```
import java.util.ArrayList;
import java.util.List;

import tech.ydb.common.transaction.TxMode;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;
import tech.ydb.table.values.ListType;
import tech.ydb.table.values.ListValue;
import tech.ydb.table.values.OptionalType;
import tech.ydb.table.values.PrimitiveType;
import tech.ydb.table.values.PrimitiveValue;
import tech.ydb.table.values.StructType;
import tech.ydb.table.values.Value;

public class UpsertExample {
    public static void main(String[] args) {
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString).build();
             QueryClient queryClient = QueryClient.newClient(transport).build()) {

            SessionRetryContext retryCtx = SessionRetryContext.create(queryClient).build();

            // Создаём таблицу, если её ещё нет (DDL — без транзакции)
            retryCtx.supplyResult(session -> QueryReader.readFrom(session.createQuery("""
                    CREATE TABLE IF NOT EXISTS series (
                        series_id Uint64,
                        title Text,
                        series_info Text,
                        comment Text,
                        PRIMARY KEY (series_id)
                    );
                    """, TxMode.NONE, Params.empty())
            )).join().getValue();

            // Подготовка данных для UPSERT через AS_TABLE($seriesData)
            StructType rowType = StructType.of(
                    "series_id", PrimitiveType.Uint64,
                    "title", PrimitiveType.Text,
                    "series_info", PrimitiveType.Text,
                    "comment", OptionalType.of(PrimitiveType.Text)
            );

            List<Value<?>> rows = new ArrayList<>();
            rows.add(rowType.newValue(
                    "series_id", PrimitiveValue.newUint64(1),
                    "title", PrimitiveValue.newText("IT Crowd"),
                    "series_info", PrimitiveValue.newText(
                            "The IT Crowd is a British sitcom produced by Channel 4..."),
                    "comment", OptionalType.of(PrimitiveType.Text).emptyValue()
            ));
            rows.add(rowType.newValue(
                    "series_id", PrimitiveValue.newUint64(2),
                    "title", PrimitiveValue.newText("Silicon Valley"),
                    "series_info", PrimitiveValue.newText(
                            "Silicon Valley is an American comedy television series..."),
                    "comment", OptionalType.of(PrimitiveType.Text).newValue(
                            PrimitiveValue.newText("lorem ipsum"))
            ));

            ListValue seriesData = ListType.of(rowType).newValue(rows);
            Params params = Params.of("$seriesData", seriesData);

            String upsertQuery = """
                    DECLARE $seriesData AS List<Struct<
                        series_id: Uint64,
                        title: Utf8,
                        series_info: Utf8,
                        comment: Optional<Utf8>
                    >>;
                    UPSERT INTO series (series_id, title, series_info, comment)
                    SELECT series_id, title, series_info, comment FROM AS_TABLE($seriesData);
                    """;

            retryCtx.supplyResult(session -> QueryReader.readFrom(
                    session.createQuery(upsertQuery, TxMode.SERIALIZABLE_RW, params)
```

```
            )).join().getValue();

            // Проверка: должно быть 2 строки
            QueryReader countReader = retryCtx.supplyResult(session -> QueryReader.readFrom(
                    session.createQuery("SELECT COUNT(*) AS cnt FROM series", TxMode.NONE, Params.empty())
            )).join().getValue();

            ResultSetReader rs = countReader.getResultSet(0);
            if (rs.next()) {
                System.out.println("Строк в таблице series: " + rs.getColumn("cnt").getUint64());
            }
        }
    }
}
```

JDBC

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class JdbcUpsertExample {
    public static void main(String[] args) throws SQLException {
        String url = System.getenv().getOrDefault(
                "YDB_JDBC_URL", "jdbc:ydb:grpc://localhost:2136/local");

        try (Connection conn = DriverManager.getConnection(url)) {
            // Создаём таблицу (DDL выполняется в режиме автокоммита)
            try (Statement stmt = conn.createStatement()) {
                stmt.execute("""
                        CREATE TABLE IF NOT EXISTS series (
                            series_id Uint64,
                            title Text,
                            series_info Text,
                            comment Text,
                            PRIMARY KEY (series_id)
                        );
                        """);
            }

            // REPLACE INTO — аналог UPSERT с полной перезаписью строки
            String replaceSql = """
                    REPLACE INTO series (series_id, title, series_info, comment)
                    VALUES (?, ?, ?, ?);
                    """;

            try (PreparedStatement ps = conn.prepareStatement(replaceSql)) {
                ps.setLong(1, 1);
                ps.setString(2, "IT Crowd");
                ps.setString(3, "The IT Crowd is a British sitcom produced by Channel 4...");
                ps.setNull(4, java.sql.Types.VARCHAR);
                ps.executeUpdate();

                ps.setLong(1, 2);
                ps.setString(2, "Silicon Valley");
                ps.setString(3, "Silicon Valley is an American comedy television series...");
                ps.setString(4, "lorem ipsum");
                ps.executeUpdate();
            }

            // Проверка количества строк
            try (Statement stmt = conn.createStatement();
                 ResultSet rs = stmt.executeQuery("SELECT COUNT(*) AS cnt FROM series")) {
                if (rs.next()) {
                    System.out.println("Строк в таблице series: " + rs.getLong("cnt"));
                }
            }
        }
    }
}
```

В Spring Boot, Hibernate, JOOQ и других фреймворках поверх JDBC драйвер также стремится оптимизировать **крупные** последовательности вставок и изменений: при необходимости **UPSERT**, `INSERT`, `UPDATE`, `DELETE` автоматически **группируются в пакеты** на стороне драйвера (в том числе при больших батчах из ORM).

Python

Native SDK

Для вставки данных используется `QuerySessionPool` и метод `execute_with_retries` с параметризованным YQL-запросом. Запрос оперирует контейнерным типом `List<Struct<...>>`, что позволяет передавать несколько строк за один вызов.

```
import os
import ydb

with ydb.Driver(
    connection_string=os.environ["YDB_CONNECTION_STRING"],
```

```
    credentials=ydb.credentials_from_env_variables(),
) as driver:
    driver.wait(timeout=5)
    pool = ydb.QuerySessionPool(driver)

    series_struct_type = ydb.StructType()
    series_struct_type.add_member("series_id", ydb.PrimitiveType.Uint64)
    series_struct_type.add_member("title", ydb.PrimitiveType.Utf8)
    series_struct_type.add_member("series_info", ydb.PrimitiveType.Utf8)
    series_struct_type.add_member("comment", ydb.OptionalType(ydb.PrimitiveType.Utf8))

    series_data = [
        {
            "series_id": 1,
            "title": "IT Crowd",
            "series_info": "The IT Crowd is a British sitcom...",
            "comment": None,
        },
        {
            "series_id": 2,
            "title": "Silicon Valley",
            "series_info": "Silicon Valley is an American comedy...",
            "comment": "lorem ipsum",
        },
    ]

    pool.execute_with_retries(
        """
        DECLARE $seriesData AS List<Struct<
            series_id: Uint64,
            title: Utf8,
            series_info: Utf8,
            comment: Optional<Utf8>
        >>;

        UPSERT INTO series
        (
            series_id,
            title,
            series_info,
            comment
        )
        SELECT
            series_id,
            title,
            series_info,
            comment
        FROM AS_TABLE($seriesData);
        """,
        {"$seriesData": (series_data, ydb.ListType(series_struct_type))},
        retry_settings=ydb.RetrySettings(idempotent=True),
    )
```

Native SDK (Asyncio)

```
import os
import ydb
import asyncio

async def ydb_init():
    async with ydb.aio.Driver(
        connection_string=os.environ["YDB_CONNECTION_STRING"],
        credentials=ydb.credentials_from_env_variables(),
    ) as driver:
        await driver.wait()
        pool = ydb.aio.QuerySessionPool(driver)

        series_struct_type = ydb.StructType()
        series_struct_type.add_member("series_id", ydb.PrimitiveType.Uint64)
        series_struct_type.add_member("title", ydb.PrimitiveType.Utf8)
        series_struct_type.add_member("series_info", ydb.PrimitiveType.Utf8)
        series_struct_type.add_member("comment", ydb.OptionalType(ydb.PrimitiveType.Utf8))

        series_data = [
            {"series_id": 1, "title": "IT Crowd", "series_info": "The IT Crowd is a British sitcom...", "comm
ent": None},
            {"series_id": 2, "title": "Silicon Valley", "series_info": "Silicon Valley is an American comed
y...", "comment": "lorem ipsum"},
        ]

        await pool.execute_with_retries(
            """
            DECLARE $seriesData AS List<Struct<
                series_id: Uint64,
                title: Utf8,
                series_info: Utf8,
                comment: Optional<Utf8>
            >>;

            UPSERT INTO series (series_id, title, series_info, comment)
            SELECT series_id, title, series_info, comment FROM AS_TABLE($seriesData);
```

```
            """,
            {"$seriesData": (series_data, ydb.ListType(series_struct_type))},
            retry_settings=ydb.RetrySettings(idempotent=True),
        )

asyncio.run(ydb_init())
```

SQLAlchemy

При использовании YDB через SQLAlchemy для вставки данных используется функция `ydb_sqlalchemy.upsert`, которая формирует запрос `UPSERT INTO` на основе таблицы и переданных значений. Можно вставлять как одну строку, так и несколько строк за один вызов:

```
import os
import sqlalchemy as sa
from sqlalchemy import Column, Integer, MetaData, String, Table
import ydb_sqlalchemy as ydb_sa

engine = sa.create_engine(os.environ["YDB_SQLALCHEMY_URL"])

series = Table(
    "series",
    MetaData(),
    Column("series_id", Integer, primary_key=True),
    Column("title", String),
    Column("series_info", String),
    Column("comment", String, nullable=True),
)

with engine.connect() as connection:
    stmt = ydb_sa.upsert(series).values(
        [
            {
                "series_id": 1,
                "title": "IT Crowd",
                "series_info": "The IT Crowd is a British sitcom...",
                "comment": None,
            },
            {
                "series_id": 2,
                "title": "Silicon Valley",
                "series_info": "Silicon Valley is an American comedy...",
                "comment": "lorem ipsum",
            },
        ]
    )
    connection.execute(stmt)
    connection.commit()
```

C#

```
using Ydb.Sdk.Ado;
using Ydb.Sdk.Ado.YdbType;

await using var dataSource = new YdbDataSource("Host=localhost;Port=2136;Database=/local");
await using var connection = await dataSource.OpenRetryableConnectionAsync();

var seriesData = new List<YdbStruct>
{
    new()
    {
        { "series_id", 1UL, YdbDbType.Uint64 },
        { "title", "IT Crowd", YdbDbType.Text },
        { "series_info", "The IT Crowd is a British sitcom produced by Channel 4.", YdbDbType.Text },
        { "comment", null, YdbDbType.Text },
    },
    new()
    {
        { "series_id", 2UL, YdbDbType.Uint64 },
        { "title", "Silicon Valley", YdbDbType.Text },
        { "series_info", "Silicon Valley is an American comedy television series.", YdbDbType.Text },
        { "comment", "lorem ipsum", YdbDbType.Text },
    },
};

var command = new YdbCommand("UPSERT INTO series SELECT * FROM AS_TABLE($series_data)", connection);
command.Parameters.Add(new YdbParameter("$series_data", seriesData));

await command.ExecuteNonQueryAsync();
```

JavaScript

```
import { Driver } from '@ydbjs/core'
import { query } from '@ydbjs/query'

const driver = new Driver('grpc://localhost:2136/local')
await driver.ready()
```

```
const users = [
  { id: 1, name: 'Alice' },
  { id: 2, name: 'Bob' },
]

const sql = query(driver)
await sql`UPSERT INTO users SELECT * FROM AS_TABLE(${users})`
```

Rust

```
use ydb::{
    ydb_params, ydb_struct, AccessTokenCredentials, ClientBuilder, Value, YdbResult,
};

fn series_row(
    series_id: u64,
    title: &str,
    series_info: &str,
    comment: Option<&str>,
) -> YdbResult<Value> {
    let comment_val = match comment {
        None => Value::optional_from(Value::Text(String::new()), None)?,
        Some(s) => Value::optional_from(
            Value::Text(String::new()),
            Some(Value::Text(s.into())),
        )?,
    };
    Ok(ydb_struct!(
        "series_id" => series_id,
        "title" => title,
        "series_info" => series_info,
        "comment" => comment_val,
    ))
}

#[tokio::main]
async fn main() -> YdbResult<()> {
    let client = ClientBuilder::new_from_connection_string(
        "grpc://localhost:2136?database=local",
    )?
    .with_credentials(AccessTokenCredentials::from("..."))
    .client()?;

    client.wait().await?;

    let example = series_row(0, "", "", None)?;
    let series_data = Value::list_from(
        example,
        vec![
            series_row(
                1,
                "IT Crowd",
                "The IT Crowd is a British sitcom...",
                None,
            )?,
            series_row(
                2,
                "Silicon Valley",
                "Silicon Valley is an American comedy...",
                Some("lorem ipsum"),
            )?,
        ],
    )?;

    client
        .query_client()
        .exec(
            r#"
            PRAGMA TablePathPrefix("/local");
            UPSERT INTO series
            (
                series_id,
                title,
                series_info,
                comment
            )
            SELECT
                series_id,
                title,
                series_info,
                comment
            FROM AS_TABLE($seriesData);
            "#,
        )
        .params(ydb_params!("$seriesData" => series_data))
        .await?;

    Ok(())
}
```

PHP

```php
<?php

use YdbPlatform\Ydb\Session;
use YdbPlatform\Ydb\Ydb;

$ydb = new Ydb($config);

$yql = <<<'EOS'
PRAGMA TablePathPrefix("/local");
DECLARE $seriesData AS List<Struct<
    series_id: Uint64,
    title: Utf8,
    series_info: Utf8,
    comment: Optional<Utf8>
>>;

UPSERT INTO series
(
    series_id,
    title,
    series_info,
    comment
)
SELECT
    series_id,
    title,
    series_info,
    comment
FROM AS_TABLE($seriesData);
EOS;

$seriesData = [
    [
        'series_id' => 1,
        'title' => 'IT Crowd',
        'series_info' => 'The IT Crowd is a British sitcom...',
        'comment' => null,
    ],
    [
        'series_id' => 2,
        'title' => 'Silicon Valley',
        'series_info' => 'Silicon Valley is an American comedy...',
        'comment' => 'lorem ipsum',
    ],
];

$ydb->table()->retryTransaction(
    function (Session $session) use ($yql, $seriesData) {
        return $session->prepare($yql)->execute([
            'seriesData' => $seriesData,
        ]);
    },
    true
);
```

Пакетная вставка данных

YDB поддерживает пакетную вставку большого количества строк без гарантий атомарности. Запись данных разбивается на несколько независимых транзакций, каждая их которых затрагивает единственную партицию, с параллельным исполнением. За счет этого такой подход более эффективен чем `YQL`. В случае успеха метод `BulkUpsert` гарантирует вставку всех данных, переданных в рамках данного запроса.

> **Важно**
>
> При использовании `BulkUpsert` для вставки данных в колоночные таблицы необходимо передавать значения **всех** колонок, включая `NULL`-значения.

Ниже приведены примеры кода использования встроенных в YDB SDK средств выполнения пакетной вставки:

C++

Native SDK

```
#include <ydb-cpp-sdk/client/table/table.h>

void BulkUpsertLogs(const NYdb::TDriver& driver) {
  NYdb::NTable::TTableClient client(driver);

  constexpr int kBatchSize = 1000;
  NYdb::TValueBuilder rowsBuilder;
  rowsBuilder.BeginList();
  for (int i = 0; i < kBatchSize; ++i) {
      rowsBuilder.AddListItem()
          .BeginStruct()
          .AddMember("App").Utf8("App_" + std::to_string(i / 256))
          .AddMember("Host").Utf8("192.168.0." + std::to_string(i % 256))
          .AddMember("Timestamp")
              .Timestamp(TInstant::Now() + TDuration::Seconds(i))
          .AddMember("HttpCode").Uint32(static_cast<uint32_t>(i % 113 == 0 ? 404 : 200))
          .AddMember("Message")
              .Utf8(i % 3 == 0 ? "GET / HTTP/1.1" : "GET /images/logo.png HTTP/1.1")
          .EndStruct();
  }
  rowsBuilder.EndList();

  NYdb::TValue rows = rowsBuilder.Build();

  NYdb::NStatusHelpers::ThrowOnError(client.RetryOperationSync(
      [&rows](NYdb::NTable::TTableClient& client) {
          return client.BulkUpsert("/local/bulk_upsert_example", NYdb::TValue{rows}).GetValueSync();
      },
      NYdb::NTable::TRetryOperationSettings()
          .Idempotent(true)
  ));
}
```

userver

```
#include <userver/ydb/io/supported_types.hpp>
#include <userver/ydb/table.hpp>

struct LogMessage final {
    ydb::Utf8 App;
    ydb::Utf8 Host;
    std::chrono::system_clock::time_point Timestamp;
    std::uint32_t HttpCode;
    ydb::Utf8 Message;
};

void BulkUpsertLogs(ydb::TableClient& client) {
    constexpr int kBatchSize = 1000;
    std::vector<LogMessage> rows;
    rows.reserve(kBatchSize);

    for (int i = 0; i < kBatchSize; ++i) {
        rows.push_back({
            .App = ydb::Utf8{"App_" + std::to_string(i / 256)},
            .Host = ydb::Utf8{"192.168.0." + std::to_string(i % 256)},
            .Timestamp = std::chrono::system_clock::now() + std::chrono::seconds{i},
            .HttpCode = static_cast<std::uint32_t>(i % 113 == 0 ? 404 : 200),
            .Message = ydb::Utf8{
                i % 3 == 0 ? "GET / HTTP/1.1" : "GET /images/logo.png HTTP/1.1"
            },
        });
    }

    client.BulkUpsert(
        "/local/bulk_upsert_example",
        rows,
        ydb::OperationSettings{
            .is_idempotent = true,
        }
```

```
    );
}
```

Go

Native SDK

Пакетная вставка нативных YDB данных

```
package main

import (
  "context"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/table"
  "github.com/ydb-platform/ydb-go-sdk/v3/table/types"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithAccessTokenCredentials(os.Getenv("YDB_TOKEN")),
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
  type logMessage struct {
    App       string
    Host      string
    Timestamp time.Time
    HTTPCode  uint32
    Message   string
  }
  // prepare native go data
  const batchSize = 10000
  logs := make([]logMessage, 0, batchSize)
  for i := 0; i < batchSize; i++ {
    message := logMessage{
      App:       fmt.Sprintf("App_%d", i/256),
      Host:      fmt.Sprintf("192.168.0.%d", i%256),
      Timestamp: time.Now().Add(time.Millisecond * time.Duration(i%1000)),
      HTTPCode:  200,
    }
    if i%2 == 0 {
      message.Message = "GET / HTTP/1.1"
    } else {
      message.Message = "GET /images/logo.png HTTP/1.1"
    }
    logs = append(logs, message)
  }
  // execute bulk upsert with native ydb data
  err = db.Table().Do( // Do retry operation on errors with best effort
    ctx, // context manage exiting from Do
    func(ctx context.Context, s table.Session) (err error) { // retry operation
      rows := make([]types.Value, 0, len(logs))
      for _, msg := range logs {
        rows = append(rows, types.StructValue(
          types.StructFieldValue("App", types.UTF8Value(msg.App)),
          types.StructFieldValue("Host", types.UTF8Value(msg.Host)),
          types.StructFieldValue("Timestamp", types.TimestampValueFromTime(msg.Timestamp)),
          types.StructFieldValue("HTTPCode", types.Uint32Value(msg.HTTPCode)),
          types.StructFieldValue("Message", types.UTF8Value(msg.Message)),
        ))
      }
      return s.BulkUpsert(ctx, "/local/bulk_upsert_example", types.ListValue(rows...))
    },
  )
  if err != nil {
    fmt.Printf("unexpected error: %v", err)
  }
}
```

Пакетная вставка `CSV`

```
package main

import (
  "context"
  "fmt"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/table"
)
```

```
func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()

  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithAccessTokenCredentials(os.Getenv("YDB_TOKEN")),
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)

  csv := `skip row

id,val
42,"text42"
43,"text43"
44,hello
`

  err = db.Table().BulkUpsert(ctx, "/local/bulk_upsert_example", table.BulkUpsertDataCsv(
    []byte(csv),
    table.WithCsvHeader(),
    table.WithCsvSkipRows(2),
    table.WithCsvNullValue([]byte("hello")), // строка "hello" будет восприниматься как NULL
  ))
  if err != nil {
    fmt.Printf("unexpected error: %v", err)
  }
}
```

Пакетная вставка `Apache Arrow`

В следующем примере для подготовки данных используется пакет arrow.

```
package main

import (
  "bytes"
  "context"
  "fmt"

  "github.com/apache/arrow-go/v18/arrow"
  "github.com/apache/arrow-go/v18/arrow/array"
  "github.com/apache/arrow-go/v18/arrow/ipc"
  "github.com/apache/arrow-go/v18/arrow/memory"
  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/table"
)

func main() {
  ctx := context.Background()
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithAccessTokenCredentials(os.Getenv("YDB_TOKEN")),
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx) // cleanup resources

  mem := memory.NewGoAllocator()

  schema := arrow.NewSchema([]arrow.Field{
    {Name: "id", Type: arrow.PrimitiveTypes.Int64},
    {Name: "val", Type: arrow.BinaryTypes.String},
  }, nil)

  b := array.NewRecordBuilder(mem, schema)
  defer b.Release()

  b.Field(0).(*array.Int64Builder).AppendValues(
    []int64{123, 234}, nil)

  b.Field(1).(*array.StringBuilder).AppendValues(
    []string{"data1", "data2"}, nil)

  rec := b.NewRecordBatch()
  defer rec.Release()

  schemaPayload := ipc.GetSchemaPayload(rec.Schema(), mem)
  defer schemaPayload.Release()

  dataPayload, err := ipc.GetRecordBatchPayload(rec)
  if err != nil {
    panic(err)
  }
  defer dataPayload.Release()
```

```
  var schemaBuf bytes.Buffer
  _, err = schemaPayload.WritePayload(&schemaBuf)
  if err != nil {
    panic(err)
  }

  var dataBuf bytes.Buffer
  _, err = dataPayload.WritePayload(&dataBuf)
  if err != nil {
    panic(err)
  }

  err = db.Table().BulkUpsert(ctx, "/local/bulk_upsert_example", table.BulkUpsertDataArrow(
    dataBuf.Bytes(),
    table.WithArrowSchema(schemaBuf.Bytes()), // schema is required
  ))
  if err != nil {
    fmt.Printf("unexpected error: %v", err)
  }
}
```

database/sql

Реализация `database/sql` драйвера для YDB не поддерживает нетранзакционную пакетную вставку данных.
Для пакетной вставки следует пользоваться транзакционной вставкой.

Java

Пакетная вставка через `BulkUpsert` эффективнее транзакционного YQL для больших объёмов данных. Для небольших наборов строк см. UPSERT. Структура таблицы описана в разделе Таблицы.

Native SDK

```
import java.time.Instant;
import java.util.ArrayList;
import java.util.List;

import tech.ydb.auth.NopAuthProvider;
import tech.ydb.common.transaction.TxMode;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.TableClient;
import tech.ydb.table.query.Params;
import tech.ydb.table.settings.BulkUpsertSettings;
import tech.ydb.table.values.ListType;
import tech.ydb.table.values.ListValue;
import tech.ydb.table.values.PrimitiveType;
import tech.ydb.table.values.PrimitiveValue;
import tech.ydb.table.values.StructType;
import tech.ydb.table.values.Value;

public class BulkUpsertExample {
    private static final String TABLE_NAME = "bulk_upsert";
    private static final int BATCH_SIZE = 1000;

    public static void main(String[] args) {
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString)
                .withAuthProvider(NopAuthProvider.INSTANCE)
                .build();
             QueryClient queryClient = QueryClient.newClient(transport).build();
             TableClient tableClient = TableClient.newClient(transport).build()) {

            SessionRetryContext queryRetry = SessionRetryContext.create(queryClient).build();
            SessionRetryContext tableRetry = SessionRetryContext.create(tableClient).build();

            // Создаём таблицу для пакетной вставки
            queryRetry.supplyResult(session -> QueryReader.readFrom(session.createQuery("""
                    CREATE TABLE IF NOT EXISTS bulk_upsert (
                        app Text,
                        timestamp Timestamp,
                        host Text,
                        http_code Uint32,
                        message Text,
                        PRIMARY KEY (app, timestamp, host)
                    );
                    """, TxMode.NONE, Params.empty())
            )).join().getValue();

            // Полный путь к таблице: /local/bulk_upsert
            String tablePath = transport.getDatabase() + "/" + TABLE_NAME;

            StructType rowType = StructType.of(
                    "app", PrimitiveType.Text,
```

```
                    "timestamp", PrimitiveType.Timestamp,
                    "host", PrimitiveType.Text,
                    "http_code", PrimitiveType.Uint32,
                    "message", PrimitiveType.Text
            );

            // Генерация пакета записей
            List<Value<?>> rows = new ArrayList<>(BATCH_SIZE);
            for (int i = 0; i < BATCH_SIZE; i++) {
                rows.add(rowType.newValue(
                        "app", PrimitiveValue.newText("App_" + i / 256),
                        "timestamp", PrimitiveValue.newTimestamp(Instant.now().plusSeconds(i)),
                        "host", PrimitiveValue.newText("192.168.0." + i % 256),
                        "http_code", PrimitiveValue.newUint32(i % 113 == 0 ? 404 : 200),
                        "message", PrimitiveValue.newText(
                                i % 3 == 0 ? "GET / HTTP/1.1" : "GET /images/logo.png HTTP/1.1")
                ));
            }

            ListValue batch = ListType.of(rowType).newValue(rows);

            // Пакетная вставка без атомарности всего батча
            tableRetry.supplyStatus(session ->
                    session.executeBulkUpsert(tablePath, batch, new BulkUpsertSettings())
            ).join().expectSuccess("bulk upsert failed");

            // Проверка количества строк
            QueryReader countReader = queryRetry.supplyResult(session -> QueryReader.readFrom(
                    session.createQuery(
                            "SELECT COUNT(*) AS cnt FROM bulk_upsert", TxMode.NONE, Params.empty())
            )).join().getValue();

            ResultSetReader rs = countReader.getResultSet(0);
            if (rs.next()) {
                System.out.println("Строк в таблице bulk_upsert: " + rs.getColumn("cnt").getUint64());
            }
        }
    }
}
```

JDBC

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Timestamp;
import java.time.Instant;

public class JdbcBulkUpsertExample {
    private static final int BATCH_SIZE = 1000;

    public static void main(String[] args) throws SQLException {
        String url = System.getenv().getOrDefault(
                "YDB_JDBC_URL", "jdbc:ydb:grpc://localhost:2136/local");

        try (Connection conn = DriverManager.getConnection(url)) {
            try (Statement stmt = conn.createStatement()) {
                stmt.execute("""
                        CREATE TABLE IF NOT EXISTS bulk_upsert (
                            app Text,
                            timestamp Timestamp,
                            host Text,
                            http_code Uint32,
                            message Text,
                            PRIMARY KEY (app, timestamp, host)
                        );
                        """);
            }

            String bulkSql = """
                    BULK UPSERT INTO bulk_upsert (app, timestamp, host, http_code, message)
                    VALUES (?, ?, ?, ?, ?);
                    """;

            try (PreparedStatement ps = conn.prepareStatement(bulkSql)) {
                for (int i = 0; i < BATCH_SIZE; i++) {
                    ps.setString(1, "App_" + i / 256);
                    ps.setTimestamp(2, Timestamp.from(Instant.now().plusSeconds(i)));
                    ps.setString(3, "192.168.0." + i % 256);
                    ps.setLong(4, i % 113 == 0 ? 404 : 200);
                    ps.setString(5, i % 3 == 0 ? "GET / HTTP/1.1" : "GET /images/logo.png HTTP/1.1");
                    ps.addBatch();
                }
                ps.executeBatch();
            }

            try (Statement stmt = conn.createStatement();
                 ResultSet rs = stmt.executeQuery("SELECT COUNT(*) AS cnt FROM bulk_upsert")) {
```

```
                if (rs.next()) {
                    System.out.println("Строк в таблице bulk_upsert: " + rs.getLong("cnt"));
                }
            }
        }
    }
}
```

В Spring Boot, Hibernate, JOOQ и других фреймворках вокруг ORM поверх JDBC можно выполнять нативный YQL (в том числе из репозиториев и `@Query`). Драйвер стремится оптимизировать крупные вставки; операции `UPDATE`, `INSERT`, `DELETE`, `UPSERT`, идущие через JDBC, при необходимости автоматически группируются в пакеты на стороне драйвера.

Python

Native SDK

```
import posixpath
import ydb

def bulk_upsert(driver: ydb.Driver, path: str):
    column_types = (
        ydb.BulkUpsertColumns()
        .add_column("id", ydb.PrimitiveType.Uint64)
        .add_column("val", ydb.OptionalType(ydb.PrimitiveType.Utf8))
    )
    rows = [
        {"id": 1, "val": "1"},
        {"id": 2, "val": "2"},
        {"id": 3, "val": "3"},
    ]
    driver.table_client.bulk_upsert(posixpath.join(path, "tablename"), rows, column_types)
```

Native SDK (Asyncio)

```
import os
import posixpath
import ydb
import asyncio

async def bulk_upsert(driver: ydb.aio.Driver, path: str):
    column_types = (
        ydb.BulkUpsertColumns()
        .add_column("id", ydb.PrimitiveType.Uint64)
        .add_column("val", ydb.OptionalType(ydb.PrimitiveType.Utf8))
    )
    rows = [
        {"id": 1, "val": "1"},
        {"id": 2, "val": "2"},
        {"id": 3, "val": "3"},
    ]
    await driver.table_client.bulk_upsert(
        posixpath.join(path, "tablename"), rows, column_types
    )

async def main():
    async with ydb.aio.Driver(
        connection_string=os.environ["YDB_CONNECTION_STRING"],
        credentials=ydb.credentials_from_env_variables(),
    ) as driver:
        await driver.wait()
        await bulk_upsert(driver, "/local")

asyncio.run(main())
```

SQLAlchemy

```
import os
import sqlalchemy as sa
import ydb

engine = sa.create_engine(os.environ["YDB_SQLALCHEMY_URL"])
with engine.connect() as connection:
    dbapi_conn = connection.connection

    column_types = (
          ydb.BulkUpsertColumns()
          .add_column("id", ydb.PrimitiveType.Uint64)
          .add_column("val", ydb.OptionalType(ydb.PrimitiveType.Utf8))
      )
    rows = [
        {"id": 1, "val": "1"},
        {"id": 2, "val": "2"},
        {"id": 3, "val": "3"},
    ]

    dbapi_conn.bulk_upsert("tablename", rows, column_types)
```

C#

Функциональность на данный момент не поддерживается.

JavaScript

Функциональность на данный момент не поддерживается.

Rust

```rust
use ydb::{ydb_struct, AccessTokenCredentials, ClientBuilder, Value, YdbResult};

#[tokio::main]
async fn main() -> YdbResult<()> {
    let client = ClientBuilder::new_from_connection_string(
        "grpc://localhost:2136?database=local",
    )?
    .with_credentials(AccessTokenCredentials::from("..."))
    .client()?;

    client.wait().await?;

    let rows: Vec<Value> = vec![
        ydb_struct!(
            "id" => 1_u64,
            "val" => Value::Text("1".into()),
        ),
        ydb_struct!(
            "id" => 2_u64,
            "val" => Value::Text("2".into()),
        ),
        ydb_struct!(
            "id" => 3_u64,
            "val" => Value::Text("3".into()),
        ),
    ];

    client
        .table_client()
        .bulk_upsert("/local/tablename", rows)
        .idempotent(true)
        .await?;

    Ok(())
}
```

PHP

```php
<?php

use YdbPlatform\Ydb\Ydb;

$ydb = new Ydb($config);

$rows = [
    ['id' => 1, 'val' => '1'],
    ['id' => 2, 'val' => '2'],
    ['id' => 3, 'val' => '3'],
];

$ydb->table()->bulkUpsert('tablename', $rows, [
    'id'  => 'UINT64',
    'val' => 'UTF8',
]);
```

Установка режима выполнения транзакции

Для выполнения запросов в YDB SDK необходимо указывать режим выполнения транзакции.

Ниже приведены примеры кода, которые используют встроенные в YDB SDK средства создания объекта *режим выполнения транзакции*.

ImplicitTx

ImplicitTx режим позволяет выполнить один запрос без явного управления транзакцией. Запрос выполняется в своей собственной неявной транзакции, которая автоматически фиксируется в случае успеха.

C++

Native SDK

```
#include <ydb-cpp-sdk/client/query/client.h>

void ImplicitTxExample(NYdb::NQuery::TSession session) {
  auto result = session.ExecuteQuery(
      "SELECT 1",
      NYdb::NQuery::TTxControl::NoTx()
  ).GetValueSync();

  // ...
}
```

userver

Функциональность на данный момент не поддерживается.

Go

Native SDK

```
package main

import (
  "context"
  "fmt"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/query"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithAccessTokenCredentials(os.Getenv("YDB_TOKEN")),
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
  row, err := db.Query().QueryRow(ctx, "SELECT 1",
    query.WithTxControl(query.ImplicitTxControl()),
  )
  if err != nil {
    fmt.Printf("unexpected error: %v", err)
  }
  // работа с row
  _ = row
}
```

database/sql

```
package main

import (
  "context"
  "database/sql"
  "fmt"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  nativeDriver, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithAccessTokenCredentials(os.Getenv("YDB_TOKEN")),
  )
  if err != nil {
    panic(err)
  }
```

```
  defer nativeDriver.Close(ctx)

  connector, err := ydb.Connector(nativeDriver)
  if err != nil {
    panic(err)
  }
  defer connector.Close()

  db := sql.OpenDB(connector)
  defer db.Close()

  // ImplicitTx - запрос без явной транзакции (авто-коммит)
  row := db.QueryRowContext(ctx, "SELECT 1")
  var result int
  if err := row.Scan(&result); err != nil {
    fmt.Printf("unexpected error: %v", err)
  }
}
```

Java

В Java SDK режим транзакции задаётся через `TxMode` при вызове `QueryClient.createQuery`. В JDBC — через `Connection.setAutoCommit`, `setReadOnly`, `setTransactionIsolation` и свойства драйвера. Подробнее о режимах — в документации по транзакциям; инициализация клиента — в Инициализация драйвера.

Native SDK

```
import tech.ydb.common.transaction.TxMode;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;

public class ImplicitTxExample {

    public static void main(String[] args) {
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString).build();
             QueryClient queryClient = QueryClient.newClient(transport).build()) {

            SessionRetryContext retryCtx = SessionRetryContext.create(queryClient).build();

            // ImplicitTx — один запрос без явного BEGIN/COMMIT
            QueryReader reader = retryCtx.supplyResult(session -> QueryReader.readFrom(
                    session.createQuery("SELECT 1 AS value", TxMode.NONE, Params.empty())
            )).join().getValue();

            ResultSetReader rs = reader.getResultSet(0);
            if (rs.next()) {
                System.out.println("ImplicitTx: SELECT 1 = " + rs.getColumn("value").getInt32());
            }
        }
    }
}
```

JDBC

JDBC-драйвер не позволяет явно указать режим `ImplicitTx` для выполнения транзакций. Он самостоятельно использует этот режим для выполнения тех запросов, для которых он необходим:

- DDL-инструкции, такие как CREATE TABLE, DROP TABLE и т.д.
- Операции BATCH UPDATE и BATCH DELETE

> **Примечание**
>
> Так как данные операции не транзакционные и не могут быть отменены через `rollback()` драйвер не позволяет выполнять их в рамках открытой транзакции. Их следует выполнять либо в режиме автокоммита либо до выполнения каких либо других запросов к базе.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class JdbcImplicitTxExample {

    public static void main(String[] args) {
        String connectionUrl = System.getenv().getOrDefault(
                "YDB_JDBC_URL", "jdbc:ydb:grpc://localhost:2136/local");

        try (Connection connection = DriverManager.getConnection(connectionUrl)) {
            // autocommit=true по умолчанию — неявная транзакция (ImplicitTx)
            try (Statement statement = connection.createStatement();
```

```
                 ResultSet rs = statement.executeQuery("SELECT 1 AS value")) {
                rs.next();
                System.out.println("ImplicitTx: SELECT 1 = " + rs.getInt("value"));
            }

            // DDL также выполняется в ImplicitTx
            try (Statement statement = connection.createStatement()) {
                statement.execute(
                        "CREATE TABLE IF NOT EXISTS tx_demo (id Int32, value Text, PRIMARY KEY (id))");
            }
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Python

Native SDK

```
import ydb

def execute_query(pool: ydb.QuerySessionPool):
    pool.execute_with_retries("SELECT 1")
```

Native SDK (Asyncio)

```
import ydb

async def execute_query(pool: ydb.aio.QuerySessionPool):
    await pool.execute_with_retries("SELECT 1")
```

SQLAlchemy

```
import sqlalchemy as sa
from ydb_sqlalchemy import IsolationLevel

engine = sa.create_engine("yql+ydb://localhost:2136/local")
with engine.connect().execution_options(isolation_level=IsolationLevel.AUTOCOMMIT) as connection:
    result = connection.execute(sa.text("SELECT 1"))
```

C#

ADO.NET

```
using Ydb.Sdk.Ado;

await using var connection = await dataSource.OpenRetryableConnectionAsync();
// Выполнение без явной транзакции (авто-коммит)
await using var command = new YdbCommand(connection) { CommandText = "SELECT 1" };
await command.ExecuteNonQueryAsync();
```

Entity Framework

```
using Microsoft.EntityFrameworkCore;

await using var context = await dbContextFactory.CreateDbContextAsync();
// Режим авто-коммита Entity Framework (без явной транзакции)
var result = await context.SomeEntities.FirstOrDefaultAsync();
```

linq2db

```
using LinqToDB;
using LinqToDB.Data;

using var db = new DataConnection(
    new DataOptions().UseConnectionString(
        "YDB",
        "Host=localhost;Port=2136;Database=/local;UseTls=false"
    )
);
// Режим авто-коммита linq2db (без явной транзакции)
var result = db.GetTable<Employee>().FirstOrDefault(e => e.Id == 1);
```

JavaScript

```
import { sql } from '@ydbjs/query';

// ...

// ImplicitTx - один запрос без явной транзакции
const result = await sql`SELECT 1`;
```

Rust

```
let mut qc = client.query_client();
// ImplicitTx — режим по умолчанию для вспомогательных методов QueryClient, выполняющих один транзакционный SQL-запрос:
// сервер выбирает изоляцию по типу SQL (SELECT → snapshot RO, DML → serializable RW).
let mut row = qc.query_row("SELECT 1 AS one").await?;
```

PHP

```
<?php

use YdbPlatform\Ydb\Ydb;

$config = [
    // YDB config
];

$ydb = new Ydb($config);
$result = $ydb->table()->query('SELECT 1;');
```

Serializable

C++

Native SDK

```
#include <ydb-cpp-sdk/client/query/client.h>

void SerializableExample(NYdb::NQuery::TSession session) {
    auto settings = NYdb::NQuery::TTxSettings::SerializableRW();
    auto result = session.ExecuteQuery(
        "SELECT 1",
        NYdb::NQuery::TTxControl::BeginTx(settings).CommitTx()
    ).GetValueSync();

    // ...
}
```

userver

```
#include <userver/ydb/table.hpp>

void SerializableExample(ydb::TableClient& client) {
    auto result = client.ExecuteQuery(
        ydb::OperationSettings{.tx_mode = ydb::TransactionMode::kSerializableRW},
        ydb::Query{"SELECT 1;"}
    );
    // ...
}
```

Go

Native SDK

```
package main

import (
  "context"
  "fmt"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/query"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithAccessTokenCredentials(os.Getenv("YDB_TOKEN")),
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
  row, err := db.Query().QueryRow(ctx, "SELECT 1",
    query.WithTxControl(query.SerializableReadWriteTxControl(query.CommitTx())),
  )
  if err != nil {
    fmt.Printf("unexpected error: %v", err)
  }
  // работа с row
```

```
  _ = row
}
```

database/sql

```
package main

import (
  "context"
  "database/sql"
  "fmt"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/retry"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  nativeDriver, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithAccessTokenCredentials(os.Getenv("YDB_TOKEN")),
  )
  if err != nil {
    panic(err)
  }
  defer nativeDriver.Close(ctx)

  connector, err := ydb.Connector(nativeDriver)
  if err != nil {
    panic(err)
  }
  defer connector.Close()

  db := sql.OpenDB(connector)
  defer db.Close()

  err = retry.DoTx(ctx, db,
    func(ctx context.Context, tx *sql.Tx) error {
      row := tx.QueryRowContext(ctx, "SELECT 1")
      var result int
      return row.Scan(&result)
    },
    retry.WithIdempotent(true),
    // Режим Serializable Read-Write используется по умолчанию для транзакций
    // Либо его можно установить явно как приведено ниже
    retry.WithTxOptions(&sql.TxOptions{
      Isolation: sql.LevelSerializable,
      ReadOnly:  false,
    }),
  )
  if err != nil {
    fmt.Printf("unexpected error: %v", err)
  }
}
```

Java

Native SDK

```
import tech.ydb.common.transaction.TxMode;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;

public class SerializableTxExample {

    public static void main(String[] args) {
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString).build();
             QueryClient queryClient = QueryClient.newClient(transport).build()) {

            SessionRetryContext retryCtx = SessionRetryContext.create(queryClient).build();

            QueryReader reader = retryCtx.supplyResult(session -> QueryReader.readFrom(
                    session.createQuery("SELECT 1 AS value", TxMode.SERIALIZABLE_RW, Params.empty())
            )).join().getValue();

            ResultSetReader rs = reader.getResultSet(0);
            if (rs.next()) {
                System.out.println("Serializable RW: SELECT 1 = " + rs.getColumn("value").getInt32());
            }
        }
```

```
    }
}
```

JDBC

JDBC-драйвер использует режим `Serializable` по умолчанию для выполнения всех не read-only запросов.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class JdbcSerializableTxExample {

    public static void main(String[] args) {
        String connectionUrl = System.getenv().getOrDefault(
                "YDB_JDBC_URL", "jdbc:ydb:grpc://localhost:2136/local");

        try (Connection connection = DriverManager.getConnection(connectionUrl)) {
            connection.setAutoCommit(false);
            connection.setReadOnly(false);
            connection.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);

            try (PreparedStatement ps = connection.prepareStatement("SELECT 1 AS value");
                 ResultSet rs = ps.executeQuery()) {
                rs.next();
                System.out.println("Serializable RW: SELECT 1 = " + rs.getInt("value"));
            }

            connection.commit();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Python

Native SDK

```
import ydb

def execute_query(pool: ydb.QuerySessionPool):
    def callee(session: ydb.QuerySession):
        with session.transaction(ydb.QuerySerializableReadWrite()).execute(
            "SELECT 1",
            commit_tx=True,
        ) as result_sets:
            pass

    pool.retry_operation_sync(callee)
```

Native SDK (Asyncio)

```
import ydb

async def execute_query(pool: ydb.aio.QuerySessionPool):
    async def callee(session):
        async with session.transaction(tx_mode=ydb.QuerySerializableReadWrite()) as tx:
            async with await tx.execute("SELECT 1", commit_tx=True) as result_sets:
                pass

    await pool.retry_operation_async(callee)
```

SQLAlchemy

```
import sqlalchemy as sa
from ydb_sqlalchemy import IsolationLevel

engine = sa.create_engine("yql+ydb://localhost:2136/local")
with engine.connect().execution_options(isolation_level=IsolationLevel.SERIALIZABLE) as connection:
    result = connection.execute(sa.text("SELECT 1"))
```

C#

ADO.NET

```
using Ydb.Sdk.Ado;

// Режим Serializable используется по умолчанию
await _ydbDataSource.ExecuteInTransactionAsync(async ydbConnection =>
    {
        var ydbCommand = ydbConnection.CreateCommand();
        ydbCommand.CommandText = """
                                 UPSERT INTO episodes (series_id, season_id, episode_id, title, air_date)
```

```
                                VALUES (2, 5, 13, "Test Episode", Date("2018-08-27"))
                                """;
        await ydbCommand.ExecuteNonQueryAsync();
        ydbCommand.CommandText = """
                                INSERT INTO episodes(series_id, season_id, episode_id, title, air_date)
                                VALUES
                                    (2, 5, 21, "Test 21", Date("2018-08-27")),
                                    (2, 5, 22, "Test 22", Date("2018-08-27"))
                                """;
        await ydbCommand.ExecuteNonQueryAsync();
    }
);
```

Entity Framework

```
var strategy = db.Database.CreateExecutionStrategy();

// Entity Framework использует режим Serializable по умолчанию
strategy.ExecuteInTransaction(
    db,
    ctx =>
    {
        ctx.Users.AddRange(
            new User { Name = "Alex", Email = "alex@example.com" },
            new User { Name = "Kirill", Email = "kirill@example.com" }
        );

        ctx.SaveChanges();

        var users = ctx.Users.OrderBy(u => u.Id).ToList();
        Console.WriteLine("Users in database:");
        foreach (var user in users)
            Console.WriteLine($"- {user.Id}: {user.Name} ({user.Email})");
    },
    ctx => ctx.Users.Any(u => u.Email == "alex@example.com")
        && ctx.Users.Any(u => u.Email == "kirill@example.com")
  );
```

linq2db

```
using LinqToDB;
using LinqToDB.Data;

// linq2db использует режим Serializable по умолчанию
using var db = new DataConnection(
    new DataOptions().UseConnectionString(
        "YDB",
        "Host=localhost;Port=2136;Database=/local;UseTls=false"
    )
);

await using var tr = await db.BeginTransactionAsync();

await db.InsertAsync(new Episode
{
    SeriesId = 2, SeasonId = 5, EpisodeId = 13, Title = "Test Episode", AirDate = new DateTime(2018, 08, 27)
});
await db.InsertAsync(new Episode
    { SeriesId = 2, SeasonId = 5, EpisodeId = 21, Title = "Test 21", AirDate = new DateTime(2018, 08, 27) });
await db.InsertAsync(new Episode
    { SeriesId = 2, SeasonId = 5, EpisodeId = 22, Title = "Test 22", AirDate = new DateTime(2018, 08, 27) });

await tr.CommitAsync();
```

JavaScript

```
import { sql } from '@ydbjs/query';

// ...

// Режим Serializable Read-Write используется по умолчанию
await sql.begin({ idempotent: true }, async (tx) => {
    return await tx`SELECT 1`;
});

// Или явно указать режим транзакции
await sql.begin({ isolation: 'serializableReadWrite', idempotent: true }, async (tx) => {
    return await tx`SELECT 1`;
});
```

Rust

```
use ydb::TxMode;

client
    .query_client()
    .retry_tx(async |tx| {
```

```
        tx.query_row("SELECT 1 AS one").await?;
        Ok(())
    })
    .isolation(TxMode::SerializableReadWrite)
    .idempotent(true)
    .await?;
```

PHP

```
<?php

use YdbPlatform\Ydb\Ydb;

$config = [
    // YDB config
];

$ydb = new Ydb($config);
$result = $ydb->table()->retryTransaction(
  function (Session $session) {
      return $session->query('SELECT 1 AS value;');
  },
  true,
  null,
  ['tx_mode' => 'serializable_read_write']
);
```

Online Read-Only

C++

Native SDK

```
#include <ydb-cpp-sdk/client/query/client.h>

void OnlineReadOnlyExample(NYdb::NQuery::TSession session) {
    auto settings = NYdb::NQuery::TTxSettings::OnlineRO();
    auto result = session.ExecuteQuery(
        "SELECT 1",
        NYdb::NQuery::TTxControl::BeginTx(settings).CommitTx()
    ).GetValueSync();

    // ...
}
```

userver

```
#include <userver/ydb/table.hpp>

void OnlineReadOnlyExample(ydb::TableClient& client) {
    auto result = client.ExecuteQuery(
        ydb::OperationSettings{.tx_mode = ydb::TransactionMode::OnlineRO},
        ydb::Query{"SELECT 1;"}
    );
    // ...
}
```

Go

Native SDK

```
package main

import (
  "context"
  "fmt"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/query"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithAccessTokenCredentials(os.Getenv("YDB_TOKEN")),
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
  row, err := db.Query().QueryRow(ctx, "SELECT 1",
    query.WithTxControl(
      query.OnlineReadOnlyTxControl(query.WithInconsistentReads()),
    ),
```

```
  )
  if err != nil {
    fmt.Printf("unexpected error: %v", err)
  }
  // работа с row
  _ = row
}
```

database/sql

```
package main

import (
  "context"
  "database/sql"
  "fmt"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/retry"
  "github.com/ydb-platform/ydb-go-sdk/v3/table"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  nativeDriver, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithAccessTokenCredentials(os.Getenv("YDB_TOKEN")),
  )
  if err != nil {
    panic(err)
  }
  defer nativeDriver.Close(ctx)

  connector, err := ydb.Connector(nativeDriver)
  if err != nil {
    panic(err)
  }
  defer connector.Close()

  db := sql.OpenDB(connector)
  defer db.Close()

  err = retry.Do(
    ydb.WithTxControl(ctx, table.OnlineReadOnlyTxControl(table.WithInconsistentReads())),
    db,
    func(ctx context.Context, conn *sql.Conn) error {
      row := conn.QueryRowContext(ctx, "SELECT 1")
      var result int
      return row.Scan(&result)
    },
    retry.WithIdempotent(true),
  )
  if err != nil {
    fmt.Printf("unexpected error: %v", err)
  }
}
```

Java

Native SDK

```
import tech.ydb.common.transaction.TxMode;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;

public class OnlineReadOnlyTxExample {

    public static void main(String[] args) {
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString).build();
             QueryClient queryClient = QueryClient.newClient(transport).build()) {

            SessionRetryContext retryCtx = SessionRetryContext.create(queryClient).build();

            QueryReader reader = retryCtx.supplyResult(session -> QueryReader.readFrom(
                    session.createQuery("SELECT 1 AS value", TxMode.ONLINE_RO, Params.empty())
            )).join().getValue();

            ResultSetReader rs = reader.getResultSet(0);
            if (rs.next()) {
                System.out.println("Online RO: SELECT 1 = " + rs.getColumn("value").getInt32());
            }
```

```
        }
    }
}
```

JDBC

JDBC стандарт не поддерживает уровни транзакций кроме стандартных. Однако JDBC драйвер позволяет задать этот режим с помощью задания нестандартной константы 16.

> **Важно**
>
> Данный режим не поддерживает интерактивные транзакции.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class JdbcOnlineReadOnlyTxExample {

    public static void main(String[] args) {
        String connectionUrl = System.getenv().getOrDefault(
                "YDB_JDBC_URL", "jdbc:ydb:grpc://localhost:2136/local");

        try (Connection connection = DriverManager.getConnection(connectionUrl)) {
            connection.setAutoCommit(true); // режим не поддерживает интерактивные транзакции
            connection.setReadOnly(true);
            connection.setTransactionIsolation(16); // ONLINE_RO в YDB JDBC

            try (Statement statement = connection.createStatement();
                 ResultSet rs = statement.executeQuery("SELECT 1 AS value")) {
                rs.next();
                System.out.println("Online RO: SELECT 1 = " + rs.getInt("value"));
            }
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Python

Native SDK

```
import ydb

def execute_query(pool: ydb.QuerySessionPool):
    def callee(session: ydb.QuerySession):
        with session.transaction(ydb.QueryOnlineReadOnly()).execute(
            "SELECT 1",
            commit_tx=True,
        ) as result_sets:
            pass

    pool.retry_operation_sync(callee)
```

Native SDK (Asyncio)

```
import ydb

async def execute_query(pool: ydb.aio.QuerySessionPool):
    async def callee(session):
        async with session.transaction(tx_mode=ydb.QueryOnlineReadOnly()) as tx:
            async with await tx.execute("SELECT 1", commit_tx=True) as result_sets:
                pass

    await pool.retry_operation_async(callee)
```

SQLAlchemy

```
import sqlalchemy as sa
from ydb_sqlalchemy import IsolationLevel

engine = sa.create_engine("yql+ydb://localhost:2136/local")
with engine.connect().execution_options(isolation_level=IsolationLevel.ONLINE_READONLY) as connection:
    result = connection.execute(sa.text("SELECT 1"))
```

C#

ADO.NET

```
using Ydb.Sdk.Ado;

// OnlineRo — данные максимально актуальны на момент каждой операции чтения;
```

```
// данные в рамках одной транзакции согласованы
await using var connection = await dataSource.OpenConnectionAsync();
await using var transaction = await connection.BeginTransactionAsync(TransactionMode.OnlineRo);
await using var command = new YdbCommand(connection) { CommandText = "SELECT 1" };
await using var reader = await command.ExecuteReaderAsync();
await transaction.CommitAsync();

// OnlineInconsistentRo — максимальная производительность, минимальная согласованность:
// данные могут быть несогласованы даже в рамках одной операции чтения
await using var connection2 = await dataSource.OpenConnectionAsync();
await using var transaction2 = await connection2.BeginTransactionAsync(TransactionMode.OnlineInconsistentRo);
await using var command2 = new YdbCommand(connection2) { CommandText = "SELECT 1" };
await using var reader2 = await command2.ExecuteReaderAsync();
await transaction2.CommitAsync();
```

Entity Framework

Entity Framework не поддерживает режим OnlineRo напрямую.
Используйте ydb-dotnet-sdk или ADO.NET для этого уровня изоляции.

linq2db

linq2db не поддерживает режим OnlineRo напрямую.
Используйте ydb-dotnet-sdk или ADO.NET для этого уровня изоляции.

JavaScript

```
import { sql } from '@ydbjs/query';

// ...

await sql.begin({ isolation: 'onlineReadOnly', idempotent: true }, async (tx) => {
    return await tx`SELECT 1`;
});
```

Rust

```
use ydb::TxMode;

let mut qc = client.query_client();

// Online RO — согласованное чтение (allow_inconsistent_reads = false)
let mut row = qc
    .query_row("SELECT 1 AS one")
    .with_tx_mode(TxMode::OnlineReadOnly)
    .await?;

// Online inconsistent RO — максимальная производительность (allow_inconsistent_reads = true)
let mut row = qc
    .query_row("SELECT 1 AS one")
    .with_tx_mode(TxMode::OnlineReadOnlyInconsistent)
    .await?;
```

PHP

```
<?php

use YdbPlatform\Ydb\Ydb;

$config = [
    // YDB config
];

$ydb = new Ydb($config);
$result = $ydb->table()->retrySession(function (Session $session) {
  $query = $session->newQuery('SELECT 1 AS value;');
  $query->beginTx('online_read_only');
  return $query->execute();
}, true);
```

Stale Read-Only

C++

Native SDK

```
#include <ydb-cpp-sdk/client/query/client.h>

void StaleReadOnlyExample(NYdb::NQuery::TSession session) {
    auto settings = NYdb::NQuery::TTxSettings::StaleRO();
    auto result = session.ExecuteQuery(
        "SELECT 1",
        NYdb::NQuery::TTxControl::BeginTx(settings).CommitTx()
    ).GetValueSync();

    // ...
}
```

userver

```
#include <userver/ydb/table.hpp>

void StaleReadOnlyExample(ydb::TableClient& client) {
    auto result = client.ExecuteQuery(
        ydb::OperationSettings{.tx_mode = ydb::TransactionMode::kStaleRO},
        ydb::Query{"SELECT 1;"}
    );
    // ...
}
```

Go

Native SDK

```
package main

import (
  "context"
  "fmt"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/query"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithAccessTokenCredentials(os.Getenv("YDB_TOKEN")),
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
  row, err := db.Query().QueryRow(ctx, "SELECT 1",
    query.WithTxControl(query.StaleReadOnlyTxControl()),
  )
  if err != nil {
    fmt.Printf("unexpected error: %v", err)
  }
  // работа с row
  _ = row
}
```

database/sql

```
package main

import (
  "context"
  "database/sql"
  "fmt"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/retry"
  "github.com/ydb-platform/ydb-go-sdk/v3/table"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  nativeDriver, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithAccessTokenCredentials(os.Getenv("YDB_TOKEN")),
  )
  if err != nil {
    panic(err)
  }
  defer nativeDriver.Close(ctx)

  connector, err := ydb.Connector(nativeDriver)
  if err != nil {
    panic(err)
  }
  defer connector.Close()

  db := sql.OpenDB(connector)
  defer db.Close()

  err = retry.Do(
    ydb.WithTxControl(ctx, table.StaleReadOnlyTxControl()),
    db,
    func(ctx context.Context, conn *sql.Conn) error {
      row := conn.QueryRowContext(ctx, "SELECT 1")
```

```
      var result int
      return row.Scan(&result)
    },
    retry.WithIdempotent(true),
  )
  if err != nil {
    fmt.Printf("unexpected error: %v", err)
  }
}
```

Java

Native SDK

```
import tech.ydb.common.transaction.TxMode;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;

public class StaleReadOnlyTxExample {

    public static void main(String[] args) {
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString).build();
             QueryClient queryClient = QueryClient.newClient(transport).build()) {

            SessionRetryContext retryCtx = SessionRetryContext.create(queryClient).build();

            QueryReader reader = retryCtx.supplyResult(session -> QueryReader.readFrom(
                    session.createQuery("SELECT 1 AS value", TxMode.STALE_RO, Params.empty())
            )).join().getValue();

            ResultSetReader rs = reader.getResultSet(0);
            if (rs.next()) {
                System.out.println("Stale RO: SELECT 1 = " + rs.getColumn("value").getInt32());
            }
        }
    }
}
```

JDBC

JDBC стандарт не поддерживает уровни транзакций кроме стандартных. Однако JDBC драйвер позволяет задать этот режим с помощью задания нестандартной константы 32.

> **Важно**
>
> Данный режим не поддерживает интерактивные транзакции.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class JdbcStaleReadOnlyTxExample {

    public static void main(String[] args) {
        String connectionUrl = System.getenv().getOrDefault(
                "YDB_JDBC_URL", "jdbc:ydb:grpc://localhost:2136/local");

        try (Connection connection = DriverManager.getConnection(connectionUrl)) {
            connection.setAutoCommit(true); // режим не поддерживает интерактивные транзакции
            connection.setReadOnly(true);
            connection.setTransactionIsolation(32); // STALE_RO в YDB JDBC

            try (Statement statement = connection.createStatement();
                 ResultSet rs = statement.executeQuery("SELECT 1 AS value")) {
                rs.next();
                System.out.println("Stale RO: SELECT 1 = " + rs.getInt("value"));
            }
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Python

Native SDK

```python
import ydb

def execute_query(pool: ydb.QuerySessionPool):
    def callee(session: ydb.QuerySession):
        with session.transaction(ydb.QueryStaleReadOnly()).execute(
            "SELECT 1",
            commit_tx=True,
        ) as result_sets:
            pass

    pool.retry_operation_sync(callee)
```

Native SDK (Asyncio)

```python
import ydb

async def execute_query(pool: ydb.aio.QuerySessionPool):
    async def callee(session):
        async with session.transaction(tx_mode=ydb.QueryStaleReadOnly()) as tx:
            async with await tx.execute("SELECT 1", commit_tx=True) as result_sets:
                pass

    await pool.retry_operation_async(callee)
```

SQLAlchemy

```python
import sqlalchemy as sa
from ydb_sqlalchemy import IsolationLevel

engine = sa.create_engine("yql+ydb://localhost:2136/local")
with engine.connect().execution_options(isolation_level=IsolationLevel.STALE_READONLY) as connection:
    result = connection.execute(sa.text("SELECT 1"))
```

C#

ADO.NET

```csharp
using Ydb.Sdk.Ado;

// StaleRo — данные могут быть незначительно устаревшими относительно актуального состояния;
// обеспечивает максимальную скорость чтения за счёт ослабленной согласованности
await using var connection = await dataSource.OpenConnectionAsync();
await using var transaction = await connection.BeginTransactionAsync(TransactionMode.StaleRo);
await using var command = new YdbCommand(connection) { CommandText = "SELECT 1", Transaction = transaction };
await using var reader = await command.ExecuteReaderAsync();
await transaction.CommitAsync();
```

Entity Framework

Entity Framework не поддерживает режим StaleRo напрямую.
Используйте ydb-dotnet-sdk или ADO.NET для этого уровня изоляции.

linq2db

linq2db не поддерживает режим StaleRo напрямую.
Используйте ydb-dotnet-sdk или ADO.NET для этого уровня изоляции.

JavaScript

```javascript
import { sql } from '@ydbjs/query';

// ...

await sql.begin({ isolation: 'staleReadOnly', idempotent: true }, async (tx) => {
    return await tx`SELECT 1`;
});
```

Rust

```rust
use ydb::TxMode;

let mut qc = client.query_client();
// Режим Stale Read-Only поддерживается только для one-shot вызовов на query-клиенте (не для retry_tx).
let mut row = qc
    .query_row("SELECT 1 AS one")
    .with_tx_mode(TxMode::StaleReadOnly)
    .await?;
```

PHP

```php
<?php

use YdbPlatform\Ydb\Ydb;

$config = [
    // YDB config
];

$ydb = new Ydb($config);
$result = $ydb->table()->retrySession(function (Session $session) {
  $query = $session->newQuery('SELECT 1 AS value;');
  $query->beginTx('stale_read_only');
  return $query->execute();
}, true);
```

Snapshot Read-Only

C++

Native SDK

```cpp
#include <ydb-cpp-sdk/client/query/client.h>

void SnapshotReadOnlyExample(NYdb::NQuery::TSession session) {
    auto settings = NYdb::NQuery::TTxSettings::SnapshotRO();
    auto result = session.ExecuteQuery(
        "SELECT 1",
        NYdb::NQuery::TTxControl::BeginTx(settings).CommitTx()
    ).GetValueSync();

    // ...
}
```

userver

```cpp
#include <userver/ydb/table.hpp>

void SnapshotReadOnlyExample(ydb::TableClient& client) {
    auto result = client.ExecuteQuery(
        ydb::OperationSettings{.tx_mode = ydb::TransactionMode::kSnapshotRO},
        ydb::Query{"SELECT 1;"}
    );
    // ...
}
```

Go

Native SDK

```go
package main

import (
  "context"
  "fmt"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/query"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithAccessTokenCredentials(os.Getenv("YDB_TOKEN")),
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
  row, err := db.Query().QueryRow(ctx, "SELECT 1",
    query.WithTxControl(query.SnapshotReadOnlyTxControl()),
  )
  if err != nil {
    fmt.Printf("unexpected error: %v", err)
  }
  // работа с row
  _ = row
}
```

database/sql

```
package main

import (
  "context"
  "database/sql"
  "fmt"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/retry"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  nativeDriver, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithAccessTokenCredentials(os.Getenv("YDB_TOKEN")),
  )
  if err != nil {
    panic(err)
  }
  defer nativeDriver.Close(ctx)

  connector, err := ydb.Connector(nativeDriver)
  if err != nil {
    panic(err)
  }
  defer connector.Close()

  db := sql.OpenDB(connector)
  defer db.Close()

  // Snapshot Read-Only — обеспечивает согласованное чтение данных на определённый момент времени
  err = retry.DoTx(ctx, db, func(ctx context.Context, tx *sql.Tx) error {
    row := tx.QueryRowContext(ctx, "SELECT 1")
    var result int
    return row.Scan(&result)
  }, retry.WithIdempotent(true), retry.WithTxOptions(&sql.TxOptions{
    Isolation: sql.LevelSnapshot,
    ReadOnly:  true,
  }))
  if err != nil {
    fmt.Printf("unexpected error: %v", err)
  }
}
```

Java

Native SDK

```
import tech.ydb.common.transaction.TxMode;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;

public class SnapshotReadOnlyTxExample {

    public static void main(String[] args) {
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString).build();
             QueryClient queryClient = QueryClient.newClient(transport).build()) {

            SessionRetryContext retryCtx = SessionRetryContext.create(queryClient).build();

            QueryReader reader = retryCtx.supplyResult(session -> QueryReader.readFrom(
                    session.createQuery("SELECT 1 AS value", TxMode.SNAPSHOT_RO, Params.empty())
            )).join().getValue();

            ResultSetReader rs = reader.getResultSet(0);
            if (rs.next()) {
                System.out.println("Snapshot RO: SELECT 1 = " + rs.getColumn("value").getInt32());
            }
        }
    }
}
```

JDBC

JDBC-драйвер использует режим `Snapshot Read-Only` для выполнения всех read-only запросов, при условии что используются стандартные режимы транзакций (TRANSACTION_SERIALIZABLE или REPEATABLE_READ).

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class JdbcSnapshotReadOnlyTxExample {

    public static void main(String[] args) {
        String connectionUrl = System.getenv().getOrDefault(
                "YDB_JDBC_URL", "jdbc:ydb:grpc://localhost:2136/local");

        try (Connection connection = DriverManager.getConnection(connectionUrl)) {
            connection.setAutoCommit(false);
            connection.setReadOnly(true); // Snapshot Read-Only
            connection.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);

            try (PreparedStatement ps = connection.prepareStatement("SELECT 1 AS value");
                 ResultSet rs = ps.executeQuery()) {
                rs.next();
                System.out.println("Snapshot RO: SELECT 1 = " + rs.getInt("value"));
            }

            connection.commit();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Python

Native SDK

```python
import ydb

def execute_query(pool: ydb.QuerySessionPool):
    def callee(session: ydb.QuerySession):
        with session.transaction(ydb.QuerySnapshotReadOnly()).execute(
            "SELECT 1",
            commit_tx=True,
        ) as result_sets:
            pass

    pool.retry_operation_sync(callee)
```

Native SDK (Asyncio)

```python
import ydb

async def execute_query(pool: ydb.aio.QuerySessionPool):
    async def callee(session):
        async with session.transaction(tx_mode=ydb.QuerySnapshotReadOnly()) as tx:
            async with await tx.execute("SELECT 1", commit_tx=True) as result_sets:
                pass

    await pool.retry_operation_async(callee)
```

SQLAlchemy

```python
import sqlalchemy as sa
from ydb_sqlalchemy import IsolationLevel

engine = sa.create_engine("yql+ydb://localhost:2136/local")
with engine.connect().execution_options(isolation_level=IsolationLevel.SNAPSHOT_READONLY) as connection:
    result = connection.execute(sa.text("SELECT 1"))
```

C#

ADO.NET

```csharp
using Ydb.Sdk.Ado;

await using var connection = await dataSource.OpenConnectionAsync();
await using var transaction = await connection.BeginTransactionAsync(TransactionMode.SnapshotRo);
await using var command = new YdbCommand(connection) { CommandText = "SELECT 1" };
await using var reader = await command.ExecuteReaderAsync();
await transaction.CommitAsync();
```

Entity Framework

Entity Framework не поддерживает режим Snapshot Read-Only напрямую.
Используйте ydb-dotnet-sdk или ADO.NET для этого уровня изоляции.

linq2db

linq2db не поддерживает режим Snapshot Read-Only напрямую.
Используйте ydb-dotnet-sdk или ADO.NET для этого уровня изоляции.

JavaScript

```
import { sql } from '@ydbjs/query';

// ...

await sql.begin({ isolation: 'snapshotReadOnly', idempotent: true }, async (tx) => {
    return await tx`SELECT 1`;
});
```

Rust

```
use ydb::TxMode;

let mut qc = client.query_client();
qc.query_row("SELECT 1 AS one")
    .with_tx_mode(TxMode::SnapshotReadOnly)
    .await?;

qc.retry_tx(async |tx| {
    tx.query_row("SELECT 1 AS one").await?;
    Ok(())
})
.isolation(TxMode::SnapshotReadOnly)
.idempotent(true)
.await?;
```

PHP

```
<?php

use YdbPlatform\Ydb\Ydb;

$config = [
    // YDB config
];

$ydb = new Ydb($config);
$result = $ydb->table()->retryTransaction(
  function (Session $session) {
      return $session->query('SELECT 1 AS value;');
  },
  true,
  null,
  ['tx_mode' => 'snapshot_read_only']
);
```

Snapshot Read-Write

C++

Native SDK

```
#include <ydb-cpp-sdk/client/query/client.h>

void SnapshotReadWriteExample(NYdb::NQuery::TSession session) {
    auto settings = NYdb::NQuery::TTxSettings::SnapshotRW();
    auto result = session.ExecuteQuery(
        "SELECT 1",
        NYdb::NQuery::TTxControl::BeginTx(settings).CommitTx()
    ).GetValueSync();

    // ...
}
```

userver

```
#include <userver/ydb/table.hpp>

void SnapshotReadWriteExample(ydb::TableClient& client) {
    auto result = client.ExecuteQuery(
        ydb::OperationSettings{.tx_mode = ydb::TransactionMode::kSnapshotRW},
        ydb::Query{"SELECT 1;"}
    );
    // ...
}
```

Go

Native SDK

```
package main

import (
  "context"
  "fmt"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/query"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  db, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithAccessTokenCredentials(os.Getenv("YDB_TOKEN")),
  )
  if err != nil {
    panic(err)
  }
  defer db.Close(ctx)
  row, err := db.Query().QueryRow(ctx, "SELECT 1",
    query.WithTxControl(query.SnapshotReadWriteTxControl(query.CommitTx())),
  )
  if err != nil {
    fmt.Printf("unexpected error: %v", err)
  }
  // работа с row
  _ = row
}
```

database/sql

```
package main

import (
  "context"
  "database/sql"
  "fmt"
  "os"

  "github.com/ydb-platform/ydb-go-sdk/v3"
  "github.com/ydb-platform/ydb-go-sdk/v3/retry"
)

func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  nativeDriver, err := ydb.Open(ctx,
    os.Getenv("YDB_CONNECTION_STRING"),
    ydb.WithAccessTokenCredentials(os.Getenv("YDB_TOKEN")),
  )
  if err != nil {
    panic(err)
  }
  defer nativeDriver.Close(ctx)

  connector, err := ydb.Connector(nativeDriver)
  if err != nil {
    panic(err)
  }
  defer connector.Close()

  db := sql.OpenDB(connector)
  defer db.Close()

  // Snapshot Read-Write — обеспечивает согласованное чтение данных на определённый момент времени
  // с возможностью записи
  err = retry.DoTx(ctx, db, func(ctx context.Context, tx *sql.Tx) error {
    row := tx.QueryRowContext(ctx, "SELECT 1")
    var result int
    return row.Scan(&result)
  }, retry.WithIdempotent(true), retry.WithTxOptions(&sql.TxOptions{
    Isolation: sql.LevelSnapshot,
    ReadOnly:  false,
  }))
  if err != nil {
    fmt.Printf("unexpected error: %v", err)
  }
}
```

Java

Native SDK

```
import tech.ydb.common.transaction.TxMode;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;

public class SnapshotReadWriteTxExample {

    public static void main(String[] args) {
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString).build();
             QueryClient queryClient = QueryClient.newClient(transport).build()) {

            SessionRetryContext retryCtx = SessionRetryContext.create(queryClient).build();

            QueryReader reader = retryCtx.supplyResult(session -> QueryReader.readFrom(
                    session.createQuery("SELECT 1 AS value", TxMode.SNAPSHOT_RW, Params.empty())
            )).join().getValue();

            ResultSetReader rs = reader.getResultSet(0);
            if (rs.next()) {
                System.out.println("Snapshot RW: SELECT 1 = " + rs.getColumn("value").getInt32());
            }
        }
    }
}
```

JDBC

Для установки режима `Snapshot Read-Write` следует использовать стандартный режим REPEATABLE_READ.

> **Примечание**
>
> Данный режим поддерживается начиная с версии JDBC-драйвера 2.3.24 и требует явного включения через опцию `repeatableReadEnabled`.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Properties;

public class JdbcSnapshotReadWriteTxExample {

    public static void main(String[] args) {
        String connectionUrl = System.getenv().getOrDefault(
                "YDB_JDBC_URL", "jdbc:ydb:grpc://localhost:2136/local");

        Properties props = new Properties();
        props.setProperty("repeatableReadEnabled", "true");

        try (Connection connection = DriverManager.getConnection(connectionUrl, props)) {
            connection.setAutoCommit(false);
            connection.setReadOnly(false);
            connection.setTransactionIsolation(Connection.TRANSACTION_REPEATABLE_READ);

            try (PreparedStatement ps = connection.prepareStatement("SELECT 1 AS value");
                 ResultSet rs = ps.executeQuery()) {
                rs.next();
                System.out.println("Snapshot RW: SELECT 1 = " + rs.getInt("value"));
            }

            connection.commit();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Python

Native SDK

```
import ydb

def execute_query(pool: ydb.QuerySessionPool):
    def callee(session: ydb.QuerySession):
        with session.transaction(ydb.QuerySnapshotReadWrite()).execute(
```

```
            "SELECT 1",
            commit_tx=True,
        ) as result_sets:
            pass

    pool.retry_operation_sync(callee)
```

Native SDK (Asyncio)

```
import ydb

async def execute_query(pool: ydb.aio.QuerySessionPool):
    async def callee(session):
        async with session.transaction(tx_mode=ydb.QuerySnapshotReadWrite()) as tx:
            async with await tx.execute("SELECT 1", commit_tx=True) as result_sets:
                pass

    await pool.retry_operation_async(callee)
```

SQLAlchemy

```
import sqlalchemy as sa
from ydb_sqlalchemy import IsolationLevel

engine = sa.create_engine("yql+ydb://localhost:2136/local")
with engine.connect().execution_options(isolation_level=IsolationLevel.SNAPSHOT_READWRITE) as connection:
    result = connection.execute(sa.text("SELECT 1"))
```

C#

ADO.NET

```
await _ydbDataSource.ExecuteInTransactionAsync(async ydbConnection =>
    {
        var ydbCommand = ydbConnection.CreateCommand();
        ydbCommand.CommandText = """
                                 UPSERT INTO episodes (series_id, season_id, episode_id, title, air_date)
                                 VALUES (2, 5, 13, "Test Episode", Date("2018-08-27"))
                                 """;
        await ydbCommand.ExecuteNonQueryAsync();
        ydbCommand.CommandText = """
                                 INSERT INTO episodes(series_id, season_id, episode_id, title, air_date)
                                 VALUES
                                     (2, 5, 21, "Test 21", Date("2018-08-27")),
                                     (2, 5, 22, "Test 22", Date("2018-08-27"))
                                 """;
        await ydbCommand.ExecuteNonQueryAsync();
    }, TransactionMode.SnapshotRw
);
```

Entity Framework

```
var strategy = db.Database.CreateExecutionStrategy();

strategy.Execute(() =>
    {
        using var ctx = new AppDbContext(options);
        using var tr = ctx.Database.BeginTransaction(IsolationLevel.Snapshot);

        ctx.Users.AddRange(
            new User { Name = "Alex", Email = "alex@example.com" },
            new User { Name = "Kirill", Email = "kirill@example.com" }
        );

        ctx.SaveChanges();

        var users = ctx.Users.OrderBy(u => u.Id).ToList();
        Console.WriteLine("Users in database:");
        foreach (var user in users)
            Console.WriteLine($"- {user.Id}: {user.Name} ({user.Email})");
      }
);
```

linq2db

```
await using var db = new MyYdb(BuildOptions());
await using var tr = await db.BeginTransactionAsync(IsolationLevel.Snapshot);

await db.InsertAsync(new Episode
{
    SeriesId = 2, SeasonId = 5, EpisodeId = 13, Title = "Test Episode", AirDate = new DateTime(2018, 08, 27)
});
await db.InsertAsync(new Episode
    { SeriesId = 2, SeasonId = 5, EpisodeId = 21, Title = "Test 21", AirDate = new DateTime(2018, 08, 27) });
await db.InsertAsync(new Episode
    { SeriesId = 2, SeasonId = 5, EpisodeId = 22, Title = "Test 22", AirDate = new DateTime(2018, 08, 27) });
```

```
await tr.CommitAsync();
```

JavaScript

```
import { sql } from '@ydbjs/query';

// ...

await sql.begin({ isolation: 'snapshotReadWrite', idempotent: true }, async (tx) => {
    return await tx`SELECT 1`;
});
```

Rust

```
use ydb::TxMode;

let mut qc = client.query_client();
qc.query_row("SELECT 1 AS one")
    .with_tx_mode(TxMode::SnapshotReadWrite)
    .await?;

qc.retry_tx(async |tx| {
    tx.query_row("SELECT 1 AS one").await?;
    Ok(())
})
.isolation(TxMode::SnapshotReadWrite)
.idempotent(true)
.await?;
```

PHP

Режим Snapshot Read-Write не поддерживается в PHP SDK.

Настройка времени жизни строк (TTL) таблицы

В этом разделе приведены примеры настройки TTL строковых и колоночных таблиц при помощи YDB SDK.

Включение TTL для существующих строковых и колоночных таблиц

В приведенном ниже примере строки таблицы `mytable` будут удаляться спустя час после наступления времени, записанного в колонке `created_at`:

C++

```
session.AlterTable(
    "mytable",
    TAlterTableSettings()
        .BeginAlterTtlSettings()
            .Set("created_at", TDuration::Hours(1))
        .EndAlterTtlSettings()
);
```

Go

```
err := session.AlterTable(ctx, "mytable",
  options.WithSetTimeToLiveSettings(
    options.NewTTLSettings().ColumnDateType("created_at").ExpireAfter(time.Hour),
  ),
)
```

Python

```
session.alter_table('mytable', set_ttl_settings=ydb.TtlSettings().with_date_type_column('created_at', 3600))
```

C#

Функциональность на данный момент не поддерживается.

JavaScript

Функциональность на данный момент не поддерживается.

Java

```
AlterTableSettings settings = new AlterTableSettings()
        .setTableTtl(TableTtl.dateTimeColumn("created_at", 3600));

session.alterTable("mytable", settings).join().expectSuccess();
```

Следующий пример демонстрирует использование колонки `modified_at` с числовым типом (`Uint32`) в качестве TTL-колонки. Значение колонки интерпретируется как секунды от Unix-эпохи:

C++

```
session.AlterTable(
    "mytable",
    TAlterTableSettings()
        .BeginAlterTtlSettings()
            .Set("modified_at", TTtlSettings::EUnit::Seconds, TDuration::Hours(1))
        .EndAlterTtlSettings()
);
```

Go

```
err := session.AlterTable(ctx, "mytable",
  options.WithSetTimeToLiveSettings(
    options.NewTTLSettings().ColumnSeconds("modified_at").ExpireAfter(time.Hour),
  ),
)
```

Python

```
session.alter_table('mytable', set_ttl_settings=ydb.TtlSettings().with_value_since_unix_epoch('modified_at',
UNIT_SECONDS, 3600))
```

C#

Функциональность на данный момент не поддерживается.

JavaScript

Функциональность на данный момент не поддерживается.

Java

```
AlterTableSettings settings = new AlterTableSettings()
        .setTableTtl(TableTtl.valueSinceUnixEpoch(
                "modified_at",
```

```
                TableTtl.TtlUnit.SECONDS,
                3600
        ));

session.alterTable("mytable", settings).join().expectSuccess();
```

Включение вытеснения во внешнее S3-совместимое хранилище

> **Важно**
>
> Поддерживается только для колоночных таблиц. Поддержка функциональности для строковых таблиц находится в разработке.

Для включения вытеснения требуется объект external data source, описывающий подключение к внешнему хранилищу. Создание объекта external data source возможно через YQL и YDB CLI.

В следующем примере строки таблицы `mytable` будут переноситься в бакет, описанный во внешнем источнике данных `/Root/s3_cold_data`, спустя час после наступления времени, записанного в колонке `created_at`, а спустя 24 часа будут удаляться:

C++

```
session.AlterTable(
    "mytable",
    TAlterTableSettings()
        .BeginAlterTtlSettings()
            .Set("created_at", {
                    TTtlTierSettings(TDuration::Hours(1), TTtlEvictToExternalStorageAction("/Root/s3_cold_dat
a")),
                    TTtlTierSettings(TDuration::Hours(24), TTtlDeleteAction("/Root/s3_cold_data"))
                })
        .EndAlterTtlSettings()
);
```

JavaScript

Функциональность на данный момент не поддерживается.

Go

Функциональность на данный момент не поддерживается.

Python

Функциональность на данный момент не поддерживается.

C#

Функциональность на данный момент не поддерживается.

Java

Функциональность на данный момент не поддерживается.

Включение TTL для вновь создаваемой таблицы

Для вновь создаваемой таблицы можно передать настройки TTL вместе с ее описанием:

C++

```
session.CreateTable(
    "mytable",
    TTableBuilder()
        .AddNullableColumn("id", EPrimitiveType::Uint64)
        .AddNullableColumn("expire_at", EPrimitiveType::Timestamp)
        .SetPrimaryKeyColumn("id")
        .SetTtlSettings("expire_at")
        .Build()
);
```

Go

```
err := session.CreateTable(ctx, "mytable",
  options.WithColumn("id", types.Optional(types.TypeUint64)),
  options.WithColumn("expire_at", types.Optional(types.TypeTimestamp)),
  options.WithTimeToLiveSettings(
    options.NewTTLSettings().ColumnDateType("expire_at"),
  ),
)
```

Python

```
session.create_table(
    'mytable',
    ydb.TableDescription()
    .with_column(ydb.Column('id', ydb.OptionalType(ydb.DataType.Uint64)))
    .with_column(ydb.Column('expire_at', ydb.OptionalType(ydb.DataType.Timestamp)))
    .with_primary_key('id')
```

```
    .with_ttl(ydb.TtlSettings().with_date_type_column('expire_at'))
)
```

C#

Функциональность на данный момент не поддерживается.

JavaScript

Функциональность на данный момент не поддерживается.

Java

TTL для таблицы задаётся в `TableDescription` при создании. Проверить настройки можно через `describeTable`.

```
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.table.TableClient;
import tech.ydb.table.description.TableDescription;
import tech.ydb.table.description.TableTtl;
import tech.ydb.table.session.SessionRetryContext;
import tech.ydb.table.values.PrimitiveType;

public class TtlCreateTableExample {

    private static final String TABLE_NAME = "mytable";

    public static void main(String[] args) {
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString).build();
             TableClient tableClient = TableClient.newClient(transport).build()) {

            SessionRetryContext retryCtx = SessionRetryContext.create(tableClient).build();
            String tablePath = transport.getDatabase() + "/" + TABLE_NAME;

            TableDescription description = TableDescription.newBuilder()
                    .addNullableColumn("id", PrimitiveType.Uint64)
                    .addNullableColumn("expire_at", PrimitiveType.Timestamp)
                    .setPrimaryKey("id")
                    .setTtlSettings(TableTtl.dateTimeColumn("expire_at", 0))
                    .build();

            retryCtx.supplyStatus(session -> session.createTable(tablePath, description))
                    .join().expectSuccess("create table failed");

            TableTtl ttl = retryCtx.supplyResult(session -> session.describeTable(tablePath))
                    .join().getValue().getTableDescription().getTableTtl();
            System.out.println("TTL column: " + ttl.getColumnName());
        }
    }
}
```

Выключение TTL

C++

```
session.AlterTable(
    "mytable",
    TAlterTableSettings()
        .BeginAlterTtlSettings()
            .Drop()
        .EndAlterTtlSettings()
);
```

Go

```
err := session.AlterTable(ctx, "mytable",
  options.WithDropTimeToLive(),
)
```

Python

```
session.alter_table('mytable', drop_ttl_settings=True)
```

C#

Функциональность на данный момент не поддерживается.

JavaScript

Функциональность на данный момент не поддерживается.

Java

```
AlterTableSettings settings = new AlterTableSettings()
        .setTableTtl(TableTtl.notSet());
```

```
session.alterTable("mytable", settings).join().expectSuccess();
```

Получение настроек TTL

Текущие настройки TTL можно получить из описания таблицы:

C++

```
auto desc = session.DescribeTable("mytable").GetValueSync().GetTableDescription();
auto ttl = desc.GetTtlSettings();
```

Go

```
desc, err := session.DescribeTable(ctx, "mytable")
if err != nil {
  // process error
}
ttl := desc.TimeToLiveSettings
```

Python

```
desc = session.describe_table('mytable')
ttl = desc.ttl_settings
```

C#

Функциональность на данный момент не поддерживается.

JavaScript

Функциональность на данный момент не поддерживается.

Java

```
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.table.TableClient;
import tech.ydb.table.description.TableTtl;
import tech.ydb.table.session.SessionRetryContext;

public class TtlDescribeExample {

    public static void main(String[] args) {
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString).build();
             TableClient tableClient = TableClient.newClient(transport).build()) {

            SessionRetryContext retryCtx = SessionRetryContext.create(tableClient).build();
            String tablePath = transport.getDatabase() + "/mytable";

            TableTtl ttl = retryCtx.supplyResult(session -> session.describeTable(tablePath))
                    .join().getValue().getTableDescription().getTableTtl();

            System.out.println("TTL enabled: " + ttl.isEnabled());
            System.out.println("TTL column: " + ttl.getColumnName());
        }
    }
}
```

Векторный поиск

В данном разделе содержатся рецепты кода на разных языках программирования для решения задач векторного поиска с использованием YDB SDK.

Подробно будут разобраны операции:

- Векторный поиск
  - Подключение к YDB
  - Создание таблицы
  - Вставка векторов
  - Добавление индекса
  - Поиск по вектору
  - Итоговый пример

В данном рецепте будет создано хранилище текстов со следующей структурой:

Поле	Пояснение
`id`	Идентификатор текста
`document`	Текст
`embedding`	Векторное представление текста

В рецепте предполагается, что `embedding` уже имеется.

Подключение к YDB

В данной секции описаны минимально необходимые действия для выполнения запросов в YDB.
Для получения более подробной информации о подключении к YDB обратитесь к статье Инициализация драйвера.

C++

```
#include <ydb-cpp-sdk/client/query/client.h>

int main() {
    auto driverConfig = NYdb::CreateFromEnvironment(endpoint + "/?database=" + database);
    NYdb::TDriver driver(driverConfig);
    NYdb::NQuery::TQueryClient client(driver);

    // ...
}
```

Go

```
package main

import (
    "context"
    "os"

    "github.com/ydb-platform/ydb-go-sdk/v3"
)

func main() {
    ctx, cancel := context.WithCancel(context.Background())
    defer cancel()
    db, err := ydb.Open(ctx,
        os.Getenv("YDB_CONNECTION_STRING"),
    )
    if err != nil {
        panic(err)
    }
    defer db.Close(ctx)
}
```

Java

Для выполнения YQL-запросов используйте `QueryClient` — клиент YQL Query Service. Повторные попытки при временных ошибках обеспечивает `SessionRetryContext`, который создаётся на базе `QueryClient`. Подробнее о настройке транспорта и клиентов — в статье Инициализация драйвера.

Ниже — минимальный запускаемый пример: подключение, создание `SessionRetryContext` и проверка соединения запросом `SELECT 1`. Объект `retryCtx` понадобится на следующих шагах рецепта.

```
import tech.ydb.common.transaction.TxMode;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;

public class VectorSearchConnectExample {

    public static void main(String[] args) {
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");
```

```
        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString).build();
             QueryClient queryClient = QueryClient.newClient(transport).build()) {

            // SessionRetryContext автоматически повторяет запрос при временных сбоях.
            SessionRetryContext retryCtx = SessionRetryContext.create(queryClient).build();

            // Проверка подключения — в следующих шагах используется тот же retryCtx.
            retryCtx.supplyResult(session -> QueryReader.readFrom(
                    session.createQuery("SELECT 1", TxMode.NONE, Params.empty())
            )).join().getValue();
        }
    }
}
```

Python

Native SDK

Для выполнения запросов необходимо создать `ydb.QuerySessionPool`.

```
import ydb

driver = ydb.Driver(
    endpoint=ydb_endpoint,
    database=ydb_database,
    credentials=ydb_credentials,
)
driver.wait(5, fail_fast=True)
pool = ydb.QuerySessionPool(driver)
```

Native SDK (Asyncio)

Для выполнения запросов необходимо создать `ydb.aio.QuerySessionPool`:

```
import asyncio
import ydb

async def main():
    async with ydb.aio.Driver(
        endpoint=ydb_endpoint,
        database=ydb_database,
        credentials=ydb_credentials,
    ) as driver:
        await driver.wait(5, fail_fast=True)
        pool = ydb.aio.QuerySessionPool(driver)
        # ... используйте pool ...

asyncio.run(main())
```

C#

Примеры векторного поиска для .NET SDK пока недоступны.

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку: ydb-dotnet-sdk#657

JavaScript

```
import { Driver } from '@ydbjs/core'
import { query, unsafe, identifier } from '@ydbjs/query'

const driver = new Driver('grpc://localhost:2136/local')
await driver.ready()
const sql = query(driver)
```

Rust

```
use ydb::{ClientBuilder, YdbResult};

let client = ClientBuilder::new_from_connection_string(connection_string)?.client()?;
client.wait().await?;
let mut qc = client.query_client();
```

PHP

Примеры векторного поиска для PHP SDK пока недоступны.

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку: ydb-php-sdk#268

Создание таблицы

Сначала необходимо создать таблицу для хранения документов и их векторных представлений.

Структура таблицы:

Название столбца	Тип данных	Пояснение
`id`	`Utf8`	идентификатор документа
`document`	`Utf8`	текст документа
`embedding`	`String`	векторное представление документа

> **Важно**
>
> Для хранения вектора используется тип `String`. Подробнее смотрите в документации по точному векторному поиску.

C++

```
void CreateVectorTable(NYdb::NQuery::TQueryClient& client, const std::string& tableName)
{
    std::string query = std::format(R"(
        CREATE TABLE IF NOT EXISTS `{}` (
            id Utf8,
            document Utf8,
            embedding String,
            PRIMARY KEY (id)
        ))", tableName);

    NYdb::NStatusHelpers::ThrowOnError(client.RetryQuerySync([&](NYdb::NQuery::TSession session) {
        return session.ExecuteQuery(query, NYdb::NQuery::TTxControl::NoTx()).ExtractValueSync();
    }));

    std::cout << "Vector table created: " << tableName << std::endl;
}
```

Go

```
func createVectorTable(ctx context.Context, db *ydb.Driver, tableName string) error {
    query := fmt.Sprintf(`
        CREATE TABLE IF NOT EXISTS %s (
            id Utf8,
            document Utf8,
            embedding String,
            PRIMARY KEY (id)
        );`, "`"+tableName+"`")

    return db.Query().Exec(ctx, query)
}
```

Java

Создайте таблицу для документов и их векторных представлений. Используйте `SessionRetryContext` из раздела Подключение к YDB.

```
import tech.ydb.common.transaction.TxMode;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;

void createVectorTable(SessionRetryContext retryCtx, String tableName) {
    String query = String.format("""
            CREATE TABLE IF NOT EXISTS `%s` (
                id Utf8,
                document Utf8,
                embedding String,
                PRIMARY KEY (id)
            );""", tableName);

    retryCtx.supplyResult(session -> QueryReader.readFrom(
            session.createQuery(query, TxMode.NONE, Params.empty())
    )).join().getValue();

    System.out.println("Vector table created: " + tableName);
}
```

Python

Native SDK

```
import ydb

def create_vector_table(pool: ydb.QuerySessionPool, table_name: str) -> None:
    query = f"""
    CREATE TABLE IF NOT EXISTS `{table_name}` (
        id Utf8,
        document Utf8,
        embedding String,
        PRIMARY KEY (id)
    );"""
```

```
    pool.execute_with_retries(query)

    print(f"Vector table {table_name} created")
```

Native SDK (Asyncio)

```
import ydb

async def create_vector_table(pool: ydb.aio.QuerySessionPool, table_name: str) -> None:
    query = f"""
    CREATE TABLE IF NOT EXISTS `{table_name}` (
        id Utf8,
        document Utf8,
        embedding String,
        PRIMARY KEY (id)
    );"""

    await pool.execute_with_retries(query)

    print(f"Vector table {table_name} created")
```

C#

Примеры векторного поиска для .NET SDK пока недоступны.

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку: ydb-dotnet-sdk#657

JavaScript

```
await sql`CREATE TABLE IF NOT EXISTS `table_name` (
    id Utf8,
    document Utf8,
    embedding String,
    PRIMARY KEY (id)
);`
```

Rust

```
qc.exec(format!(
    "CREATE TABLE IF NOT EXISTS `{table_name}` (
        id Utf8,
        document Utf8,
        embedding String,
        PRIMARY KEY (id)
    );", tableName,
))
.await?;
```

PHP

Примеры векторного поиска для PHP SDK пока недоступны.

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку: ydb-php-sdk#268

Вставка векторов

Для вставки векторов необходимо подготовить и выполнить правильный YQL-запрос. Для унификации вставки разных данных он параметризован.

Запрос оперирует контейнерным типом данных `List<Struct<...>>` (список структур), что позволяет передавать через параметры произвольное количество объектов за один раз.

В YDB таблицах же вектора хранятся в виде сериализованной последовательности байт. Конвертацию в такое представление **рекомендуется выполнять на клиенте**. Альтернативный способ — делегировать конвертацию на сервер с помощью функции преобразования Knn UDF. Ниже будут приведены примеры, демонстрирующие оба подхода.

C++

Рекомендуемый способ

Для сериализации вектора используйте `NYdb::NValueHelpers::Embedding` из C++ SDK версии v3.24.0 или новее.

```
#include <ydb-cpp-sdk/client/value/embedding.h>

void InsertItemsAsBytes(
    NYdb::NQuery::TQueryClient& client,
    const std::string& tableName,
    const std::vector<TItem>& items)
{
    std::string query = std::format(R"(
        DECLARE $items AS List<Struct<
            id: Utf8,
            document: Utf8,
            embedding: Bytes
        >>;
```

```
        UPSERT INTO `{0}`
        (
            id,
            document,
            embedding
        )
        SELECT
            id,
            document,
            embedding,
        FROM AS_TABLE($items);
    )", tableName);

    NYdb::TParamsBuilder paramsBuilder;
    auto& valueBuilder = paramsBuilder.AddParam("$items");
    valueBuilder.BeginList();
    for (const auto& item : items) {
        valueBuilder.AddListItem();
        valueBuilder.BeginStruct();
        valueBuilder.AddMember("id").Utf8(item.Id);
        valueBuilder.AddMember("document").Utf8(item.Document);
        valueBuilder.AddMember("embedding", NYdb::NValueHelpers::Embedding(item.Embedding));
        valueBuilder.EndStruct();
    }
    valueBuilder.EndList();
    valueBuilder.Build();

    NYdb::NStatusHelpers::ThrowOnError(client.RetryQuerySync([params = paramsBuilder.Build(), &query](NYdb::N
Query::TSession session) {
        return session.ExecuteQuery(query, NYdb::NQuery::TTxControl::BeginTx(NYdb::NQuery::TTxSettings::Seria
lizableRW()).CommitTx(), params).ExtractValueSync();
    }));

    std::cout << items.size() << " items inserted" << std::endl;
}
```

Альтернативный способ

> **Важно**
>
> Передача вектора как `List<Float>` с преобразованием на стороне YQL через `Knn::ToBinaryStringFloat` даёт худшую производительность, чем кодирование byte array на клиенте.

```
void InsertItemsAsFloatList(
    NYdb::NQuery::TQueryClient& client,
    const std::string& tableName,
    const std::vector<TItem>& items)
{
    std::string query = std::format(R"(
        DECLARE $items AS List<Struct<
            id: Utf8,
            document: Utf8,
            embedding: List<Float>
        >>;

        UPSERT INTO `{}`
        (
            id,
            document,
            embedding
        )
        SELECT
            id,
            document,
            Untag(Knn::ToBinaryStringFloat(embedding), "FloatVector"),
        FROM AS_TABLE($items);
    )", tableName);

    NYdb::TParamsBuilder paramsBuilder;
    auto& valueBuilder = paramsBuilder.AddParam("$items");
    valueBuilder.BeginList();
    for (const auto& item : items) {
        valueBuilder.AddListItem();
        valueBuilder.BeginStruct();
        valueBuilder.AddMember("id").Utf8(item.Id);
        valueBuilder.AddMember("document").Utf8(item.Document);
        valueBuilder.AddMember("embedding").BeginList();
        for (const auto& value : item.Embedding) {
            valueBuilder.AddListItem().Float(value);
        }
        valueBuilder.EndList();
        valueBuilder.EndStruct();
    }
    valueBuilder.EndList();
    valueBuilder.Build();

    NYdb::NStatusHelpers::ThrowOnError(client.RetryQuerySync([params = paramsBuilder.Build(), &query](NYdb::N
Query::TSession session) {
        return session.ExecuteQuery(query, NYdb::NQuery::TTxControl::BeginTx(NYdb::NQuery::TTxSettings::Seria
```

```
lizableRW()).CommitTx(), params).ExtractValueSync();
    }));

    std::cout << items.size() << " items inserted" << std::endl;
}
```

Go

Для сериализации вектора используйте `sugar.Embedding` из Go SDK версии v3.153.0 или новее. Затем выполните параметризованный запрос:

```
import "github.com/ydb-platform/ydb-go-sdk/v3/sugar"

func insertItems(ctx context.Context, db *ydb.Driver, tableName string, items []Item) error {
    query := fmt.Sprintf(`
    DECLARE $items AS List<Struct<
        id: Utf8,
        document: Utf8,
        embedding: String
    >>;

    UPSERT INTO %s
    (id, document, embedding)
    SELECT id, document, embedding
    FROM AS_TABLE($items);
    `, "`"+tableName+"`")

    rows := make([]types.Value, 0, len(items))
    for _, item := range items {
    rows = append(rows, types.StructValue(
        types.StructFieldValue("id", types.UTF8Value(item.ID)),
        types.StructFieldValue("document", types.UTF8Value(item.Document)),
        types.StructFieldValue("embedding", sugar.Embedding(item.Embedding...)),
    ))
    }

    return db.Query().Exec(ctx, query,
    query.WithParameters(
        ydb.ParamsBuilder().Param("$items").BeginList().AddItems(rows...).EndList().Build(),
    ),
    )
}
```

Java

Рекомендуемый способ

```
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.util.ArrayList;
import java.util.List;

import tech.ydb.common.transaction.TxMode;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;
import tech.ydb.table.values.ListType;
import tech.ydb.table.values.ListValue;
import tech.ydb.table.values.PrimitiveType;
import tech.ydb.table.values.PrimitiveValue;
import tech.ydb.table.values.StructType;
import tech.ydb.table.values.Value;

byte[] convertVectorToBytes(float[] vector) {
    ByteBuffer bb = ByteBuffer.allocate(vector.length * Float.BYTES + 1).order(ByteOrder.LITTLE_ENDIAN);
    for (float v : vector) {
        bb.putFloat(v);
    }
    bb.put((byte) 0x01);
    return bb.array();
}

void insertItemsAsBytes(SessionRetryContext retryCtx, String tableName, List<Item> items) {
    String query = String.format("""
            DECLARE $items AS List<Struct<
                id: Utf8,
                document: Utf8,
                embedding: String
            >>;

            UPSERT INTO `%s`
            (
                id,
                document,
                embedding
            )
            SELECT
                id,
                document,
                embedding,
            FROM AS_TABLE($items);""", tableName);
```

```
    StructType rowType = StructType.of(
            "id", PrimitiveType.Text,
            "document", PrimitiveType.Text,
            "embedding", PrimitiveType.Bytes
    );

    List<Value<?>> rows = new ArrayList<>(items.size());
    for (Item item : items) {
        rows.add(rowType.newValue(
                "id", PrimitiveValue.newText(item.id()),
                "document", PrimitiveValue.newText(item.document()),
                "embedding", PrimitiveValue.newBytes(convertVectorToBytes(item.embedding()))
        ));
    }

    ListValue itemsParam = ListType.of(rowType).newValue(rows);
    Params params = Params.of("$items", itemsParam);

    retryCtx.supplyResult(session -> QueryReader.readFrom(
            session.createQuery(query, TxMode.SERIALIZABLE_RW, params)
    )).join().getValue();

    System.out.println(items.size() + " items inserted");
}
```

Примечание

В функции `ConvertVectorToBytes` подразумевается, что на клиенте используется процессор с little-endian порядком байт, например x86_64. Если используется другой порядок байт, функцию `ConvertVectorToBytes` необходимо адаптировать.

Альтернативный способ

Важно

Передача вектора как `List<Float>` с преобразованием на стороне YQL через `Knn::ToBinaryStringFloat` даёт худшую производительность, чем кодирование byte array на клиенте.

Вариант с передачей компонентов вектора как `List<Float>` и преобразованием на стороне YQL через `Knn::ToBinaryStringFloat` — по той же схеме, что для альтернативных примеров на Python и C++ ниже:

```
import java.util.ArrayList;
import java.util.List;

import tech.ydb.common.transaction.TxMode;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;
import tech.ydb.table.values.ListType;
import tech.ydb.table.values.ListValue;
import tech.ydb.table.values.PrimitiveType;
import tech.ydb.table.values.PrimitiveValue;
import tech.ydb.table.values.StructType;
import tech.ydb.table.values.Value;

void insertItemsAsFloatList(SessionRetryContext retryCtx, String tableName, List<Item> items) {
    String query = String.format("""
            DECLARE $items AS List<Struct<
                id: Utf8,
                document: Utf8,
                embedding: List<Float>
            >>;

            UPSERT INTO `%s`
            (
                id,
                document,
                embedding
            )
            SELECT
                id,
                document,
                Untag(Knn::ToBinaryStringFloat(embedding), "FloatVector"),
            FROM AS_TABLE($items);""", tableName);

    StructType rowType = StructType.of(
            "id", PrimitiveType.Text,
            "document", PrimitiveType.Text,
            "embedding", ListType.of(PrimitiveType.Float)
    );

    List<Value<?>> rows = new ArrayList<>(items.size());
    for (Item item : items) {
        Value<?>[] emb = new Value<?>[item.embedding().length];
        for (int i = 0; i < item.embedding().length; i++) {
```

```
            emb[i] = PrimitiveValue.newFloat(item.embedding()[i]);
        }
        ListValue embList = ListType.of(PrimitiveType.Float).newValueOwn(emb);
        rows.add(rowType.newValue(
                "id", PrimitiveValue.newText(item.id()),
                "document", PrimitiveValue.newText(item.document()),
                "embedding", embList
        ));
    }

    ListValue itemsParam = ListType.of(rowType).newValue(rows);
    Params params = Params.of("$items", itemsParam);

    retryCtx.supplyResult(session -> QueryReader.readFrom(
            session.createQuery(query, TxMode.SERIALIZABLE_RW, params)
    )).join().getValue();

    System.out.println(items.size() + " items inserted");
}
```

Python

Рекомендуемый способ

Для сериализации векторов используйте `ydb.convert_floats_to_embedding_bytes` из Python SDK версии 3.33.1 или новее.

Метод принимает массив словарей `items`, где каждый словарь содержит поля `id` - идентификатор, `document` - текст, `embedding` - векторное представление текста, заранее сериализованное в последовательность байт.

Для использования структуры в примере ниже создается `items_struct_type = ydb.StructType()`, в котором задаются типы всех полей. Для передачи списка таких структур его необходимо обернуть в `ydb.ListType`: `ydb.ListType(items_struct_type)`.

Native SDK

```
import ydb

def insert_items_vector_as_bytes(
    pool: ydb.QuerySessionPool,
    table_name: str,
    items: list[dict],
) -> None:
    query = f"""
    DECLARE $items AS List<Struct<
        id: Utf8,
        document: Utf8,
        embedding: String
    >>;

    UPSERT INTO `{table_name}`
    (
        id,
        document,
        embedding
    )
    SELECT
        id,
        document,
        embedding,
    FROM AS_TABLE($items);
    """

    items_struct_type = ydb.StructType()
    items_struct_type.add_member("id", ydb.PrimitiveType.Utf8)
    items_struct_type.add_member("document", ydb.PrimitiveType.Utf8)
    items_struct_type.add_member("embedding", ydb.PrimitiveType.String)

    for item in items:
        item["embedding"] = ydb.convert_floats_to_embedding_bytes(item["embedding"])

    pool.execute_with_retries(
        query, {"$items": (items, ydb.ListType(items_struct_type))}
    )

    print(f"{len(items)} items inserted")
```

Native SDK (Asyncio)

```
import ydb

async def insert_items_vector_as_bytes(
    pool: ydb.aio.QuerySessionPool,
    table_name: str,
    items: list[dict],
) -> None:
    query = f"""
    DECLARE $items AS List<Struct<
        id: Utf8,
        document: Utf8,
```

```
        embedding: String
    >>;

    UPSERT INTO `{table_name}`
    (
        id,
        document,
        embedding
    )
    SELECT
        id,
        document,
        embedding,
    FROM AS_TABLE($items);
    """

    items_struct_type = ydb.StructType()
    items_struct_type.add_member("id", ydb.PrimitiveType.Utf8)
    items_struct_type.add_member("document", ydb.PrimitiveType.Utf8)
    items_struct_type.add_member("embedding", ydb.PrimitiveType.String)

    for item in items:
        item["embedding"] = ydb.convert_floats_to_embedding_bytes(item["embedding"])

    await pool.execute_with_retries(
        query, {"$items": (items, ydb.ListType(items_struct_type))}
    )

    print(f"{len(items)} items inserted")
```

Альтернативный способ

> **Важно**
>
> Передача вектора как `List<Float>` с преобразованием на стороне YQL через `Knn::ToBinaryStringFloat` даёт худшую производительность, чем кодирование byte array на клиенте.

Метод принимает массив словарей `items`, где каждый словарь содержит поля `id` - идентификатор, `document` - текст, `embedding` - векторное представление текста.

Для использования структуры в примере ниже создается `items_struct_type = ydb.StructType()`, в котором задаются типы всех полей. Для передачи списка таких структур его необходимо обернуть в `ydb.ListType`: `ydb.ListType(items_struct_type)`.

Native SDK

```
import ydb

def insert_items_vector_as_float_list(
    pool: ydb.QuerySessionPool,
    table_name: str,
    items: list[dict],
) -> None:
    query = f"""
    DECLARE $items AS List<Struct<
        id: Utf8,
        document: Utf8,
        embedding: List<Float>
    >>;

    UPSERT INTO `{table_name}`
    (
        id,
        document,
        embedding
    )
    SELECT
        id,
        document,
        Untag(Knn::ToBinaryStringFloat(embedding), "FloatVector"),
    FROM AS_TABLE($items);
    """

    items_struct_type = ydb.StructType()
    items_struct_type.add_member("id", ydb.PrimitiveType.Utf8)
    items_struct_type.add_member("document", ydb.PrimitiveType.Utf8)
    items_struct_type.add_member("embedding", ydb.ListType(ydb.PrimitiveType.Float))

    pool.execute_with_retries(
        query, {"$items": (items, ydb.ListType(items_struct_type))}
    )

    print(f"{len(items)} items inserted")
```

Native SDK (Asyncio)

```
import ydb
```

```python
async def insert_items_vector_as_float_list(
    pool: ydb.aio.QuerySessionPool,
    table_name: str,
    items: list[dict],
) -> None:
    query = f"""
    DECLARE $items AS List<Struct<
        id: Utf8,
        document: Utf8,
        embedding: List<Float>
    >>;

    UPSERT INTO `{table_name}`
    (
        id,
        document,
        embedding
    )
    SELECT
        id,
        document,
        Untag(Knn::ToBinaryStringFloat(embedding), "FloatVector"),
    FROM AS_TABLE($items);
    """

    items_struct_type = ydb.StructType()
    items_struct_type.add_member("id", ydb.PrimitiveType.Utf8)
    items_struct_type.add_member("document", ydb.PrimitiveType.Utf8)
    items_struct_type.add_member("embedding", ydb.ListType(ydb.PrimitiveType.Float))

    await pool.execute_with_retries(
        query, {"$items": (items, ydb.ListType(items_struct_type))}
    )

    print(f"{len(items)} items inserted")
```

C#

Примеры векторного поиска для .NET SDK пока недоступны.

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку: ydb-dotnet-sdk#657

JavaScript

Рекомендуемый способ

```javascript
function convertVectorToBytes(vector) {
    const bytes = new Uint8Array(vector.length * 4 + 1);
    const view = new DataView(bytes.buffer);

    for (let i = 0; i < vector.length; i++) {
        view.setFloat32(i * 4, vector[i], true);
    }

    bytes[bytes.length - 1] = 0x01;
    return bytes;
}

const items = [
    {
    id: "first_doc",
    document: "My Document",
    embedding: convertVectorToBytes(new Float32Array([1.5, 2.5, 3.5]))
    }
]

await sql`
    UPSERT INTO `table_name` (id, document, embedding)
    SELECT id, document, embedding,
    FROM AS_TABLE($items);`
```

Альтернативный способ

> **Важно**
>
> Передача вектора как `List<Float>` с преобразованием на стороне YQL через `Knn::ToBinaryStringFloat` даёт худшую производительность, чем кодирование byte array на клиенте.

```javascript
const items = [
    {
    id: "first_doc",
    document: "My Document",
    embedding: new Float32Array([1.5, 2.5, 3.5])
    }
]
```

```
await sql`
    UPSERT INTO `table_name` (id, document, embedding)
    SELECT id, document, Untag(Knn::ToBinaryStringFloat(embedding), "FloatVector"),
    FROM AS_TABLE($items);`
```

Rust

```
use ydb::{Bytes, Value, ydb_struct};

fn convert_vector_to_bytes(vector: &[f32]) -> Bytes {
    let mut buf = Vec::with_capacity(vector.len() * 4 + 1);
    for v in vector {
        buf.extend_from_slice(&v.to_le_bytes());
    }
    buf.push(0x01);
    Bytes::from(buf)
}

let example = ydb_struct!(
    "id" => "",
    "document" => "",
    "embedding" => Bytes::default(),
);
let rows: Vec<Value> = items
    .iter()
    .map(|item| ydb_struct!(
        "id" => item.id,
        "document" => item.document,
        "embedding" => convert_vector_to_bytes(&item.embedding),
    ))
    .collect();
let list = Value::list_from(example, rows)?;

qc.exec(insert_query).param("$items", list).await?;
```

PHP

Примеры векторного поиска для PHP SDK пока недоступны.

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку: ydb-php-sdk#268

Добавление индекса

Использование векторного индекса позволяет эффективно решать задачу приближённого поиска ближайших векторов. Подробнее о преимуществах и особенностях использования описано в документации по векторному индексу.

Для добавления индекса необходимо выполнить две операции:

1. Создать временный индекс;
2. Сохранить временный индекс как постоянный.

Такой подход позволяет создавать индекс как при его первоначальном создании, так и при перестроении (если индекс уже существует).

Доступные стратегии:

- `similarity=cosine`;
- `similarity=inner_product`;
- `distance=cosine`;
- `distance=euclidean`;
- `distance=manhattan`.

Каждая стратегия определяет функцию, которая будет использоваться для последующего поиска. Более подробно функции описаны в документации по функциям расстояния и сходства.

Параметры, применяемые при создании индекса типа `vector_kmeans_tree`, описаны в документации векторного индекса.

C++

```
void AddIndex(
    NYdb::TDriver& driver,
    NYdb::NQuery::TQueryClient& client,
    const std::string& database,
    const std::string& tableName,
    const std::string& indexName,
    const std::string& strategy)
{
    std::string query = std::format(R"(
        ALTER TABLE `{0}`
        ADD INDEX {1}__temp
        GLOBAL USING vector_kmeans_tree
        ON (embedding)
        WITH (
            {2}
        );
    )", tableName, indexName, strategy);
```

```
    NYdb::NStatusHelpers::ThrowOnError(client.RetryQuerySync([&](NYdb::NQuery::TSession session) {
        return session.ExecuteQuery(query, NYdb::NQuery::TTxControl::NoTx()).ExtractValueSync();
    }));

    NYdb::NTable::TTableClient tableClient(driver);
    NYdb::NStatusHelpers::ThrowOnError(tableClient.RetryOperationSync([&](NYdb::NTable::TSession session) {
        return session.AlterTable(database + "/" + tableName, NYdb::NTable::TAlterTableSettings()
            .AppendRenameIndexes(NYdb::NTable::TRenameIndex{
                .SourceName_ = indexName + "__temp",
                .DestinationName_ = indexName,
                .ReplaceDestination_ = true
            })
        ).ExtractValueSync();
    }));

    std::cout << "Table index `" << indexName << "` for table `" << tableName << "` added" << std::endl;
}
```

Go

```
func addVectorIndex(
    ctx context.Context,
    db *ydb.Driver,
    tableName, indexName, strategy string
) error {
    tempIndexName := indexName + "__temp"
    query := fmt.Sprintf(`
        ALTER TABLE %s
        ADD INDEX %s
        GLOBAL USING vector_kmeans_tree
        ON (embedding)
        WITH (
        %s
        );
    `, "`"+tableName+"`", tempIndexName, strategy)

    if err := db.Query().Exec(ctx, query); err != nil {
        return err
    }

    return db.Table().Do(ctx, func(ctx context.Context, s table.Session) error {
        return s.AlterTable(ctx, path.Join(db.Name(), tableName),
        options.WithRenameIndex(tempIndexName, indexName, true),
        )
    })
}
```

Java

Создайте векторный индекс для таблицы из предыдущих шагов. YQL-запрос `ALTER TABLE ... ADD INDEX` выполняется через `queryRetry`; переименование временного индекса — через `TableClient` и `tableRetry` (см. Подключение). Параметры `dimension`, `levels` и `clusters` задают геометрию индекса `vector_kmeans_tree` — как в итоговом примере на Python.

```
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.common.transaction.TxMode;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;
import tech.ydb.table.settings.AlterTableSettings;

static void addVectorIndex(
        GrpcTransport transport,
        SessionRetryContext queryRetry,
        SessionRetryContext tableRetry,
        String tableName,
        String indexName,
        String strategy,
        int dimension,
        int levels,
        int clusters) {

    String tempIndexName = indexName + "__temp";
    String query = String.format("""
            ALTER TABLE `%s`
            ADD INDEX %s
            GLOBAL USING vector_kmeans_tree
            ON (embedding)
            WITH (
                %s,
                vector_type="Float",
                vector_dimension=%d,
                levels=%d,
                clusters=%d
            );
            """, tableName, tempIndexName, strategy, dimension, levels, clusters);

    // Шаг 1: создать временный индекс через YQL Query Service.
    queryRetry.supplyResult(session -> QueryReader.readFrom(
            session.createQuery(query, TxMode.NONE, Params.empty())
    )).join().getValue();
```

```
    // Шаг 2: переименовать временный индекс в постоянный через Table API.
    String tablePath = transport.getDatabase() + "/" + tableName;
    AlterTableSettings settings = new AlterTableSettings()
            .addRenameIndex(tempIndexName, indexName, true);

    tableRetry.supplyStatus(session -> session.alterTable(tablePath, settings))
            .join()
            .expectSuccess("alter table rename index");

    System.out.println("Table index `" + indexName + "` for table `" + tableName + "` added");
}
```

Python

Native SDK

```
import ydb

def add_vector_index(
    pool: ydb.QuerySessionPool,
    driver: ydb.Driver,
    table_name: str,
    index_name: str,
    strategy: str,
):
    temp_index_name = f"{index_name}__temp"
    query = f"""
    ALTER TABLE `{table_name}`
    ADD INDEX {temp_index_name}
    GLOBAL USING vector_kmeans_tree
    ON (embedding)
    WITH (
        {strategy}
    );
    """

    pool.execute_with_retries(query)
    driver.table_client.alter_table(
        f"{driver._driver_config.database}/{table_name}",
        rename_indexes=[
            ydb.RenameIndexItem(
                source_name=temp_index_name,
                destination_name=f"{index_name}",
                replace_destination=True,
            ),
        ],
    )
    print(f"Table index {index_name} created.")
```

Native SDK (Asyncio)

```
import ydb

async def add_vector_index(
    pool: ydb.aio.QuerySessionPool,
    driver: ydb.aio.Driver,
    table_name: str,
    index_name: str,
    strategy: str
):
    temp_index_name = f"{index_name}__temp"
    query = f"""
    ALTER TABLE `{table_name}`
    ADD INDEX {temp_index_name}
    GLOBAL USING vector_kmeans_tree
    ON (embedding)
    WITH (
        {strategy}
    );
    """

    await pool.execute_with_retries(query)
    await driver.table_client.alter_table(
        f"{driver._driver_config.database}/{table_name}",
        rename_indexes=[
            ydb.RenameIndexItem(
                source_name=temp_index_name,
                destination_name=f"{index_name}",
                replace_destination=True,
            ),
        ],
    )

    print(f"Table index {index_name} created.")
```

C#

Примеры векторного поиска для .NET SDK пока недоступны.

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку: ydb-dotnet-sdk#657

JavaScript

Пример добавления векторного индекса и полная программа для JavaScript SDK пока недоступны.

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку: ydb-js-sdk#627

Rust

```
let temp_index_name = format!("{index_name}__temp");
qc.exec(format!(
    "ALTER TABLE `{table_name}`
    ADD INDEX {temp_index_name}
    GLOBAL USING vector_kmeans_tree
    ON (embedding)
    WITH ({strategy}, vector_type=\"Float\", vector_dimension={dimension},
            levels={levels}, clusters={clusters});"
))
.await?;
qc.exec(format!(
    "ALTER TABLE `{table_name}` RENAME INDEX `{temp_index_name}` TO `{index_name}`;"
))
.await?;
```

PHP

Примеры векторного поиска для PHP SDK пока недоступны.

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку: ydb-php-sdk#268

Поиск по вектору

Для поиска документов по вектору используется специальный YQL-запрос, в котором необходимо определить функцию сходства или расстояния.
Доступные значения:

- `CosineSimilarity`;
- `InnerProductSimilarity`;
- `CosineDistance`;
- `ManhattanDistance`;
- `EuclideanDistance`.

Подробнее функции описаны в документации по функциям расстояния и сходства.

Метод позволяет указать имя индекса. Если оно задано, в запрос будет добавлено выражение `VIEW index_name`, что позволит использовать векторный индекс при поиске.

Метод возвращает список, состоящий из словарей с полями `id`, `document`, а также `score` — числом, отражающим степень сходства (или расстояния) с искомым вектором.

C++

Рекомендуемый способ

```
#include <ydb-cpp-sdk/client/value/embedding.h>

std::vector<TResultItem> SearchItemsAsBytes(
    NYdb::NQuery::TQueryClient& client,
    const std::string& tableName,
    const std::vector<float>& embedding,
    const std::string& strategy,
    std::uint64_t limit,
    std::uint64_t topClusters = 10,
    const std::optional<std::string>& indexName = std::nullopt)
{
    std::string viewIndex = indexName ? "VIEW " + *indexName : "";
    std::string sortOrder = strategy.ends_with("Similarity") ? "DESC" : "ASC";

    std::string query = std::format(R"(
        PRAGMA ydb.KMeansTreeSearchTopSize = "{5}";
        DECLARE $embedding as Bytes;
        SELECT
            id,
            document,
            Knn::{2}(embedding, $embedding) as score
        FROM {0} {1}
        ORDER BY score {3}
        LIMIT {4};
    )", tableName, viewIndex, strategy, sortOrder, limit, topClusters);

    auto params = NYdb::TParamsBuilder()
```

```
        .AddParam("$embedding", NYdb::NValueHelpers::Embedding(embedding))
        .Build();

    std::vector<TResultItem> result;

    NYdb::NStatusHelpers::ThrowOnError(client.RetryQuerySync([params, &query, &result](NYdb::NQuery::TSession
session) {
        auto execResult = session.ExecuteQuery(query, NYdb::NQuery::TTxControl::BeginTx(NYdb::NQuery::TTxSett
ings::SerializableRW()).CommitTx(), params).ExtractValueSync();
        if (execResult.IsSuccess()) {
            auto parser = execResult.GetResultSetParser(0);
            while (parser.TryNextRow()) {
                result.push_back({
                    .Id = *parser.ColumnParser(0).GetOptionalUtf8(),
                    .Document = *parser.ColumnParser(1).GetOptionalUtf8(),
                    .Score = *parser.ColumnParser(2).GetOptionalFloat()
                });
            }
        }
        return execResult;
    }));

    return result;
}
```

Альтернативный способ

> **Важно**
>
> Передача вектора как `List<Float>` с преобразованием на стороне YQL через `Knn::ToBinaryStringFloat` даёт худшую производительность, чем кодирование byte array на клиенте.

```
std::vector<TResultItem> SearchItemsAsFloatList(
    NYdb::NQuery::TQueryClient& client,
    const std::string& tableName,
    const std::vector<float>& embedding,
    const std::string& strategy,
    std::uint64_t limit,
    std::uint64_t topClusters = 10,
    const std::optional<std::string>& indexName = std::nullopt)
{
    std::string viewIndex = indexName ? "VIEW " + *indexName : "";
    std::string sortOrder = strategy.ends_with("Similarity") ? "DESC" : "ASC";

    std::string query = std::format(R"(
        PRAGMA ydb.KMeansTreeSearchTopSize = "{5}";
        DECLARE $embedding as List<Float>;

        $TargetEmbedding = Knn::ToBinaryStringFloat($embedding);

        SELECT
            id,
            document,
            Knn::{2}(embedding, $TargetEmbedding) as score
        FROM {0} {1}
        ORDER BY score
        {3}
        LIMIT {4};
    )", tableName, viewIndex, strategy, sortOrder, limit, topClusters);

    NYdb::TParamsBuilder paramsBuilder;
    auto& valueBuilder = paramsBuilder.AddParam("$embedding");
    valueBuilder.BeginList();
    for (auto value : embedding) {
        valueBuilder.AddListItem().Float(value);
    }
    valueBuilder.EndList().Build();

    std::vector<TResultItem> result;

    NYdb::NStatusHelpers::ThrowOnError(client.RetryQuerySync([params = paramsBuilder.Build(), &query, &resul
t](NYdb::NQuery::TSession session) {
        auto execResult = session.ExecuteQuery(query, NYdb::NQuery::TTxControl::BeginTx(NYdb::NQuery::TTxSett
ings::SerializableRW()).CommitTx(), params).ExtractValueSync();
        if (execResult.IsSuccess()) {
            auto parser = execResult.GetResultSetParser(0);
            while (parser.TryNextRow()) {
                result.push_back({
                    .Id = *parser.ColumnParser(0).GetOptionalUtf8(),
                    .Document = *parser.ColumnParser(1).GetOptionalUtf8(),
                    .Score = *parser.ColumnParser(2).GetOptionalFloat()
                });
            }
        }
        return execResult;
    }));
```

```
    return result;
}
```

Go

```
type ResultItem struct {
    ID       string
    Document string
    Score    float32
}

func searchItems(
    ctx context.Context,
    db *ydb.Driver,
    tableName string,
    embedding []float32,
    strategy string,
    limit int,
    indexName string,
) ([]ResultItem, error) {
    viewIndex := ""
    if indexName != "" {
        viewIndex = "VIEW " + indexName
    }
    sortOrder := "DESC"
    if !strings.HasSuffix(strategy, "Similarity") {
        sortOrder = "ASC"
    }
    q := fmt.Sprintf(`
        DECLARE $embedding AS String;
        SELECT id, document, Knn::%s(embedding, $embedding) AS score
        FROM %s %s
        ORDER BY score %s
        LIMIT %d;
        `, strategy, tableName, viewIndex, sortOrder, limit)

    row, err := db.Query().Query(ctx, q,
    query.WithParameters(
        ydb.ParamsBuilder().Param("$embedding").Any(sugar.Embedding(embedding...)).Build(),
    ),
    )
    if err != nil {
        return nil, err
    }
    defer row.Close(ctx)

    var items []ResultItem
    for rs, err := row.NextResultSet(ctx); err == nil; rs, err = row.NextResultSet(ctx) {
        for r, err := rs.NextRow(ctx); err == nil; r, err = rs.NextRow(ctx) {
            var item ResultItem
            if err := r.ScanNamed(
                query.Named("id", &item.ID),
                query.Named("document", &item.Document),
                query.Named("score", &item.Score),
            ); err != nil {
                return nil, err
            }
            items = append(items, item)
        }
    }
    return items, nil
}
```

Java

Найдите ближайшие документы к запросному вектору в таблице, заполненной на шаге Вставка векторов. Передайте имя индекса в `indexName`, чтобы использовать `VIEW` (после шага Добавление индекса). Параметр `topClusters` задаёт `PRAGMA ydb.KMeansTreeSearchTopSize` — как `top_clusters` в Python-примере.

Рекомендуемый способ

```
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

import tech.ydb.common.transaction.TxMode;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;
import tech.ydb.table.result.ResultSetReader;
import tech.ydb.table.values.PrimitiveValue;

byte[] convertVectorToBytes(float[] vector) {
    ByteBuffer bb = ByteBuffer.allocate(vector.length * Float.BYTES + 1).order(ByteOrder.LITTLE_ENDIAN);
    for (float v : vector) {
        bb.putFloat(v);
    }
    bb.put((byte) 0x01);
```

```
    return bb.array();
}

List<ResultItem> searchItemsAsBytes(
        SessionRetryContext retryCtx,
        String tableName,
        float[] embedding,
        String strategy,
        long limit,
        Optional<String> indexName,
        long topClusters) {

    String viewIndex = indexName.map(n -> "VIEW " + n).orElse("");
    String sortOrder = strategy.endsWith("Similarity") ? "DESC" : "ASC";

    String query = String.format("""
            PRAGMA ydb.KMeansTreeSearchTopSize = "%d";
            DECLARE $embedding as String;
            SELECT
                id,
                document,
                Knn::%s(embedding, $embedding) as score
            FROM %s %s
            ORDER BY score %s
            LIMIT %d;
            """, topClusters, strategy, tableName, viewIndex, sortOrder, limit);

    Params params = Params.of("$embedding", PrimitiveValue.newBytes(convertVectorToBytes(embedding)));

    QueryReader reader = retryCtx.supplyResult(session -> QueryReader.readFrom(
            session.createQuery(query, TxMode.SERIALIZABLE_RW, params)
    )).join().getValue();

    List<ResultItem> result = new ArrayList<>();
    ResultSetReader rs = reader.getResultSet(0);
    while (rs.next()) {
        result.add(new ResultItem(
                rs.getColumn("id").getText(),
                rs.getColumn("document").getText(),
                rs.getColumn("score").getFloat()
        ));
    }
    return result;
}
```

Альтернативный способ

> **Важно**
>
> Передача вектора как `List<Float>` с преобразованием на стороне YQL через `Knn::ToBinaryStringFloat` даёт худшую производительность, чем кодирование byte array на клиенте.

Тот же поиск с передачей запросного вектора как `List<Float>`:

```
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

import tech.ydb.common.transaction.TxMode;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;
import tech.ydb.table.result.ResultSetReader;
import tech.ydb.table.values.ListType;
import tech.ydb.table.values.ListValue;
import tech.ydb.table.values.PrimitiveType;
import tech.ydb.table.values.PrimitiveValue;
import tech.ydb.table.values.Value;

List<ResultItem> searchItemsAsFloatList(
        SessionRetryContext retryCtx,
        String tableName,
        float[] embedding,
        String strategy,
        long limit,
        Optional<String> indexName) {

    String viewIndex = indexName.map(n -> "VIEW " + n).orElse("");
    String sortOrder = strategy.endsWith("Similarity") ? "DESC" : "ASC";

    String query = String.format("""
            DECLARE $embedding as List<Float>;

            $target_embedding = Knn::ToBinaryStringFloat($embedding);

            SELECT
                id,
                document,
```

```
                Knn::%s(embedding, $target_embedding) as score
            FROM %s %s
            ORDER BY score
            %s
            LIMIT %d;
            """, strategy, tableName, viewIndex, sortOrder, limit);

    Value<?>[] floats = new Value<?>[embedding.length];
    for (int i = 0; i < embedding.length; i++) {
        floats[i] = PrimitiveValue.newFloat(embedding[i]);
    }
    ListValue emb = ListType.of(PrimitiveType.Float).newValueOwn(floats);
    Params params = Params.of("$embedding", emb);

    QueryReader reader = retryCtx.supplyResult(session -> QueryReader.readFrom(
            session.createQuery(query, TxMode.SERIALIZABLE_RW, params)
    )).join().getValue();

    List<ResultItem> result = new ArrayList<>();
    ResultSetReader rs = reader.getResultSet(0);
    while (rs.next()) {
        result.add(new ResultItem(
                rs.getColumn("id").getText(),
                rs.getColumn("document").getText(),
                rs.getColumn("score").getFloat()
        ));
    }
    return result;
}
```

Python

Рекомендуемый способ

Native SDK

```
import ydb

def search_items_vector_as_bytes(
    pool: ydb.QuerySessionPool,
    table_name: str,
    embedding: list[float],
    strategy: str = "CosineSimilarity",
    limit: int = 1,
    index_name: str | None = None,
    top_clusters: int = 10,
) -> list[dict]:
    view_index = f"VIEW {index_name}" if index_name else ""

    sort_order = "DESC" if strategy.endswith("Similarity") else "ASC"

    query = f"""
    PRAGMA ydb.KMeansTreeSearchTopSize = "{top_clusters}";
    DECLARE $embedding as String;

    SELECT
        id,
        document,
        Knn::{strategy}(embedding, $embedding) as score
    FROM {table_name} {view_index}
    ORDER BY score {sort_order}
    LIMIT {limit};
    """

    result = pool.execute_with_retries(
        query,
        {
            "$embedding": (
                ydb.convert_floats_to_embedding_bytes(embedding),
                ydb.PrimitiveType.String,
            ),
        },
    )

    items = []

    for result_set in result:
        for row in result_set.rows:
            items.append(
                {
                    "id": row["id"],
                    "document": row["document"],
                    "score": row["score"],
                }
            )

    return items
```

Native SDK (Asyncio)

```
import ydb

async def search_items_vector_as_bytes(
    pool: ydb.aio.QuerySessionPool,
    table_name: str,
    embedding: list[float],
    strategy: str = "CosineSimilarity",
    limit: int = 1,
    index_name: str | None = None,
) -> list[dict]:
    view_index = f"VIEW {index_name}" if index_name else ""

    sort_order = "DESC" if strategy.endswith("Similarity") else "ASC"

    query = f"""
    DECLARE $embedding as String;

    SELECT
        id,
        document,
        Knn::{strategy}(embedding, $embedding) as score
    FROM {table_name} {view_index}
    ORDER BY score {sort_order}
    LIMIT {limit};
    """

    result = await pool.execute_with_retries(
        query,
        {
            "$embedding": (
                ydb.convert_floats_to_embedding_bytes(embedding),
                ydb.PrimitiveType.String,
            ),
        },
    )

    items = []

    for result_set in result:
        for row in result_set.rows:
            items.append(
                {
                    "id": row["id"],
                    "document": row["document"],
                    "score": row["score"],
                }
            )

    return items
```

Альтернативный способ

> **Важно**
>
> Передача вектора как `List<Float>` с преобразованием на стороне YQL через `Knn::ToBinaryStringFloat` даёт худшую производительность, чем кодирование byte array на клиенте.

Native SDK

```
import ydb

def search_items_vector_as_float_list(
    pool: ydb.QuerySessionPool,
    table_name: str,
    embedding: list[float],
    strategy: str = "CosineSimilarity",
    limit: int = 1,
    index_name: str | None = None,
    top_clusters: int = 10,
) -> list[dict]:
    view_index = f"VIEW {index_name}" if index_name else ""

    sort_order = "DESC" if strategy.endswith("Similarity") else "ASC"

    query = f"""
    PRAGMA ydb.KMeansTreeSearchTopSize = "{top_clusters}";
    DECLARE $embedding as List<Float>;

    $target_embedding = Knn::ToBinaryStringFloat($embedding);

    SELECT
        id,
        document,
        Knn::{strategy}(embedding, $target_embedding) as score
    FROM {table_name} {view_index}
    ORDER BY score
```

```
    {sort_order}
    LIMIT {limit};
    """

    result = pool.execute_with_retries(
        query,
        {
            "$embedding": (embedding, ydb.ListType(ydb.PrimitiveType.Float)),
        },
    )

    items = []

    for result_set in result:
        for row in result_set.rows:
            items.append(
                {
                    "id": row["id"],
                    "document": row["document"],
                    "score": row["score"],
                }
            )

    return items
```

Native SDK (Asyncio)

```
import ydb

async def search_items_vector_as_float_list(
    pool: ydb.aio.QuerySessionPool,
    table_name: str,
    embedding: list[float],
    strategy: str = "CosineSimilarity",
    limit: int = 1,
    index_name: str | None = None,
) -> list[dict]:
    view_index = f"VIEW {index_name}" if index_name else ""

    sort_order = "DESC" if strategy.endswith("Similarity") else "ASC"

    query = f"""
    DECLARE $embedding as List<Float>;

    $target_embedding = Knn::ToBinaryStringFloat($embedding);

    SELECT
        id,
        document,
        Knn::{strategy}(embedding, $target_embedding) as score
    FROM {table_name} {view_index}
    ORDER BY score
    {sort_order}
    LIMIT {limit};
    """

    result = await pool.execute_with_retries(
        query,
        {
            "$embedding": (embedding, ydb.ListType(ydb.PrimitiveType.Float)),
        },
    )

    items = []

    for result_set in result:
        for row in result_set.rows:
            items.append(
                {
                    "id": row["id"],
                    "document": row["document"],
                    "score": row["score"],
                }
            )

    return items
```

C#

Примеры векторного поиска для .NET SDK пока недоступны.

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку: ydb-dotnet-sdk#657

JavaScript

Рекомендуемый способ

```
const limit;
const embedding = convertVectorToBytes(new Float32Array([1.5, 2.5, 3.5]))

await sql`SELECT
    id,
    document,
    Knn::CosineSimilarity(embedding, ${embedding}) as score
FROM `table_name`
ORDER BY score DESC
LIMIT ${unsafe(limit)};
```

Альтернативный способ

> **Важно**
>
> Передача вектора как `List<Float>` с преобразованием на стороне YQL через `Knn::ToBinaryStringFloat` даёт худшую производительность, чем кодирование byte array на клиенте.

```
const limit;
const embedding = new Float32Array([1.5, 2.5, 3.5])

await sql`SELECT
        id,
        document,
        Knn::CosineSimilarity(embedding, Knn::ToBinaryStringFloat(${embedding})) as score
    FROM `table_name`
    ORDER BY score DESC
    LIMIT ${unsafe(limit)};
```

Rust

```
let view_index = index_name.map(|n| format!("VIEW {n}")).unwrap_or_default();
let sort_order = if strategy.ends_with("Similarity") { "DESC" } else { "ASC" };

let mut stream = qc
    .query(search_query)
    .param("$embedding", convert_vector_to_bytes(&embedding))
    .await?;
while let Some(result_set) = stream.next_result_set().await? {
    for mut row in result_set {
        // id, document, score
    }
}
stream.close().await?;
```

PHP

Примеры векторного поиска для PHP SDK пока недоступны.

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку: ydb-php-sdk#268

Итоговый пример

Объединим все вышеописанные методы в один пример, который включает следующие шаги:

1. Удаление существующей таблицы
2. Создание новой таблицы
3. Вставка объектов
4. Поиск ближайших векторов без использования индекса
5. Добавление векторного индекса
6. Поиск ближайших векторов с использованием индекса

C++

```
void PrintResults(const std::vector<TResultItem>& items)
{
    if (items.empty()) {
        std::cout << "No items found" << std::endl;
        return;
    }

    for (const auto& item : items) {
        std::cout << "[score=" << item.Score << "] " << item.Id << ": " << item.Document << std::endl;
    }
}

void VectorExample(
    const std::string& endpoint,
```

```
    const std::string& database,
    const std::string& tableName,
    const std::string& indexName)
{
    auto driverConfig = NYdb::CreateFromEnvironment(endpoint + "/?database=" + database);
    NYdb::TDriver driver(driverConfig);
    NYdb::NQuery::TQueryClient client(driver);

    try {
        DropVectorTable(client, tableName);
        CreateVectorTable(client, tableName);
        std::vector<TItem> items = {
            {.Id = "1", .Document = "document 1", .Embedding = {0.98, 0.1, 0.01}},
            {.Id = "2", .Document = "document 2", .Embedding = {1.0, 0.05, 0.05}},
            {.Id = "3", .Document = "document 3", .Embedding = {0.9, 0.1, 0.1}},
            {.Id = "4", .Document = "document 4", .Embedding = {0.03, 0.0, 0.99}},
            {.Id = "5", .Document = "document 5", .Embedding = {0.0, 0.0, 0.99}},
            {.Id = "6", .Document = "document 6", .Embedding = {0.0, 0.02, 1.0}},
            {.Id = "7", .Document = "document 7", .Embedding = {0.0, 1.05, 0.05}},
            {.Id = "8", .Document = "document 8", .Embedding = {0.02, 0.98, 0.1}},
            {.Id = "9", .Document = "document 9", .Embedding = {0.0, 1.0, 0.05}},
        };
        InsertItemsAsBytes(client, tableName, items);
        PrintResults(SearchItemsAsBytes(client, tableName, {1.0, 0.0, 0.0}, "CosineSimilarity", 3));
        AddIndex(driver, client, database, tableName, indexName, "similarity=cosine", 3, 1, 3);
        PrintResults(SearchItemsAsBytes(client, tableName, {1.0, 0.0, 0.0}, "CosineSimilarity", 3, 10, indexN
ame));
    } catch (const std::exception& e) {
        std::cerr << "Execution failed: " << e.what() << std::endl;
    }

    driver.Stop(true);
}
```

Полный код программы доступен по ссылке.

Go

Функциональность векторного поиска полностью поддерживается в Go SDK. Полный пример, объединяющий все вышеописанные операции (создание таблицы, вставка данных, создание индекса, поиск), строится из приведённых выше фрагментов кода. Рабочий пример см. в репозитории ydb-go-sdk.

Java

Пример объединяет шаги из разделов выше: `QueryClient` + `SessionRetryContext` для YQL и `TableClient` + `SessionRetryContext` для `ALTER TABLE` с переименованием индекса.

```
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

import tech.ydb.common.transaction.TxMode;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.TableClient;
import tech.ydb.table.query.Params;
import tech.ydb.table.result.ResultSetReader;
import tech.ydb.table.settings.AlterTableSettings;
import tech.ydb.table.values.ListType;
import tech.ydb.table.values.ListValue;
import tech.ydb.table.values.PrimitiveType;
import tech.ydb.table.values.PrimitiveValue;
import tech.ydb.table.values.StructType;
import tech.ydb.table.values.Value;

public class VectorSearchJavaExample {

    record Item(String id, String document, float[] embedding) {}
    record ResultItem(String id, String document, float score) {}

    public static void main(String[] args) {
        String connectionString = System.getenv().getOrDefault("YDB_CONNECTION_STRING", "grpc://localhost:213
6/local");
        String tableName = "ydb_vector_search";
        String indexName = "ydb_vector_index";

        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString).build();
            QueryClient queryClient = QueryClient.newClient(transport).build();
            TableClient tableClient = TableClient.newClient(transport).build()) {

            SessionRetryContext queryRetry = SessionRetryContext.create(queryClient).build();
            SessionRetryContext tableRetry = SessionRetryContext.create(tableClient).build();

            // 1. Удаление существующей таблицы
            dropVectorTableIfExists(queryRetry, tableName);
            // 2. Создание таблицы
            createVectorTable(queryRetry, tableName);

            List<Item> items = List.of(
```

```
                new Item("1", "vector 1", new float[]{0.98f, 0.1f, 0.01f}),
                new Item("2", "vector 2", new float[]{1.0f, 0.05f, 0.05f}),
                new Item("3", "vector 3", new float[]{0.9f, 0.1f, 0.1f}),
                new Item("4", "vector 4", new float[]{0.03f, 0.0f, 0.99f}),
                new Item("5", "vector 5", new float[]{0.0f, 0.0f, 0.99f}),
                new Item("6", "vector 6", new float[]{0.0f, 0.02f, 1.0f}),
                new Item("7", "vector 7", new float[]{0.0f, 1.05f, 0.05f}),
                new Item("8", "vector 8", new float[]{0.02f, 0.98f, 0.1f}),
                new Item("9", "vector 9", new float[]{0.0f, 1.0f, 0.05f})
        );

        // 3. Вставка документов
        insertItemsAsBytes(queryRetry, tableName, items);
        // 4. Поиск без индекса
        printResults(searchItemsAsBytes(queryRetry, tableName, new float[]{1, 0, 0},
                "CosineSimilarity", 3, Optional.empty(), 10));

        // 5. Добавление векторного индекса
        addVectorIndex(transport, queryRetry, tableRetry, tableName, indexName,
                "similarity=cosine", 3, 1, 3);

        // 6. Поиск с использованием индекса
        printResults(searchItemsAsBytes(queryRetry, tableName, new float[]{1, 0, 0},
                "CosineSimilarity", 3, Optional.of(indexName), 10));
    }
}

static void dropVectorTableIfExists(SessionRetryContext queryRetry, String tableName) {
    String ddl = String.format("DROP TABLE IF EXISTS `%s`", tableName);
    queryRetry.supplyResult(s -> QueryReader.readFrom(
            s.createQuery(ddl, TxMode.NONE, Params.empty())
    )).join().getValue();
    System.out.println("Vector table dropped");
}

static void createVectorTable(SessionRetryContext retryCtx, String tableName) {
    String query = String.format("""
            CREATE TABLE IF NOT EXISTS `%s` (
                id Utf8,
                document Utf8,
                embedding String,
                PRIMARY KEY (id)
            );""", tableName);

    retryCtx.supplyResult(session -> QueryReader.readFrom(
            session.createQuery(query, TxMode.NONE, Params.empty())
    )).join().getValue();

    System.out.println("Vector table created: " + tableName);
}

static byte[] convertVectorToBytes(float[] vector) {
    ByteBuffer bb = ByteBuffer.allocate(vector.length * Float.BYTES + 1).order(ByteOrder.LITTLE_ENDIAN);
    for (float v : vector) {
        bb.putFloat(v);
    }
    bb.put((byte) 0x01);
    return bb.array();
}

static void insertItemsAsBytes(SessionRetryContext retryCtx, String tableName, List<Item> items) {
    String query = String.format("""
            DECLARE $items AS List<Struct<
                id: Utf8,
                document: Utf8,
                embedding: String
            >>;

            UPSERT INTO `%s`
            (
                id,
                document,
                embedding
            )
            SELECT
                id,
                document,
                embedding,
            FROM AS_TABLE($items);""", tableName);

    StructType rowType = StructType.of(
            "id", PrimitiveType.Text,
            "document", PrimitiveType.Text,
            "embedding", PrimitiveType.Bytes
    );

    List<Value<?>> rows = new ArrayList<>(items.size());
    for (Item item : items) {
        rows.add(rowType.newValue(
                "id", PrimitiveValue.newText(item.id()),
                "document", PrimitiveValue.newText(item.document()),
```

```
                "embedding", PrimitiveValue.newBytes(convertVectorToBytes(item.embedding()))
            ));
        }

        ListValue itemsParam = ListType.of(rowType).newValue(rows);
        Params params = Params.of("$items", itemsParam);

        retryCtx.supplyResult(session -> QueryReader.readFrom(
                session.createQuery(query, TxMode.SERIALIZABLE_RW, params)
        )).join().getValue();

        System.out.println(items.size() + " items inserted");
    }

    static void addVectorIndex(
            GrpcTransport transport,
            SessionRetryContext queryRetry,
            SessionRetryContext tableRetry,
            String tableName,
            String indexName,
            String strategy,
            int dimension,
            int levels,
            int clusters) {

        String tempIndexName = indexName + "__temp";
        String query = String.format("""
                ALTER TABLE `%s`
                ADD INDEX %s
                GLOBAL USING vector_kmeans_tree
                ON (embedding)
                WITH (
                    %s,
                    vector_type="Float",
                    vector_dimension=%d,
                    levels=%d,
                    clusters=%d
                );
                """, tableName, tempIndexName, strategy, dimension, levels, clusters);

        queryRetry.supplyResult(session -> QueryReader.readFrom(
                session.createQuery(query, TxMode.NONE, Params.empty())
        )).join().getValue();

        String tablePath = transport.getDatabase() + "/" + tableName;
        AlterTableSettings settings = new AlterTableSettings()
                .addRenameIndex(tempIndexName, indexName, true);

        tableRetry.supplyStatus(session -> session.alterTable(tablePath, settings))
                .join()
                .expectSuccess("alter table rename index");

        System.out.println("Table index `" + indexName + "` for table `" + tableName + "` added");
    }

    static List<ResultItem> searchItemsAsBytes(
            SessionRetryContext retryCtx,
            String tableName,
            float[] embedding,
            String strategy,
            long limit,
            Optional<String> indexName,
            long topClusters) {

        String viewIndex = indexName.map(n -> "VIEW " + n).orElse("");
        String sortOrder = strategy.endsWith("Similarity") ? "DESC" : "ASC";

        String query = String.format("""
                PRAGMA ydb.KMeansTreeSearchTopSize = "%d";
                DECLARE $embedding as String;
                SELECT
                    id,
                    document,
                    Knn::%s(embedding, $embedding) as score
                FROM %s %s
                ORDER BY score %s
                LIMIT %d;
                """, topClusters, strategy, tableName, viewIndex, sortOrder, limit);

        Params params = Params.of("$embedding", PrimitiveValue.newBytes(convertVectorToBytes(embedding)));

        QueryReader reader = retryCtx.supplyResult(session -> QueryReader.readFrom(
                session.createQuery(query, TxMode.SERIALIZABLE_RW, params)
        )).join().getValue();

        List<ResultItem> result = new ArrayList<>();
        ResultSetReader rs = reader.getResultSet(0);
        while (rs.next()) {
            result.add(new ResultItem(
                    rs.getColumn("id").getText(),
                    rs.getColumn("document").getText(),
```

```
                    rs.getColumn("score").getFloat()
            ));
        }
        return result;
    }

    static void printResults(List<ResultItem> items) {
        if (items.isEmpty()) {
            System.out.println("No items found");
            return;
        }
        for (ResultItem item : items) {
            System.out.printf("[score=%f] %s: %s%n", item.score(), item.id(), item.document());
        }
    }
}
```

Вывод совпадает с примером на Python.

Python

Пример использования

Native SDK

```
import os
import ydb

def print_results(items):
    if len(items) == 0:
        print("No items found")
        return

    for item in items:
        print(f"[score={item['score']}] {item['id']}: {item['document']}")

def drop_vector_table_if_exists(pool: ydb.QuerySessionPool, table_name: str) -> None:
    pool.execute_with_retries(f"DROP TABLE IF EXISTS `{table_name}`")

    print("Vector table dropped")

def main(
    ydb_endpoint: str,
    ydb_database: str,
    ydb_credentials: ydb.AbstractCredentials,
    table_name: str,
    index_name: str,
):
    driver = ydb.Driver(
        endpoint=ydb_endpoint,
        database=ydb_database,
        credentials=ydb_credentials,
    )
    driver.wait(5, fail_fast=True)
    pool = ydb.QuerySessionPool(driver)

    drop_vector_table_if_exists(pool, table_name)

    create_vector_table(pool, table_name)

    items = [
        {"id": "1", "document": "vector 1", "embedding": [0.98, 0.1, 0.01]},
        {"id": "2", "document": "vector 2", "embedding": [1.0, 0.05, 0.05]},
        {"id": "3", "document": "vector 3", "embedding": [0.9, 0.1, 0.1]},
        {"id": "4", "document": "vector 4", "embedding": [0.03, 0.0, 0.99]},
        {"id": "5", "document": "vector 5", "embedding": [0.0, 0.0, 0.99]},
        {"id": "6", "document": "vector 6", "embedding": [0.0, 0.02, 1.0]},
        {"id": "7", "document": "vector 7", "embedding": [0.0, 1.05, 0.05]},
        {"id": "8", "document": "vector 8", "embedding": [0.02, 0.98, 0.1]},
        {"id": "9", "document": "vector 9", "embedding": [0.0, 1.0, 0.05]},
    ]

    insert_items_vector_as_bytes(pool, table_name, items)

    items = search_items_vector_as_bytes(
        pool,
        table_name,
        embedding=[1, 0, 0],
        strategy="CosineSimilarity",
        limit=3,
        top_clusters=10,
    )
    print_results(items)

    add_vector_index(
        pool,
        driver,
        table_name,
        index_name=index_name,
        strategy="similarity=cosine",
        dimension=3,
        levels=1,
```

```
        clusters=3,
    )

    items = search_items_vector_as_bytes(
        pool,
        table_name,
        embedding=[1, 0, 0],
        index_name=index_name,
        strategy="CosineSimilarity",
        limit=3,
        top_clusters=10,
    )
    print_results(items)

    pool.stop()
    driver.stop()

if __name__ == "__main__":
    main(
        ydb_endpoint=os.environ.get("YDB_ENDPOINT", "grpc://localhost:2136"),
        ydb_database=os.environ.get("YDB_DATABASE", "/local"),
        ydb_credentials=ydb.credentials_from_env_variables(),
        table_name="ydb_vector_search",
        index_name="ydb_vector_index",
    )
```

Native SDK (Asyncio)

```
import os
import ydb
import asyncio

def print_results(items):
    if len(items) == 0:
        print("No items found")
        return

    for item in items:
        print(f"[score={item['score']}] {item['id']}: {item['document']}")

async def drop_vector_table_if_exists(pool: ydb.aio.QuerySessionPool, table_name: str) -> None:
    await pool.execute_with_retries(f"DROP TABLE IF EXISTS `{table_name}`")

    print("Vector table dropped")

async def main(
    ydb_endpoint: str,
    ydb_database: str,
    ydb_credentials: ydb.AbstractCredentials,
    table_name: str,
    index_name: str,
):
    async with ydb.aio.Driver(
        endpoint=ydb_endpoint,
        database=ydb_database,
        credentials=ydb_credentials,
    ) as driver:
        await driver.wait(5, fail_fast=True)
        pool = ydb.aio.QuerySessionPool(driver)

        await drop_vector_table_if_exists(pool, table_name)

        await create_vector_table(pool, table_name)

        items = [
            {"id": "1", "document": "vector 1", "embedding": [0.98, 0.1, 0.01]},
            {"id": "2", "document": "vector 2", "embedding": [1.0, 0.05, 0.05]},
            {"id": "3", "document": "vector 3", "embedding": [0.9, 0.1, 0.1]},
            {"id": "4", "document": "vector 4", "embedding": [0.03, 0.0, 0.99]},
            {"id": "5", "document": "vector 5", "embedding": [0.0, 0.0, 0.99]},
            {"id": "6", "document": "vector 6", "embedding": [0.0, 0.02, 1.0]},
            {"id": "7", "document": "vector 7", "embedding": [0.0, 1.05, 0.05]},
            {"id": "8", "document": "vector 8", "embedding": [0.02, 0.98, 0.1]},
            {"id": "9", "document": "vector 9", "embedding": [0.0, 1.0, 0.05]},
        ]

        await insert_items_vector_as_bytes(pool, table_name, items)

        items = await search_items_vector_as_bytes(
            pool,
            table_name,
            embedding=[1, 0, 0],
            strategy="CosineSimilarity",
            limit=3,
        )
        print_results(items)

        await add_vector_index(
            pool,
            driver,
            table_name,
```

```
            index_name=index_name,
            strategy="similarity=cosine",
            dimension=3,
            levels=1,
            clusters=3,
        )

        items = await search_items_vector_as_bytes(
            pool,
            table_name,
            embedding=[1, 0, 0],
            index_name=index_name,
            strategy="CosineSimilarity",
            limit=3,
        )
        print_results(items)

        await pool.stop()

if __name__ == "__main__":
    asyncio.run(main(
        ydb_endpoint=os.environ.get("YDB_ENDPOINT", "grpc://localhost:2136"),
        ydb_database=os.environ.get("YDB_DATABASE", "/local"),
        ydb_credentials=ydb.credentials_from_env_variables(),
        table_name="ydb_vector_search",
        index_name="ydb_vector_index",
    ))
```

Вывод программы:

```
Vector table dropped
Vector table created
9 items inserted
[score=0.997509241104126] 2: vector 2
[score=0.9947828650474548] 1: vector 1
[score=0.9878783822059631] 3: vector 3
Table index ydb_vector_index created.
[score=0.997509241104126] 2: vector 2
[score=0.9947828650474548] 1: vector 1
[score=0.9878783822059631] 3: vector 3
```

В результате видно, что таблица была создана, добавлено 9 документов и успешно выполнен поиск по близости векторов — как до, так и после добавления векторного индекса.

Полный код программы доступен по ссылке.

C#

Примеры векторного поиска для .NET SDK пока недоступны.

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку: ydb-dotnet-sdk#657

JavaScript

Пример добавления векторного индекса и полная программа для JavaScript SDK пока недоступны.

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку: ydb-js-sdk#627

Rust

Полный пример на GitHub: vector-search.rs.

```
drop_vector_table_if_exists(&mut qc, table_name).await?;
create_vector_table(&mut qc, table_name).await?;
insert_items_as_bytes(&mut qc, table_name, &items).await?;
let hits = search_items_as_bytes(&mut qc, table_name, &[1.0, 0.0, 0.0],
    "CosineSimilarity", 3, None).await?;
add_vector_index(&mut qc, table_name, index_name, "similarity=cosine", 3, 1, 3).await?;
let hits = search_items_as_bytes(&mut qc, table_name, &[1.0, 0.0, 0.0],
    "CosineSimilarity", 3, Some(index_name)).await?;
```

PHP

Примеры векторного поиска для PHP SDK пока недоступны.

Функциональность на данный момент не поддерживается.

Отслеживать прогресс или проголосовать за поддержку: ydb-php-sdk#268

Распределённая блокировка

Рассмотрим сценарий, где необходимо обеспечить, чтобы с разделяемым ресурсом в один момент времени работал только один экземпляр клиентского приложения. Для его реализации можно воспользоваться механизмом семафоров в узлах координации YDB.

Механизм аренды семафоров

В отличие от локального многопоточного программирования, клиенты в распределённых системах не захватывают блокировки или семафоры напрямую. Вместо этого они арендуют их на определённое время, которое можно периодически продлевать. Поскольку физическое время может различаться на разных машинах, клиенты и сервер могут оказаться в ситуации, когда несколько клиентов считают, что их сессии захватили один и тот же семафор одновременно, даже если с точки зрения сервера это не так. Чтобы уменьшить вероятность таких ситуаций, важно заранее настроить автоматическую синхронизацию времени как на серверах с клиентскими приложениями, так и на стороне YDB, желательно с использованием единого источника времени.

Таким образом, реализация распределённой блокировки через такие механизмы не может гарантировать отсутствие одновременного доступа к ресурсу, но может значительно снизить вероятность такого события. Это может быть использовано как оптимизация, чтобы клиенты не конкурировали за общий ресурс, когда это не имеет смысла. Гарантии отсутствия конкурентных запросов к ресурсу могут быть реализованы на стороне самого ресурса.

Пример кода

C++

```
#include <ydb-cpp-sdk/client/coordination/coordination.h>

void CoordinationExclusiveWork(
    const NYdb::TDriver& driver,
    const std::string& nodePath,
    const std::string& semaphoreName)
{
    using namespace NYdb::NStatusHelpers;

    NYdb::NCoordination::TClient client(driver);

    auto sessionResult = client.StartSession(nodePath).ExtractValueSync();
    ThrowOnError(sessionResult);

    auto session = sessionResult.ExtractResult();

    auto acquireSettings = NYdb::NCoordination::TAcquireSemaphoreSettings()
        .Ephemeral(true)
        .Exclusive()
        .Timeout(TDuration::Minutes(5));

    auto acquireResult = session.AcquireSemaphore(semaphoreName, acquireSettings).ExtractValueSync();
    ThrowOnError(acquireResult);

    if (!acquireResult.GetResult()) {
        // semaphore was not acquired
        return;
    }

    // lock acquired, start processing

    auto releaseStatus = session.ReleaseSemaphore(semaphoreName).ExtractValueSync();
    ThrowOnError(releaseStatus);

    if (!releaseStatus.GetResult()) {
        // semaphore was not released
        return;
    }

    // lock released, end processing
}
```

Go

```
for {
  if session, err := db.Coordination().CreateSession(ctx, path); err != nil {
    return fmt.Errorf("cannot create session: %v", err);
  }

  if lease, err := session.AcquireSemaphore(ctx,
    semaphore,
    coordination.Exclusive,
    options.WithEphemeral(true),
  ); err != nil {
    // the session is likely lost, try to create a new one and get the lock in it
    session.Close(ctx);
    continue;
  }

  // lock acquired, start processing
  select {
     case <-lease.Context().Done():
  }
```

```
  // lock released, end processing
}
```

Python

Native SDK

```python
import ydb

def coordination_service_workflow(driver: ydb.Driver, node_path: str, semaphore_name: str):
    client = driver.coordination_client

    client.create_node(node_path)

    with client.session(node_path) as session:
        with session.semaphore(semaphore_name) as semaphore:
            print("Some exclusive work")
```

Native SDK (Asyncio)

```python
import os
import ydb

async def coordination_service_workflow(driver: ydb.aio.Driver, node_path: str, semaphore_name: str):
    client = driver.coordination_client
    await client.create_node(node_path)
    async with client.session(node_path) as session:
        async with session.semaphore(semaphore_name) as semaphore:
            print("Some exclusive work")
```

C#

Функциональность на данный момент не поддерживается.

JavaScript

```javascript
import { Driver } from '@ydbjs/core'
import { CoordinationClient } from '@ydbjs/coordination'

const driver = new Driver('grpc://localhost:2136/local')
const client = new CoordinationClient(driver)

await using session = await client.createSession('/local/my-app')
await using lock = await session.mutex('job-lock').lock()
await doWork(lock.signal)

// For long lived applications

for await (let session of client.openSession('/local/my-app')) {
  let mutex = session.mutex('job-lock')

  try {
    // Blocks until the lock is acquired.
    await using lock = await mutex.lock()

    await doWork(lock.signal)
  } catch {
    if (session.signal.aborted) continue // session expired, retry
    throw error
  }

  break
}
```

Java

```java
import java.time.Duration;

import tech.ydb.common.transaction.TxMode;
import tech.ydb.coordination.CoordinationClient;
import tech.ydb.coordination.CoordinationSession;
import tech.ydb.coordination.SemaphoreLease;
import tech.ydb.core.grpc.GrpcTransport;
import tech.ydb.query.QueryClient;
import tech.ydb.query.result.ResultSetReader;
import tech.ydb.query.tools.QueryReader;
import tech.ydb.query.tools.SessionRetryContext;
import tech.ydb.table.query.Params;

public class DistributedLockExample {

    private static final String NODE_PATH_SUFFIX = "/my-app-lock";
    private static final String SEMAPHORE_NAME = "job-lock";

    public static void main(String[] args) {
        String connectionString = System.getenv().getOrDefault(
                "YDB_CONNECTION_STRING", "grpc://localhost:2136/local");
```

```
        try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString).build();
             CoordinationClient coordinationClient = CoordinationClient.newClient(transport).build();
             QueryClient queryClient = QueryClient.newClient(transport).build()) {

            String nodePath = transport.getDatabase() + NODE_PATH_SUFFIX;
            coordinationClient.createNode(nodePath).join().expectSuccess("create node failed");

            try (CoordinationSession session = coordinationClient.createSession(nodePath)) {
                session.connect().join().expectSuccess("connect failed");

                SemaphoreLease lease = session.acquireEphemeralSemaphore(
                        SEMAPHORE_NAME, true, Duration.ofMinutes(5)
                ).join().getValue();

                try {
                    // монопольная работа с ресурсом
                    SessionRetryContext retryCtx = SessionRetryContext.create(queryClient).build();
                    QueryReader reader = retryCtx.supplyResult(s -> QueryReader.readFrom(
                            s.createQuery("SELECT 1 AS value", TxMode.NONE, Params.empty())
                    )).join().getValue();

                    ResultSetReader rs = reader.getResultSet(0);
                    if (rs.next()) {
                        System.out.println("Lock acquired, SELECT 1 = " + rs.getColumn("value").getInt32());
                    }
                } finally {
                    lease.release().join();
                }
            }
        }
    }
}
```

Rust

```
use ydb::{ClientBuilder, NodeConfigBuilder, SessionOptionsBuilder, YdbResult};

#[tokio::main]
async fn main() -> YdbResult<()> {
    let client = ClientBuilder::new_from_connection_string("grpc://localhost:2136?database=local")?
        .client()?;
    client.wait().await?;

    let mut coordination_client = client.coordination_client();
    coordination_client
        .create_node(
            "/local/my_lock_node".into(),
            NodeConfigBuilder::default().build()?,
        )
        .await?;

    let session = coordination_client
        .create_session(
            "/local/my_lock_node".into(),
            SessionOptionsBuilder::default().build()?,
        )
        .await?;

    session.create_semaphore("resource", 1, vec![]).await?;
    let _lease = session.acquire_semaphore("resource".into(), 1).await?;
    // критическая секция
    Ok(())
}
```

PHP

Функциональность на данный момент не поддерживается.

Выбор лидера (leader election)

Рассмотрим сценарий, где несколько экземпляров приложения хотят выбрать лидера и всегда знать, кто им является.

Этот сценарий можно реализовать с помощью семафоров в узлах координации YDB следующим образом:

1. Создаётся семафор (например, с именем `my-service-leader`) с `Limit=1`.
2. Все экземпляры приложения выполняют `AcquireSemaphore` с `Count=1`, указывая в `Data` свой endpoint.
3. Только у одного экземпляра приложения вызов завершится быстро, остальные встанут в очередь. Тот, у кого вызов завершился успешно, становится текущим лидером.
4. Все экземпляры приложения выполняют `DescribeSemaphore` с `WatchOwners=true` и `IncludeOwners=true`. В результате вызова в `Owners` будет максимум один элемент, из `Data` узнаётся endpoint текущего лидера.
5. При смене лидера вызывается `OnChanged`. В этом случае экземпляры приложения выполняют аналогичный вызов `DescribeSemaphore` и узнают endpoint нового лидера.

Обнаружение сервисов (service discovery)

Рассмотрим сценарий, где экземпляры приложения динамически поднимаются и публикуют свой endpoint, а другие клиенты хотят получать этот список и реагировать на его изменения.

Этот сценарий можно реализовать с помощью семафоров в узлах координации YDB следующим образом:

1. Создаётся семафор (например, с именем `my-service-endpoints`) с `Limit=Max<ui64>()`.
2. Все экземпляры приложения выполняют `AcquireSemaphore` с `Count=1`, указывая в `Data` свой endpoint.
3. Поскольку лимит семафора очень большой, все вызовы `AcquireSemaphore` должны завершиться быстро.
4. На этом этапе публикация завершена, и экземплярам приложения нужно только реагировать на остановку сессии, публикуя себя заново через новую сессию.
5. Клиенты выполняют `DescribeSemaphore` с `IncludeOwners=true` и, при необходимости, с `WatchOwners=true`. В результате вызова в поле `Owners` в `Data` будут содержаться endpoint'ы зарегистрированных экземпляров приложения.
6. При изменении списка endpoint'ов вызывается `OnChanged`. В этом случае клиенты выполняют аналогичный вызов `DescribeSemaphore` и получают обновлённый список.

Публикация конфигурации

Рассмотрим сценарий, где необходимо публиковать небольшую конфигурацию для экземпляров приложения, которые должны оперативно реагировать на её изменения.

Этот сценарий можно реализовать с помощью семафоров в узлах координации YDB следующим образом:

1. Создаётся семафор (например, с именем `my-service-config`).
2. Через `UpdateSemaphore` публикуется обновлённая конфигурация.
3. Экземпляры приложения делают `DescribeSemaphore` с `WatchData=true`, в результате вызова в `Data` будет текущая версия конфигурации.
4. В случае изменения конфигурации вызывается `OnChanged`. В этом случае экземпляры приложения делают аналогичный вызов `DescribeSemaphore` и получают обновлённую конфигурацию.

YDB CLI рецепты

Этот раздел содержит рецепты для различных задач, которые можно решить с помощью YDB CLI.

Содержание:

- Конвертация таблиц между строковой и колоночной ориентацией
- Проведение нагрузочного тестирования
- Настройка времени жизни строк (TTL) таблицы

Смотрите также:

- YDB CLI
- YDB для разработчиков приложений

Конвертация таблиц между строковой и колоночной ориентацией

YDB поддерживает два основных типа таблиц: строковые и колоночные. Выбранный тип таблицы определяет физическое представление данных на дисках, поэтому изменение типа существующей таблицы невозможно. Однако вы можете создать новую таблицу другого типа и скопировать в неё данные. Этот рецепт состоит из следующих шагов:

1. Подготовка новой таблицы
2. Копирование данных
3. Переключение нагрузки *(опционально)*

Эти инструкции предполагают, что исходная таблица является строковой, и целью является получение аналогичной колоночной таблицы; однако роли таблиц могут быть изменены на противоположные.

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Подготовка новой таблицы

Возьмите копию исходной команды `CREATE TABLE`, использованной для создания исходной таблицы. Проделайте в этом запросе следующие изменения, чтобы создать файл с запросом `CREATE TABLE` для таблицы назначения:

1. Измените имя таблицы на желаемое новое имя.
2. Установите значение параметра `STORE` в `COLUMN`, чтобы сделать таблицу колоночной.

Запустите этот запрос (предполагается, что он сохранён в файле с именем `create_column_oriented_table.yql`):

```
$ ydb -p quickstart yql -f create_column_oriented_table.yql
```

Примеры схем тестовых таблиц и заполнения их данными

Исходная строковая таблица:

```
CREATE TABLE `row_oriented_table` (
    id Int64 NOT NULL,
    metric_a Double,
    metric_b Double,
    metric_c Double,
    PRIMARY KEY (id)
)
```

Колоночная таблица назначения:

```
CREATE TABLE `column_oriented_table` (
    id Int64 NOT NULL,
    metric_a Double,
    metric_b Double,
    metric_c Double,
    PRIMARY KEY (id)
)
PARTITION BY HASH(id)
WITH (STORE = COLUMN)
```

> **Примечание**
>
> Обратитесь к документации для разработчиков приложений, чтобы узнать больше о партицировании колоночных таблиц и выборе ключа их партицирования (параметр `PARTITION BY`).

Заполнение исходной строковой таблицы случайными данными:

```
INSERT INTO `row_oriented_table` (id, metric_a, metric_b, metric_c)
SELECT
    id,
    Random(id + 1),
    Random(id + 2),
    Random(id + 3)
FROM (
    SELECT ListFromRange(1, 1000) AS id
) FLATTEN LIST BY id
```

Копирование данных

На данный момент рекомендуемый способ копирования данных между YDB таблицами разных типов — это экспорт и импорт:

1. Экспорт данных в локальную файловую систему:

```
$ ydb -p quickstart dump -p row_oriented_table -o tmp_backup/
```

2. Импорт этих данных в другую таблицу YDB:

```
$ ydb -p quickstart import file csv -p column_oriented_table tmp_backup/row_oriented_table/*.csv
```

Убедитесь, что у вас достаточно свободного места в файловой системе для хранения всех данных.

Переключение нагрузки

В настоящее время невозможно бесшовно заменить оригинальную таблицу на колоночную. Однако, при необходимости, вы можете постепенно переключить ваши запросы на работу с новой таблицей, заменив путь к оригинальной таблице в запросах на новый.

Если исходная таблица больше не нужна, её можно удалить с помощью `ydb -p quickstart table drop row_oriented_table` или `yql -p quickstart yql -s "DROP TABLE row_oriented_table"`.

Смотрите также

- YDB CLI
- YDB для разработчиков приложений

Проведение нагрузочного тестирования

В YDB интегрирован инструментарий для проведения нагрузочных тестов с использованием стандартных бенчмарков:

Бенчмарк	Справка
TPC-C	tpcc
TPC-H	tpch
TPC-DS	tpcds
ClickBench	clickbench

Помимо стандартных бенчмарков есть еще несколько внутренних:

Бенчмарк	Справка
`Key Value`	kv
`Stock`	stock
`Topic`	topic
`Transfer`	topic

Также предусмотрена возможность запуска пользовательских сценариев тестирования, которые инициируются посредством команды `ydb workload query`, см. описание. Подробности приведены в соответствующем разделе.

Все указанные методы эмулируют пользовательскую нагрузку на базу данных в рамках заданных сценариев. Детальное описание каждого метода представлено в соответствующих разделах, ссылки на которые приведены выше.

Все команды для работы с бенчмарками сгруппированы в соответствующие категории:

```
ydb workload tpcc --path path/in/database ...
ydb workload clickbench --path path/in/database ...
ydb workload tpch --path path/in/database ...
ydb workload tpcds --path path/in/database ...
ydb workload query --path path/in/database ...
ydb workload kv --path path/in/database ...
ydb workload stock --path path/in/database ...
ydb workload topic ...
ydb workload transfer ...
```

Нагрузочное тестирование состоит из трёх этапов:

1. Подготовка данных
2. Тестирование
3. Очистка

Подготовка данных

Состоит из двух этапов, это инициализация таблиц и наполнение их данными.

Инициализация

Инициализация производится командой `init`:

```
ydb workload tpcc --path tpcc/10wh init
ydb workload clickbench --path clickbench/hits init --store=row
ydb workload tpch --path tpch/s1 init --store=column
ydb workload tpcds --path tpcds/s1 init --store=external-s3
ydb workload query --path user/suite1 init --suite-path /home/user/user_suite
ydb workload kv --path kv init --store=column
ydb workload stock --path stock init --store=row
ydb workload topic init --topic some_topic
ydb workload transfer topic-to-table init --topic some_topic --table /db/table
```

На этапе создания таблиц, если вы запускаете `tpch`, `tpcds` или `clickbench`, возможно дополнительно настроить создаваемые таблицы:

- Выбрать тип используемых таблиц: строковые, колоночные, внешние и тд. (параметр `--store`);
- Выбрать типы используемых колонок: строк (параметр `--string`), дат и времени (`--datetime`) и тип вещественных чисел (`--float-mode`).

Также можно указать, что перед созданием таблицы должны быть удалены, если они уже созданы (параметр `--clear`).

Подробнее см. описание команд для каждого бенчмарка:

- tpcc init
- clickbench init
- tpch init
- tpcds init
- query init
- kv init
- stock init
- topic init

- transfer init

Наполнение данными

Наполнение созданных таблиц данными выполняется с помощью команды `import`. Эта команда специфична для каждого бенчмарка, и её поведение зависит от дополнительных опций.

Подробное описание см. в соответствующих разделах:

- tpcc import
- clickbench import
- tpch import
- tpcds import
- query import

Примеры:

```
ydb workload tpcc --path tpcc/10wh import
ydb workload clickbench --path clickbench/hits import files --input hits.csv.gz
ydb workload tpch --path tpch/s1 import generator --scale 1
ydb workload tpcds --path tpcds/s1 import generator --scale 1
ydb workload query --path user/suite1 import --suite-path /home/user/user_suite
```

Тестирование

Непосредственный запуск нагрузочного тестирования выполняется командой `run`. Её поведение практически одинаково для разных бенчмарков, хотя некоторые различия всё-таки присутствуют.

Примеры:

```
ydb workload tpcc --path tpcc/10wh run
ydb workload clickbench --path clickbench/hits run --include 1-5,8
ydb workload tpch --path tpch/s1 run --exclude 3,4 --iterations 3
ydb workload tpcds --path tpcds/s1 run --plan ~/query_plan --include 2 --iterations 5
ydb workload query --path user/suite1 run --plan ~/query_plan --include first_query_set.1.sql,second_query_se
t.2.sql --iterations 5
ydb workload kv --path kv run mixed
ydb workload stock --path stock run add-rand-order
ydb workload topic run full --topic some_topic
ydb workload transfer topic-to-table run --topic some_topic --table /db/table
```

Команда `run` для каждого из бенчмарков имеет ряд дополнительных параметров для настройки видов генерируемых отчётов, сбора статистики и прочих результатов нагрузочного тестирования.

Подробное описание см. в соответствующих разделах:

- tpcc run
- clickbench run
- tpch run
- tpcds run
- query run
- kv run
- stock run
- topic run
- transfer run

Очистка

При завершении работ по нагрузочному тестированию тестовые данные и таблицы можно удалить командой `clean`:

```
ydb workload tpcc --path tpcc/10wh clean
ydb workload clickbench --path clickbench/hits clean
ydb workload tpch --path tpch/s1 clean
ydb workload tpcds --path tpcds/s1 clean
ydb workload query --path user/suite1 clean
ydb workload kv --path kv clean
ydb workload stock --path stock clean
ydb workload topic clean --topic some_topic
ydb workload transfer topic-to-table clean --topic some_topic --table /db/table
```

Подробное описание см. в соответствующих разделах:

- tpcc clean
- clickbench clean
- tpch clean
- tpcds clean
- query clean
- kv clean
- stock clean
- topic clean
- transfer clean

Настройка времени жизни строк (TTL) таблицы

В этом разделе приведены примеры настройки TTL строковых и колоночных таблиц при помощи YDB CLI.

Включение TTL для существующих строковых и колоночных таблиц

В приведенном ниже примере строки таблицы `mytable` будут удаляться спустя час после наступления времени, записанного в колонке `created_at`:

```
$ ydb -e <endpoint> -d <database> table ttl set --column created_at --expire-after 3600 mytable
```

Следующий пример демонстрирует использование колонки `modified_at` с числовым типом (`Uint32`) в качестве TTL-колонки. Значение колонки интерпретируется как секунды от Unix-эпохи:

```
$ ydb -e <endpoint> -d <database> table ttl set --column modified_at --expire-after 3600 --unit seconds mytab
le
```

Включение вытеснения данных во внешнее S3-совместимое хранилище

Важно

Поддерживается только для колоночных таблиц. Поддержка функциональности для строковых таблиц находится в разработке.

Для включения вытеснения требуется объект external data source, описывающий подключение к внешнему хранилищу. Пример создания объекта можно найти в рецептах YQL.

Следующий пример демонстрирует включение вытеснения данных через вызов YQL-запроса из YDB CLI. Строки таблицы `mytable` будут переноситься в бакет, описанный во внешнем источнике данных `/Root/s3_cold_data`, спустя час после наступления времени, записанного в колонке `created_at`, а спустя 24 часа будут удаляться.

```
$ ydb -e <endpoint> -d <database> sql -s '
    ALTER TABLE `mytable` SET (
        TTL =
            Interval("PT1H") TO EXTERNAL DATA SOURCE `/Root/s3_cold_data`,
            Interval("PT24H") DELETE
        ON modified_at AS SECONDS
    );
'
```

Выключение TTL

```
$ ydb -e <endpoint> -d <database> table ttl reset mytable
```

Получение настроек TTL

Текущие настройки TTL можно получить из описания таблицы:

```
$ ydb -e <endpoint> -d <database> scheme describe mytable
```

YQL рецепты

Этот раздел содержит рецепты запросов для различных задач, которые можно решить с помощью YQL, SQL-диалекта YDB.

Содержание:

- Доступ к значениям в JSON с помощью YQL
- Изменение JSON с помощью YQL
- Настройка времени жизни строк (TTL) таблицы

Смотрите также:

- YQL - Обзор
- YDB для разработчиков приложений

Доступ к значениям в JSON с помощью YQL

YQL предоставляет два основных способа извлечения значений из JSON:

- Использование **JSON-функций из SQL стандарта**. Этот подход рекомендуется для простых случаев и для команд, которые знакомы с ними по другим СУБД.
- Использование **Yson UDF**, встроенных функций для работы со списками и словарями, а также лямбд. Этот подход более гибкий и тесно интегрирован с системой типов данных YDB, поэтому рекомендуется для сложных случаев.

Ниже приведены рецепты, которые используют один и тот же входной JSON, чтобы показать, как использовать каждый из этих вариантов для проверки существования ключа, получения конкретного значения и извлечения поддерева.

JSON-функции

```
$json = @@{
    "friends": [
        {
            "name": "James Holden",
            "age": 35
        },
        {
            "name": "Naomi Nagata",
            "age": 30
        }
    ]
}@@j;

SELECT
    JSON_EXISTS($json, "$.friends[*].name"), -- True
    CAST(JSON_VALUE($json, "$.friends[0].age") AS Int32), -- 35
    JSON_QUERY($json, "$.friends[0]"); -- {"name": "James Holden", "age": 35}
```

Функции `JSON_*` ожидают на вход данные типа `Json`. В этом примере строковый литерал имеет суффикс `j`, обозначающий его как `Json`. В таблицах данные могут храниться либо в формате JSON, либо как строковое представление. Для преобразования данных из `String` в тип данных `JSON` используйте функцию `CAST`, например `CAST(my_string AS JSON)`.

Yson UDF

Этот подход обычно сочетает в себе несколько функций и выражений, поэтому запрос может использовать различные конкретные стратегии.

Преобразование всего JSON в YQL контейнеры

```
$json = @@{
    "friends": [
        {
            "name": "James Holden",
            "age": 35
        },
        {
            "name": "Naomi Nagata",
            "age": 30
        }
    ]
}@@j;

$containers = Yson::ConvertTo($json, Struct<friends:List<Struct<name:String?,age:Int32?>>>);
$has_name = ListAny(
    ListMap($containers.friends, ($friend) -> {
        return $friend.name IS NOT NULL;
    })
);
$get_age = $containers.friends[0].age;
$get_first_friend = Yson::SerializeJson(Yson::From($containers.friends[0]));

SELECT
    $has_name, -- True
    $get_age, -- 35
    $get_first_friend; -- {"name": "James Holden", "age": 35}
```

Не обязательно преобразовывать весь JSON объект в структурированное сочетание контейнеров. Некоторые поля могут быть опущены, если они не используются, в то время как некоторые поддеревья могут быть оставлены в неструктурированном типе данных, таком как `Json`.

Работа с представлением в памяти

```
$json = @@{
    "friends": [
        {
            "name": "James Holden",
            "age": 35
        },
        {
            "name": "Naomi Nagata",
            "age": 30
        }
    ]
}@@j;
```

```
$has_name = ListAny(
    ListMap(Yson::ConvertToList($json.friends), ($friend) -> {
        return Yson::Contains($friend, "name");
    })
);
$get_age = Yson::ConvertToInt64($json.friends[0].age);
$get_first_friend = Yson::SerializeJson($json.friends[0]);

SELECT
    $has_name, -- True
    $get_age, -- 35
    $get_first_friend; -- {"name": "James Holden", "age": 35}
```

Смотрите также

- Изменение JSON с помощью YQL

Изменение JSON с помощью YQL

В памяти YQL работает с неизменяемыми значениями. Таким образом, когда запросу нужно изменить что-то внутри значения JSON, следует думать об этом как о создании нового значения из частей старого.

Данный пример запроса принимает входной JSON названный `$fields`, парсит его, заменяет ключ `a` на 0, удаляет ключ `d` и добавляет ключ `c` со значением 3:

```
$fields = '{"a": 1, "b": 2, "d": 4}'j;
$pairs = DictItems(Yson::ConvertToInt64Dict($fields));
$result_pairs = ListExtend(ListNotNull(ListMap($pairs, ($item) -> {
    $item = if ($item.0 == "a", ("a", 0), $item);
    return if ($item.0 == "d", null, $item);
})), [("c", 3)]);
$result_dict = ToDict($result_pairs);
SELECT Yson::SerializeJson(Yson::From($result_dict));
```

Смотрите также

- Yson
- Функции для работы со списками
- Функции для работы со словарями
- Доступ к значениям в JSON с помощью YQL

Настройка времени жизни строк (TTL) таблицы

В этом разделе приведены примеры настройки TTL строковых и колоночных таблиц при помощи YQL.

Включение TTL для существующих строковых и колоночных таблиц

В приведенном ниже примере строки таблицы `mytable` будут удаляться спустя час после наступления времени, записанного в колонке `created_at`:

```
ALTER TABLE `mytable` SET (TTL = Interval("PT1H") ON created_at);
```

> **Совет**
>
> `Interval` создается из строкового литерала в формате ISO 8601 с некоторыми ограничениями.

Следующий пример демонстрирует использование колонки `modified_at` с числовым типом (`Uint32`) в качестве TTL-колонки. Значение колонки интерпретируется как секунды от Unix-эпохи:

```
ALTER TABLE `mytable` SET (TTL = Interval("PT1H") ON modified_at AS SECONDS);
```

Включение вытеснения данных во внешнее S3-совместимое хранилище

> **Важно**
>
> Поддерживается только для колоночных таблиц. Поддержка функциональности для строковых таблиц находится в разработке.

Для включения вытеснения требуется объект external data source, описывающий подключение к внешнему хранилищу. В приведённом ниже примере создаётся external data source `/Root/s3_cold_data`: он описывает подключение к бакету `test_cold_data`, расположенному в Yandex Object Storage, с авторизацией через статический ключ доступа, данные которого хранятся в секретах `access_key` и `secret_key`.

```
CREATE SECRET access_key WITH (value="...");
CREATE SECRET secret_key WITH (value="...");

CREATE EXTERNAL DATA SOURCE `/Root/s3_cold_data` WITH (
    SOURCE_TYPE="ObjectStorage",
    AUTH_METHOD="AWS",
    LOCATION="http://storage.yandexcloud.net/test_cold_data",
    AWS_ACCESS_KEY_ID_SECRET_PATH="access_key",
    AWS_SECRET_ACCESS_KEY_SECRET_PATH="secret_key",
    AWS_REGION="ru-central1"
)
```

Используя объект external data source, можно включить вытеснение данных следующим образом.

В следующем примере строки таблицы `mytable` будут переноситься в бакет, описанный во внешнем источнике данных `/Root/s3_cold_data`, спустя час после наступления времени, записанного в колонке `created_at`, а спустя 24 часа будут удаляться:

```
ALTER TABLE `mytable` SET (
    TTL =
        Interval("PT1H") TO EXTERNAL DATA SOURCE `/Root/s3_cold_data`,
        Interval("PT24H") DELETE
    ON modified_at AS SECONDS
);
```

В следующем примере строки таблицы `mytable` будут переноситься в бакет `/Root/s3_cold` через час и в бакет `/Root/s3_frozen` через 30 дней после наступления времени, записанного в колонке `created_at`:

```
ALTER TABLE `mytable` SET (
    TTL =
        Interval("PT1H") TO EXTERNAL DATA SOURCE `/Root/s3_cold`,
        Interval("PT30D") TO EXTERNAL DATA SOURCE `/Root/s3_frozen`
    ON modified_at AS SECONDS
);
```

Включение TTL для вновь создаваемой таблицы

Для вновь создаваемой таблицы можно передать настройки TTL вместе с ее описанием:

```
CREATE TABLE `mytable` (
  id Uint64,
  expire_at Timestamp,
  PRIMARY KEY (id)
) WITH (
  TTL = Interval("PT1H") ON expire_at
);
```

Выключение TTL

```
ALTER TABLE `mytable` RESET (TTL);
```

Рецепты работы с векторными индексами

Этот раздел содержит рецепты работы с векторными индексами.

Содержание:

- Векторный индекс — быстрый старт
- Векторный индекс с загрузкой внешнего набора данных

Векторный индекс — быстрый старт

Эта статья поможет быстро начать работу с векторными индексами в YDB на простейшем модельном примере.

В статье будут рассмотрены следующие шаги работы с векторным индексом:

- создание таблицы с векторами;
- заполнение таблицы данными;
- построение векторного индекса;
- выполнение векторного поиска без индекса;
- выполнение векторного поиска с индексом.

Шаг 1. Создание таблицы с векторами

Сначала нужно создать таблицу в YDB, в которой будут храниться векторы. Это можно сделать с помощью SQL-запроса:

```
CREATE TABLE Vectors (
  id Uint64,
  embedding String,
  PRIMARY KEY (id)
);
```

Эта таблица `Vectors` имеет два столбца:

- `id` — уникальный идентификатор каждого вектора;
- `embedding` — вектор вещественных чисел, упакованный в строку.

Шаг 2. Заполнение таблицы данными

После создания таблицы следует добавить в нее векторы запросом `UPSERT INTO`:

```
UPSERT INTO Vectors(id, embedding)
VALUES
    (1, Untag(Knn::ToBinaryStringFloat([1.f, 1.f, 1.f, 1.f, 1.f]), "FloatVector")),
    (2, Untag(Knn::ToBinaryStringFloat([1.f, 1.f, 1.f, 1.f, 1.25f]), "FloatVector")),
    (3, Untag(Knn::ToBinaryStringFloat([1.f, 1.f, 1.f, 1.f, 1.5f]), "FloatVector")),
    (4, Untag(Knn::ToBinaryStringFloat([-1.f, -1.f, -1.f, -1.f, -1.f]), "FloatVector")),
    (5, Untag(Knn::ToBinaryStringFloat([-2.f, -2.f, -2.f, -2.f, -4.f]), "FloatVector")),
    (6, Untag(Knn::ToBinaryStringFloat([-3.f, -3.f, -3.f, -3.f, -6.f]), "FloatVector"));
```

Описание функции `Knn::ToBinaryStringFloat` см. здесь.

В этих примерах преобразуются векторы, заданные в YQL. Если векторы передаются параметрами запроса из C++ SDK, сериализуйте их на клиенте с помощью `NYdb::NValueHelpers::Embedding` (C++ SDK v3.24.0 или новее) и объявляйте параметры как `Bytes`; см. рецепт для SDK.

Шаг 3. Построение векторного индекса

Для создания векторного индекса `EmbeddingIndex` на таблице `Vectors` нужно использовать команду:

```
ALTER TABLE Vectors
ADD INDEX EmbeddingIndex
GLOBAL USING vector_kmeans_tree
ON (embedding)
WITH (distance=cosine)
```

Данная команда создаёт индекс типа `vector_kmeans_tree`. В этом модельном примере указан параметр `distance=cosine`, задающий функцию расстояния — косинусное. Это важный параметр: используемая в индексе функция расстояния должна совпадать с той, на которой обучалась модель эмбеддингов. Параметры `clusters`, `levels`, `overlap_clusters` в этом примере не указаны — они определяются автоматически. Если данных много и есть необходимость тонко настроить индекс, см. подробное описание параметров в Векторный индекс.

Подробную информацию о векторных индексах и их параметрах смотрите в разделе Векторные индексы.

Шаг 4. Поиск в таблице без использования векторного индекса

На данном шаге выполняется точный поиск 3-х ближайших соседей для заданного вектора `[1.f, 1.f, 1.f, 1.f, 4.f]` **без** использования индекса.

Сначала целевой вектор кодируется в бинарное представление с помощью `Knn::ToBinaryStringFloat`.

Затем вычисляется косинусное расстояние от `embedding` каждой строки до целевого вектора.

Записи сортируются по возрастанию расстояния, и выбираются три (`$K`) первых записей, которые являются ближайшими.

```
$K = 3;
$TargetEmbedding = Knn::ToBinaryStringFloat([1.f, 1.f, 1.f, 1.f, 4.f]);

SELECT id, Knn::CosineDistance(embedding, $TargetEmbedding) As CosineDistance
FROM Vectors
ORDER BY Knn::CosineDistance(embedding, $TargetEmbedding)
LIMIT $K;
```

Результат выполнения запроса:

```
id CosineDistance
3  0.1055728197
```

```
2  0.1467181444
1  0.1999999881
```

Подробную информацию о точном векторном поиске без использования векторных индексов см. здесь.

Шаг 5. Поиск в таблице с использованием векторного индекса

Для поиска 3-х ближайших соседей вектора `[1.f, 1.f, 1.f, 1.f, 4.f]` с использованием индекса `EmbeddingIndex`, который был создан на шаге 3, нужно выполнить запрос:

```
PRAGMA ydb.KMeansTreeSearchTopSize="3";

$K = 3;
$TargetEmbedding = Knn::ToBinaryStringFloat([1.f, 1.f, 1.f, 1.f, 4.f]);

SELECT id, Knn::CosineDistance(embedding, $TargetEmbedding) As CosineDistance
FROM Vectors VIEW EmbeddingIndex
ORDER BY Knn::CosineDistance(embedding, $TargetEmbedding)
LIMIT $K;
```

Обратите внимание, что в данном запросе после названия таблицы указано, что отбор записей в ней следует производить с помощью векторного индекса: `FROM Vectors VIEW EmbeddingIndex`.

Результат выполнения запроса:

```
id CosineDistance
3  0.1055728197
2  0.1467181444
1  0.1999999881
```

Благодаря использованию индекса поиск ближайших векторов происходит значительно быстрее на больших выборках.

Заключение

Данная статья приводит простой пример работы с векторным индексом: создание таблицы с векторами, заполнение таблицы векторами, построение векторного индекса для такой таблицы и поиск вектора в таблице с использованием векторного индекса или без него.

В случае маленькой таблицы, как в этом модельном примере, невозможно увидеть разницу в производительности запросов. Эти примеры призваны проиллюстрировать синтаксис при работе с векторными индексами. Более реалистичный пример с бóльшим объёмом данных см. здесь.

Более подробную информацию о векторных индексах см. здесь.

Векторный индекс с загрузкой внешнего набора данных

Эта статья поможет научиться работать с векторными индексами в YDB. В качестве данных используется набор данных с текстами из Википедии на английском языке (485 859 строк), подготовленный сообществом Hugging Face.

В статье будут рассмотрены следующие шаги работы с векторным индексом:

- создание таблицы для загрузки данных;
- скачивание набора данных;
- импорт данных в таблицу;
- построение векторного индекса;
- выполнение векторного поиска без индекса;
- выполнение векторного поиска с индексом.

Пререквизиты

Для выполнения примеров из этой статьи понадобятся:

- Установленная БД YDB с включённой поддержкой векторных индексов. Об установке простого одноузлового кластера YDB можно прочитать здесь. Рекомендации по развёртыванию YDB для промышленного использования см. здесь.
- На рабочей машине:
  - Утилита YDB CLI;
  - Python 3;
- Необходим сетевой доступ с рабочей машины к хостам, где установлена YDB.

Шаг 1. Создание таблицы

Первым шагом необходимо создать таблицу в YDB, в которой будут храниться данные. Это можно сделать с помощью SQL-запроса:

```
CREATE TABLE wikipedia (
  id Uint64 NOT NULL,
  title Utf8,
  text Utf8,
  url Utf8,
  wiki_id Uint32,
  views Float,
  paragraph_id Uint32,
  langs Uint32,
  emb Utf8,
  embedding String,
  PRIMARY KEY (id)
);
```

Шаг 2. Скачивание набора данных

На этом шаге будет скачан набор данных, подготовленный сообществом Hugging Face. Это форк датасета Cohere `wikipedia-22-12-simple-embeddings`, который распространяется под лицензией Apache-2.0. Он содержит тексты из английской Википедии, разбитые на параграфы. Каждому параграфу сопоставлен вектор-эмбеддинг.

> **Примечание**
>
> В форке схема колонок отличается от таблицы в шаге 1: текст параграфа лежит в `contents`, а `title`, `url`, `wiki_id` и другие поля — в JSON-колонке `meta`. Скрипт ниже приводит данные к схеме этой таблицы.

В рабочей директории нужно создать файл `import_dataset.py`, содержащий скрипт на Python:

```python
#!/usr/bin/env python
import json
from datasets import load_dataset

# Load the dataset
dataset = load_dataset(
    "timescale/wikipedia-22-12-simple-embeddings",
    split="train",
)

# Flatten Timescale schema to match the wikipedia table
def flatten(row):
    meta = row["meta"]
    if isinstance(meta, str):
        meta = json.loads(meta)
    return {
        "id": row["id"],
        "title": meta.get("title"),
        "text": row["contents"],
        "url": meta.get("url"),
        "wiki_id": meta.get("wiki_id"),
        "views": meta.get("views"),
        "paragraph_id": meta.get("paragraph_id"),
        "langs": meta.get("langs"),
        # Space-separate floats for the YQL conversion in step 3
        "emb": row["embedding"].replace(",", " "),
    }

# Save it as CSV
```

```
flattened = dataset.map(flatten, remove_columns=dataset.column_names)
flattened.to_csv("wikipedia_embeddings_train.csv", index=False)
```

Далее следует установить пакет `datasets` и выполнить подготовленный скрипт `import_dataset.py`:

```
pip3 install datasets
python3 import_dataset.py
```

В рабочей директории будет создан файл `wikipedia_embeddings_train.csv`, содержащий набор данных с текстами из Википедии и эмбеддингами.

Шаг 3. Импорт данных из файла в таблицу

С помощью утилиты YDB CLI следует импортировать полученный файл `wikipedia_embeddings_train.csv` в созданную на Шаге 1 таблицу `wikipedia`.

В скрипте ниже:

- `<endpoint>` - эндпоинт к БД YDB
- `<database>` - путь к базе данных к БД YDB

```
ydb -e <endpoint> -d <database> -v import file csv --path wikipedia --header wikipedia_embeddings_train.csv --timeout 30
```

В данном примере тестовый сервер YDB был поднят без авторизации, поэтому при вызове YDB CLI не понадобилось передавать данные о пользователе. Об аутентификации с помощью YDB CLI см. здесь.

Размер файла `wikipedia_embeddings_train.csv` — 6,26 ГБ, поэтому его загрузка может занять некоторое время. По окончании загрузки можно убедиться, что все строки загружены в таблицу `wikipedia`, созданную на Шаге 1, следующим запросом:

```
SELECT COUNT(*) AS row_count FROM wikipedia
```

Результат выполнения запроса:

```
row_count
485859
```

Для построения векторного индекса в YDB вектора должны быть представлены в строковом формате (подробнее см. здесь). Эмбеддинги импортированного датасета в таблице `wikipedia` (которые записаны в столбце `emb`) могут быть преобразованы в столбец `embedding` нужного формата скриптом на рабочей машине:

```
for (( i=0; i <= 490000; i+=10000 ))
do
echo $i
echo "{\"begin\":$i}" | ydb -e <endpoint> -d <database> table query execute -q 'declare $begin As Int32; UPDATE wikipedia SET embedding = Unwrap(Untag(Knn::ToBinaryStringFloat(Cast(String::SplitToList(String::ReplaceAll(String::RemoveAll(String::RemoveFirst(String::RemoveLast(emb, "]"), "["), "\n"), "  ", " "), " ") AS List<Float>)), "FloatVector")) WHERE id>=$begin AND id < $begin + 10000;'
done
```

Шаг 4. Построение векторного индекса

Для создания векторного индекса `idx_vector` на таблице `wikipedia` нужно выполнить запрос:

```
ALTER TABLE wikipedia
ADD INDEX idx_vector
GLOBAL USING vector_kmeans_tree
ON (embedding)
WITH (
  distance=cosine);
```

Данный запрос создаёт индекс типа `vector_kmeans_tree`.

Подробную информацию о векторных индексах и их параметрах смотрите в разделе Векторные индексы.

Шаг 5. Поиск в таблице без использования векторного индекса

На данном шаге выполняется точный поиск 3-х ближайших соседей для заданного вектора **без** использования индекса. Предполагается, что входной текст для поиска в таблице `wikipedia` преобразован в данный вектор с помощью модели-энкодера (Embed от cohere.com).

Сначала целевой вектор кодируется в бинарное представление с помощью `Knn::ToBinaryStringFloat`.

Здесь поисковый вектор задан в YQL. Если он передаётся параметром запроса из C++ SDK, сериализуйте его на клиенте с помощью `NYdb::NValueHelpers::Embedding` (C++ SDK v3.24.0 или новее) и объявляйте параметр как `Bytes`; см. рецепт для SDK.

Затем вычисляется косинусное расстояние от `embedding` каждой строки до целевого вектора.

Записи сортируются по возрастанию расстояния, и выбираются три (`$K`) первых записей, которые являются ближайшими.

```
$K = 3;
$TargetEmbedding = Knn::ToBinaryStringFloat(Cast([0.1961289,0.51426697,0.03864574,0.5552187,-0.041873194,0.24177523,0.46322846,-0.3476358,-0.0802049,0.44246107,-0.06727136,-0.04970105,-0.0012320493,0.29773152,-0.377186
```

```
4,0.047693416,0.30664062,0.15911901,0.27795044,0.11875397,-0.056650203,0.33322853,-0.28901896,-0.43791273,-0.
014167095,0.36109218,-0.16923136,0.29162315,-0.22875166,0.122518055,0.030670911,-0.13762642,-0.13884683,0.314
55114,-0.21587017,0.32154146,-0.4452795,-0.058932953,0.07103838,0.4289945,-0.6023675,-0.14161813,0.11005565,
0.19201005,0.2591869,-0.24074492,0.18088372,-0.16547637,0.08194011,0.10669302,-0.049760908,0.15548608,0.01103
5396,0.16121127,-0.4862669,0.5691393,-0.4885568,0.90131176,0.20769958,0.010636337,-0.2094356,-0.15292564,-0.2
704138,-0.01326699,0.11226809,0.37113565,-0.018971693,0.86532146,0.28991342,0.004782651,-0.0024367527,-0.0861
291,0.39704522,0.25665164,-0.45121723,-0.2728092,0.1441502,-0.5042585,0.3507123,-0.38818485,0.5468399,0.16378
048,-0.11177127,0.5224827,-0.05927702,0.44906104,-0.036211397,-0.08465567,-0.33162776,0.25222498,-0.22274417,
0.15050206,-0.012386843,0.23640677,-0.18704978,0.1139806,0.19379948,-0.2326912,0.36477265,-0.2544955,0.271431
18,-0.095495716,-0.1727166,0.29109988,0.32738894,0.0016002139,0.052142758,0.37208632,0.034044757,0.17740013,
0.16472393,-0.20134833,0.055949032,-0.06671674,0.04691583,0.13196157,-0.13174891,-0.17132106,-0.4257385,-1.10
67779,0.55262613,0.37117195,-0.37033138,-0.16229,-0.31594914,-0.87293816,0.62064904,-0.32178572,0.28461748,0.
41640115,-0.050539408,0.009697271,0.3483608,0.4401717,-0.08273758,0.4873984,0.057845585,0.28128678,-0.4395515
6,-0.18790118,0.40001884,0.54413813,0.054571174,0.65416795,0.04503013,0.40744695,-0.048226677,0.4787822,0.097
00139,0.07739511,0.6503141,0.39685145,-0.54047453,0.041596334,-0.22190939,0.25528133,0.17406437,-0.17308964,
0.22076453,0.31207982,0.8434676,0.2086337,-0.014262581,0.05081182,-0.30908328,-0.35717097,0.17224313,0.526684
6,0.58924395,-0.29272506,0.01910475,0.061457288,0.18099669,0.04807291,0.34706554,0.32477927,0.17174402,-0.070
991516,0.5819317,0.71045977,0.07172716,0.32184732,0.19009985,0.04727492,0.3004647,0.26943457,0.61640364,0.165
5051,-0.6033329,0.09797926,-0.20623252,0.10987298,1.016591,-0.29540864,0.25161317,0.19790122,0.14642714,0.508
1536,-0.22128952,0.4286613,-0.029895071,0.23768105,-0.0023987228,0.086968,0.42884818,-0.33578634,-0.38033295,
-0.16163215,-0.18072455,-0.5015756,0.28035417,-0.0066010267,0.67613393,-0.026721207,0.22796173,-0.008428602,-
0.38017297,-0.33044866,0.4519961,-0.05542353,-0.2976922,0.37046987,0.23409955,-0.24246313,-0.12839256,-0.4206
849,-0.049280513,-0.7651326,0.1649417,-0.2321146,0.106625736,-0.37506104,0.14470209,-0.114986554,-0.17738944,
0.612335,0.25292027,-0.092776075,-0.3876576,-0.08905502,0.3793106,0.7376429,-0.3080258,-0.3869677,0.5239047,-
0.41152182,0.22852719,0.42226496,-0.28244498,0.0651847,0.3525671,-0.5396397,-0.17514983,0.29470462,-0.4767109
8,0.43471992,0.38677526,0.054752454,0.2183725,0.06853758,-0.12792642,0.67841107,0.24607432,0.18936129,0.24056
062,-0.30873874,0.62442464,0.5792256,0.20426203,0.54328054,0.56583667,-0.7724596,-0.08384111,-0.16767848,-0.2
1682987,0.05710991,-0.015403866,0.38889074,-0.6050326,0.4075437,0.40839496,0.2507789,-0.32695654,0.24276069,
0.1271161,-0.010688765,-0.31864303,0.15747054,-0.4670915,-0.21059138,0.7470888,0.47273478,-0.119508654,-0.636
59865,0.64500844,0.5370401,0.28596714,0.0046216915,0.12771192,-0.18660222,0.47342712,-0.32039297,0.10946048,
0.25172964,0.021965463,-0.12397459,-0.048939236,0.2881649,-0.61231786,-0.33459276,-0.29495123,-0.14027011,-0.
23020774,0.73250633,0.71871173,0.78408533,0.4140183,0.1398299,0.7395877,0.06801048,-0.8895956,-0.64981127,-0.
37226167,0.1905936,0.12819989,-0.47098637,-0.14334664,-0.933116,0.4597078,0.09895813,0.38114703,0.14368558,-
0.42793563,-0.10805895,0.025374172,0.40162122,-0.1686769,0.5257471,-0.3540743,0.08181256,-0.34759146,0.005307
8625,0.09163392,0.074487045,-0.14934056,0.034427803,0.19613744,-0.00032829077,0.27792764,0.09889235,-0.029708
104,0.3528952,0.22679164,-0.27263018,0.6655268,-0.21362385,0.13035864,0.41666874,0.1253278,-0.22861275,0.1050
85365,0.09412938,0.03228179,0.11568338,0.23504587,-0.044100706,0.0104857525,-0.07461301,0.1034835,0.3078725,
0.5257031,-0.015183647,-0.0060899477,-0.02852683,-0.39821762,-0.20495597,-0.14892153,0.44850922,0.40366673,-
0.10324784,0.4095244,0.8356313,0.21190739,-0.12822983,0.06830399,0.036365107,0.044244137,0.26112562,0.0334776
27,-0.41074416,-0.009961431,0.23717403,0.12438699,-0.05255729,-0.18411024,-0.18563229,-0.16543737,-0.12230024
5,0.40962145,-0.4751102,0.5309857,0.04474563,0.103834346,0.14118321,4.2373734,0.45751426,0.21709882,0.686677
8,0.14838168,-0.1831362,0.10963214,-0.33557487,-0.1084519,0.3299757,0.076113895,0.12850489,-0.07326015,-0.237
70756,0.11080451,0.29712623,-0.13904962,0.25797644,-0.5074562,0.4018296,-0.23186816,0.24427155,0.39540753,0.0
15477164,0.14021018,0.273185,0.013538655,0.47227964,0.52339536,0.54428,0.16983595,0.5470162,-0.0042650895,0.2
1768,0.090606116,-0.13433483,0.5818122,-0.1384567,0.2354754,0.08440857,-0.2166868,0.48664945,-0.13175073,0.45
613387,0.089229666,0.15436831,0.08720108,0.37597507,0.52855235,-0.019367872,0.544358,-0.327109,-0.20839518,-
0.33598265,0.033363096,0.42312673,0.13452567,0.40526676,0.08402101,-0.19661862,-0.24802914,0.23069139,0.51535
08,0.13562717,-0.23842931,-0.23257096,-0.009195984,0.41388315,0.56304437,-0.23492545,-0.2642354,0.3038204,-0.
09548942,-0.22467934,-0.2561862,-0.34057313,-0.19744347,0.0007430283,-0.12842518,-0.13980682,0.6849243,0.1795
335,-0.5626032,-0.07626079,-0.062749654,0.6660117,-0.4479761,0.07978033,0.6269782,0.536793,0.6801336,-0.22563
715,0.38902125,-0.09493616,0.21312712,0.17763247,0.1796997,-3.868085,0.08134122,0.10347531,-0.034904435,-0.27
92477,-0.17850947,0.083218865,0.26535586,-0.25551575,0.28172702,0.1383222,0.10376686,-0.123248994,0.1985073,-
0.40000066,0.44763976,0.028454497,0.37575415,0.071487874,-0.16965964,0.38927504,0.29088503,-0.011822928,-0.19
522227,-0.1766321,0.1731763,0.49192554,0.44358602,-0.49064636,0.024170646,0.025736902,-0.17963372,0.38337404,
0.07339889,0.042465065,0.5910191,0.07904464,-0.043729525,-0.16969916,0.4008944,-0.04921039,-0.3757768,0.60753
14,-0.24661873,-0.1780646,0.60300773,-0.09518917,0.2213779,-0.46496615,-0.41421738,0.23309247,0.14687467,-0.3
6499617,0.04227981,0.88024706,0.57489127,0.21026954,-0.13666761,0.05710815,0.22095469,-0.033460964,0.1386156
1,0.22527887,0.1660716,-0.3286249,-0.060175333,-0.2971499,0.2454142,0.6536238,-0.22991207,0.046677545,-0.0266
31566,-0.04271381,-0.53681016,0.11866242,-0.24970472,-0.37882543,0.33650783,0.7634871,-0.2858582,0.029164914,
0.28833458,-0.39263156,0.64842117,2.6358266,0.058920268,2.2507918,0.6809379,-0.41290292,0.36954543,-0.6079356
7,0.42561662,0.2498035,0.27133986,-0.005307673,0.32910514,-0.03169463,-0.02270061,-0.14702365,-0.25256258,0.5
4468036,-0.46112943,-0.07411629,-0.030253865,0.20578359,0.6495886,-0.11674013,0.029835526,0.019896187,-0.0081
01909,0.3706806,-0.26088533,-0.018712807,0.17228629,0.15223767,0.0675542,0.6338221,-0.15303946,0.02908536,0.2
7217266,-0.10829474,4.503505,-0.37745082,0.20543274,-0.087563366,-0.14404398,0.5562983,0.41639867,-0.3819121
4,-0.16266975,-0.46071815,0.51874137,0.36326376,0.027115177,-0.06804209,0.35159302,-0.41162485,0.30493516,0.1
8828706,0.63608,-0.04735176,0.13811842,0.09368063,0.037441075,-0.0012712433,-0.19929455,0.34804425,0.4697542
8,0.38857734,-0.061463855,0.122808196,0.37608445,5.2436657,0.25659403,-0.19236223,-0.25611007,0.22265173,0.58
98642,-0.28255892,-0.4123271,-0.4214137,0.09197922,-0.060595497,-0.13819462,-0.13570791,0.25433356,0.5907837,
0.2548469,-0.39375016,-0.37651995,0.701745,-0.0359955,-0.048193086,0.4458719,0.088069156,-0.015497342,0.52568
024,-0.4795603,-0.025876174,0.76476455,-0.32245165,-0.038828112,0.6325802,0.06385053,-0.26389623,0.2439906,-
0.4231506,0.19213657,0.5828574,0.053197365,0.45217928,0.040650904,0.83714896,0.63782233,-0.737095,-0.4102670
6,0.23113042,0.19471557,-0.24410644,-0.35155243,0.20881484,-0.01721743,-0.29494065,-0.114185065,1.2226206,-0.
16469914,0.083336286,0.63608664,0.41011855,-0.032080106,-0.08833447,-0.6261006,0.22665286,0.08313674,-0.16372
047,0.5235312,0.39580458,0.0007253827,0.10186727,-0.15955615,0.54162663,0.32992217,-0.02491269,0.16312002,0.1
18171245,-0.029900813,0.038405042,0.31396118,0.45241603,-0.07010825,0.07611299,0.084779754,0.34168348,-0.6067
6336,0.054825004,-0.16054128,0.2525291,0.20532744,-0.1510394,0.4857572,0.32150552,0.35749313,0.4483151,0.0057
622716,0.28705776,-0.018361313,0.08605509,-0.08649293,0.26918742,0.4806176,0.098294765,0.3284613,0.0001066465
6,0.43832678,-0.33351916,0.02354738,0.004953976,-0.14319824,-0.33351237,-0.7268964,0.56292313,0.1275613,0.443
8945,0.7984555,-0.19372283,0.2940397,-0.11770557] AS List<Float>));

SELECT id, title, text, wiki_id, Knn::CosineDistance(embedding, $TargetEmbedding) as CosineDistance
FROM wikipedia VIEW PRIMARY KEY
ORDER BY Knn::CosineDistance(embedding, $TargetEmbedding)
LIMIT $K;
```

Результат выполнения запроса:

```
id   title          text                                        wiki_id CosineDistance
1    24-hour clock  A time in the 24-hour clock is written...   9985    0
```

```
2    24-hour clock  However, the US military prefers not to...  9985    0.06253927946
322  12-hour clock  Both names are from Latin, and numbered...  55462   0.07501709461
```

Подробную информацию о точном векторном поиске без использования векторных индексов см. в документации по Knn UDF.

Шаг 6. Поиск в таблице с использованием векторного индекса

Для поиска 3-х ближайших соседей заданного вектора с использованием индекса `idx_vector`, созданного на шаге 4, нужно выполнить запрос:

```
PRAGMA ydb.KMeansTreeSearchTopSize="3";

$K = 3;
$TargetEmbedding = Knn::ToBinaryStringFloat(Cast([0.1961289,0.51426697,0.03864574,0.5552187,-0.041873194,0.24
177523,0.46322846,-0.3476358,-0.0802049,0.44246107,-0.06727136,-0.04970105,-0.0012320493,0.29773152,-0.377186
4,0.047693416,0.30664062,0.15911901,0.27795044,0.11875397,-0.056650203,0.33322853,-0.28901896,-0.43791273,-0.
014167095,0.36109218,-0.16923136,0.29162315,-0.22875166,0.122518055,0.030670911,-0.13762642,-0.13884683,0.314
55114,-0.21587017,0.32154146,-0.4452795,-0.058932953,0.07103838,0.4289945,-0.6023675,-0.14161813,0.11005565,
0.19201005,0.2591869,-0.24074492,0.18088372,-0.16547637,0.08194011,0.10669302,-0.049760908,0.15548608,0.01103
5396,0.16121127,-0.4862669,0.5691393,-0.4885568,0.90131176,0.20769958,0.010636337,-0.2094356,-0.15292564,-0.2
704138,-0.01326699,0.11226809,0.37113565,-0.018971693,0.86532146,0.28991342,0.004782651,-0.0024367527,-0.0861
291,0.39704522,0.25665164,-0.45121723,-0.2728092,0.1441502,-0.5042585,0.3507123,-0.38818485,0.5468399,0.16378
048,-0.11177127,0.5224827,-0.05927702,0.44906104,-0.036211397,-0.08465567,-0.33162776,0.25222498,-0.22274417,
0.15050206,-0.012386843,0.23640677,-0.18704978,0.1139806,0.19379948,-0.2326912,0.36477265,-0.2544955,0.271431
18,-0.095495716,-0.1727166,0.29109988,0.32738894,0.0016002139,0.052142758,0.37208632,0.034044757,0.17740013,
0.16472393,-0.20134833,0.055949032,-0.06671674,0.04691583,0.13196157,-0.13174891,-0.17132106,-0.4257385,-1.10
67779,0.55262613,0.37117195,-0.37033138,-0.16229,-0.31594914,-0.87293816,0.62064904,-0.32178572,0.28461748,0.
41640115,-0.050539408,0.009697271,0.3483608,0.4401717,-0.08273758,0.4873984,0.057845585,0.28128678,-0.4395515
6,-0.18790118,0.40001884,0.54413813,0.054571174,0.65416795,0.04503013,0.40744695,-0.048226677,0.4787822,0.097
00139,0.07739511,0.6503141,0.39685145,-0.54047453,0.041596334,-0.22190939,0.25528133,0.17406437,-0.17308964,
0.22076453,0.31207982,0.8434676,0.2086337,-0.014262581,0.05081182,-0.30908328,-0.35717097,0.17224313,0.526684
6,0.58924395,-0.29272506,0.01910475,0.061457288,0.18099669,0.04807291,0.34706554,0.32477927,0.17174402,-0.070
991516,0.5819317,0.71045977,0.07172716,0.32184732,0.19009985,0.04727492,0.3004647,0.26943457,0.61640364,0.165
5051,-0.6033329,0.09797926,-0.20623252,0.10987298,1.016591,-0.29540864,0.25161317,0.19790122,0.14642714,0.508
1536,-0.22128952,0.4286613,-0.029895071,0.23768105,-0.0023987228,0.086968,0.42884818,-0.33578634,-0.38033295,
-0.16163215,-0.18072455,-0.5015756,0.28035417,-0.0066010267,0.67613393,-0.026721207,0.22796173,-0.008428602,-
0.38017297,-0.33044866,0.4519961,-0.05542353,-0.2976922,0.37046987,0.23409955,-0.24246313,-0.12839256,-0.4206
849,-0.049280513,-0.7651326,0.1649417,-0.2321146,0.106625736,-0.37506104,0.14470209,-0.114986554,-0.17738944,
0.612335,0.25292027,-0.092776075,-0.3876576,-0.08905502,0.3793106,0.7376429,-0.3080258,-0.3869677,0.5239047,-
0.41152182,0.22852719,0.42226496,-0.28244498,0.0651847,0.3525671,-0.5396397,-0.17514983,0.29470462,-0.4767109
8,0.43471992,0.38677526,0.054752454,0.2183725,0.06853758,-0.12792642,0.67841107,0.24607432,0.18936129,0.24056
062,-0.30873874,0.62442464,0.5792256,0.20426203,0.54328054,0.56583667,-0.7724596,-0.08384111,-0.16767848,-0.2
1682987,0.05710991,-0.015403866,0.38889074,-0.6050326,0.4075437,0.40839496,0.2507789,-0.32695654,0.24276069,
0.1271161,-0.010688765,-0.31864303,0.15747054,-0.4670915,-0.21059138,0.7470888,0.47273478,-0.119508654,-0.636
59865,0.64500844,0.5370401,0.28596714,0.0046216915,0.12771192,-0.18660222,0.47342712,-0.32039297,0.10946048,
0.25172964,0.021965463,-0.12397459,-0.048939236,0.2881649,-0.61231786,-0.33459276,-0.29495123,-0.14027011,-0.
23020774,0.73250633,0.71871173,0.78408533,0.4140183,0.1398299,0.7395877,0.06801048,-0.8895956,-0.64981127,-0.
37226167,0.1905936,0.12819989,-0.47098637,-0.14334664,-0.933116,0.4597078,0.09895813,0.38114703,0.14368558,-
0.42793563,-0.10805895,0.025374172,0.40162122,-0.1686769,0.5257471,-0.3540743,0.08181256,-0.34759146,0.005307
8625,0.09163392,0.074487045,-0.14934056,0.034427803,0.19613744,-0.00032829077,0.27792764,0.09889235,-0.029708
104,0.3528952,0.22679164,-0.27263018,0.6655268,-0.21362385,0.13035864,0.41666874,0.1253278,-0.22861275,0.1050
85365,0.09412938,0.03228179,0.11568338,0.23504587,-0.044100706,0.0104857525,-0.07461301,0.1034835,0.3078725,
0.5257031,-0.015183647,-0.0060899477,-0.02852683,-0.39821762,-0.20495597,-0.14892153,0.44850922,0.40366673,-
0.10324784,0.4095244,0.8356313,0.21190739,-0.12822983,0.06830399,0.036365107,0.044244137,0.26112562,0.0334776
27,-0.41074416,-0.009961431,0.23717403,0.12438699,-0.05255729,-0.18411024,-0.18563229,-0.16543737,-0.12230024
5,0.40962145,-0.4751102,0.5309857,0.04474563,0.103834346,0.14118321,4.2373734,0.45751426,0.21709882,0.686677
8,0.14838168,-0.1831362,0.10963214,-0.33557487,-0.1084519,0.3299757,0.076113895,0.12850489,-0.07326015,-0.237
70756,0.11080451,0.29712623,-0.13904962,0.25797644,-0.5074562,0.4018296,-0.23186816,0.24427155,0.39540753,0.0
15477164,0.14021018,0.273185,0.013538655,0.47227964,0.52339536,0.54428,0.16983595,0.5470162,-0.0042650895,0.2
1768,0.090606116,-0.13433483,0.5818122,-0.1384567,0.2354754,0.08440857,-0.2166868,0.48664945,-0.13175073,0.45
613387,0.089229666,0.15436831,0.08720108,0.37597507,0.52855235,-0.019367872,0.544358,-0.327109,-0.20839518,-
0.33598265,0.033363096,0.42312673,0.13452567,0.40526676,0.08402101,-0.19661862,-0.24802914,0.23069139,0.51535
08,0.13562717,-0.23842931,-0.23257096,-0.009195984,0.41388315,0.56304437,-0.23492545,-0.2642354,0.3038204,-0.
09548942,-0.22467934,-0.2561862,-0.34057313,-0.19744347,0.0007430283,-0.12842518,-0.13980682,0.6849243,0.1795
335,-0.5626032,-0.07626079,-0.062749654,0.6660117,-0.4479761,0.07978033,0.6269782,0.536793,0.6801336,-0.22563
715,0.38902125,-0.09493616,0.21312712,0.17763247,0.1796997,-3.868085,0.08134122,0.10347531,-0.034904435,-0.27
92477,-0.17850947,0.083218865,0.26535586,-0.25551575,0.28172702,0.1383222,0.10376686,-0.123248994,0.1985073,-
0.40000066,0.44763976,0.028454497,0.37575415,0.071487874,-0.16965964,0.38927504,0.29088503,-0.011822928,-0.19
522227,-0.1766321,0.1731763,0.49192554,0.44358602,-0.49064636,0.024170646,0.025736902,-0.17963372,0.38337404,
0.07339889,0.042465065,0.5910191,0.07904464,-0.043729525,-0.16969916,0.4008944,-0.04921039,-0.3757768,0.60753
14,-0.24661873,-0.1780646,0.60300773,-0.09518917,0.2213779,-0.46496615,-0.41421738,0.23309247,0.14687467,-0.3
6499617,0.04227981,0.88024706,0.57489127,0.21026954,-0.13666761,0.05710815,0.22095469,-0.033460964,0.1386156
1,0.22527887,0.1660716,-0.3286249,-0.060175333,-0.2971499,0.2454142,0.6536238,-0.22991207,0.046677545,-0.0266
31566,-0.04271381,-0.53681016,0.11866242,-0.24970472,-0.37882543,0.33650783,0.7634871,-0.2858582,0.029164914,
0.28833458,-0.39263156,0.64842117,2.6358266,0.058920268,2.2507918,0.6809379,-0.41290292,0.36954543,-0.6079356
7,0.42561662,0.2498035,0.27133986,-0.005307673,0.32910514,-0.03169463,-0.02270061,-0.14702365,-0.25256258,0.5
4468036,-0.46112943,-0.07411629,-0.030253865,0.20578359,0.6495886,-0.11674013,0.029835526,0.019896187,-0.0081
01909,0.3706806,-0.26088533,-0.018712807,0.17228629,0.15223767,0.0675542,0.6338221,-0.15303946,0.02908536,0.2
7217266,-0.10829474,4.503505,-0.37745082,0.20543274,-0.087563366,-0.14404398,0.5562983,0.41639867,-0.3819121
4,-0.16266975,-0.46071815,0.51874137,0.36326376,0.027115177,-0.06804209,0.35159302,-0.41162485,0.30493516,0.1
8828706,0.63608,-0.04735176,0.13811842,0.09368063,0.037441075,-0.0012712433,-0.19929455,0.34804425,0.4697542
8,0.38857734,-0.061463855,0.122808196,0.37608445,5.2436657,0.25659403,-0.19236223,-0.25611007,0.22265173,0.58
98642,-0.28255892,-0.4123271,-0.4214137,0.09197922,-0.060595497,-0.13819462,-0.13570791,0.25433356,0.5907837,
0.2548469,-0.39375016,-0.37651995,0.701745,-0.0359955,-0.048193086,0.4458719,0.088069156,-0.015497342,0.52568
024,-0.4795603,-0.025876174,0.76476455,-0.32245165,-0.038828112,0.6325802,0.06385053,-0.26389623,0.2439906,-
0.4231506,0.19213657,0.5828574,0.053197365,0.45217928,0.040650904,0.83714896,0.63782233,-0.737095,-0.4102670
6,0.23113042,0.19471557,-0.24410644,-0.35155243,0.20881484,-0.01721743,-0.29494065,-0.114185065,1.2226206,-0.
16469914,0.083336286,0.63608664,0.41011855,-0.032080106,-0.08833447,-0.6261006,0.22665286,0.08313674,-0.16372
047,0.5235312,0.39580458,0.0007253827,0.10186727,-0.15955615,0.54162663,0.32992217,-0.02491269,0.16312002,0.1
18171245,-0.029900813,0.038405042,0.31396118,0.45241603,-0.07010825,0.07611299,0.084779754,0.34168348,-0.6067
6336,0.054825004,-0.16054128,0.2525291,0.20532744,-0.1510394,0.4857572,0.32150552,0.35749313,0.4483151,0.0057
```

```
622716,0.28705776,-0.018361313,0.08605509,-0.08649293,0.26918742,0.4806176,0.098294765,0.3284613,0.0001066465
6,0.43832678,-0.33351916,0.02354738,0.004953976,-0.14319824,-0.33351237,-0.7268964,0.56292313,0.1275613,0.443
8945,0.7984555,-0.19372283,0.2940397,-0.11770557] AS List<Float>));

SELECT id, title, text, wiki_id, Knn::CosineDistance(embedding, $TargetEmbedding) as CosineDistance
FROM wikipedia VIEW idx_vector
ORDER BY Knn::CosineDistance(embedding, $TargetEmbedding)
LIMIT $K;
```

Результат выполнения запроса:

```
id   title          text                                      wiki_id CosineDistance
1    24-hour clock  A time in the 24-hour clock is written...  9985    0
2    24-hour clock  However, the US military prefers not to...  9985    0.06253927946
0    24-hour clock  The 24-hour clock is a way of telling...    9985    0.07683444023
```

Легко заметить, что результат запроса в случае использования векторного индекса отличается от результата запроса без использования векторного индекса.

Природа векторного индекса такова, что он обеспечивает приближённый векторный поиск за более короткое время. Приближённый поиск отбрасывает часть данных из рассмотрения для повышения эффективности, однако небольшая часть отброшенных данных могла фактически быть близка к целевому вектору, из-за чего результат приближённого поиска может отличаться от результата точного поиска.

В зависимости от размеров наборов данных время выполнения одинаковых запросов с векторным индексом и без него может отличаться на несколько порядков.

Заключение

Данная статья приводит пример работы с векторным индексом с загрузкой внешних данных: создание таблицы с векторами, заполнение таблицы векторами из внешнего набора данных, построение векторного индекса для такой таблицы и поиск вектора в таблице с использованием векторного индекса или без него.

Точный векторный поиск позволяет найти действительно ближайших соседей, но при этом стоимость вычислений оказывается высокой, особенно на больших массивах данных. Приближённые методы поиска, напротив, позволяют получать результаты значительно быстрее, хоть и с некоторой потерей точности, что может быть оправдано в реальных сценариях применения.

Более подробную информацию о векторных индексах см. здесь.

Рецепты полнотекстового поиска

Этот раздел содержит рецепты по использованию полнотекстовых индексов и функций полнотекстового поиска.

- Полнотекстовый индекс — быстрый старт

Полнотекстовый индекс — быстрый старт

В этом руководстве показано, как создать полнотекстовый индекс и выполнять полнотекстовые запросы в YDB.

Создайте таблицу и полнотекстовый индекс

```
CREATE TABLE articles (
  id Uint64,
  title String,
  body String,
  PRIMARY KEY (id)
);

ALTER TABLE articles ADD INDEX ft_idx
  GLOBAL USING fulltext_relevance
  ON (body) COVER (title)
  WITH (tokenizer=standard, use_filter_lowercase=true);
```

Добавьте тестовые данные

```
UPSERT INTO articles (id, title, body) VALUES
  (1, "Введение", "Машинное обучение — область ИИ"),
  (2, "Поиск",    "Полнотекстовый поиск находит документы по словам"),
  (3, "Заметки",  "Нейронные сети используются в машинном обучении");
```

Фильтрация документов

```
SELECT id, title
FROM articles VIEW ft_idx
WHERE FulltextMatch(body, "машинное обучение")
LIMIT 20;
```

Результат выполнения запроса:

```
id title
1  Введение
```

Ранжирование документов

```
SELECT id, title, FulltextScore(body, "машинное обучение") AS relevance
FROM articles VIEW ft_idx
WHERE FulltextScore(body, "машинное обучение") > 0
ORDER BY relevance DESC
LIMIT 10;
```

Результат выполнения запроса:

```
id title relevance
1  Введение 1.6215210957338408
```

Подробнее:

- Полнотекстовые индексы
- VIEW (Полнотекстовый индекс)
- Базовые функции полнотекстового поиска

Рецепты поиска по JSON-документам

Этот раздел содержит готовые рецепты по использованию JSON-индексов для эффективного отбора данных по содержимому колонок типа `Json` и `JsonDocument`.

Рецепты построены вокруг функций JSON_EXISTS и JSON_VALUE с выражениями JsonPath. Для гарантированного использования индекса все примеры используют явное обращение через `VIEW IndexName`.

- JSON-индекс — быстрый старт — минимальный сценарий: создание таблицы, индекса, вставка данных и базовые запросы.
- Каталог со вложенными атрибутами — пример каталога товаров с вложенными атрибутами, диапазонными условиями и поиском по вложенным массивам.
- Параметризованные запросы и переменные JsonPath — параметризованные запросы и передача переменных в JsonPath через секцию `PASSING`.
- Проверка типа поля и наличия пути — проверка типа поля и наличия пути с помощью методов JsonPath.

Перед запуском примеров убедитесь, что на кластере включена поддержка JSON-индексов.

JSON-индекс — быстрый старт

В этом руководстве показано, как создать JSON-индекс и выполнять запросы с использованием функций JSON_EXISTS и JSON_VALUE в YDB.

Создайте таблицу и JSON-индекс

```
CREATE TABLE documents (
    id Uint64,
    payload JsonDocument,
    PRIMARY KEY (id),
    INDEX json_idx GLOBAL USING json ON (payload)
);
```

Тип колонки `JsonDocument` хранит JSON в компактном бинарном формате и предпочтителен для индексируемой колонки. Альтернативно может использоваться тип `Json` (текстовое представление).

Первичный ключ таблицы должен состоять из единственной колонки целочисленного типа (`Uint64`, `Uint32`, `Int64` или `Int32`) — это текущее ограничение реализации JSON-индексов.

Добавьте тестовые данные

```
UPSERT INTO documents (id, payload) VALUES
    (1, JsonDocument(@@{"user": {"id": 100, "name": "Alice"}, "active": true}@@)),
    (2, JsonDocument(@@{"user": {"id": 101, "name": "Bob"}, "active": false}@@)),
    (3, JsonDocument(@@{"user": {"id": 102, "name": "Charlie"}, "archived": true}@@));
```

Здесь конструкция `@@...@@` — это многострочный строковый литерал, удобный для записи JSON без экранирования кавычек. Функция `JsonDocument(...)` преобразует текст в значение типа `JsonDocument`.

Фильтр по наличию пути в документе

Функция JSON_EXISTS проверяет, существует ли в документе путь, заданный выражением JsonPath.

```
SELECT id
FROM documents VIEW json_idx
WHERE JSON_EXISTS(payload, '$.user.id');
```

Результат:

```
id
1
2
3
```

Для поиска по индексу используется токен пути `$.user.id`. Индекс возвращает результат без сканирования основной таблицы.

Отбор строк с конкретным значением поля документа

Функция JSON_VALUE извлекает скалярное значение по JsonPath; для использования индекса обязательно нужно указать тип возвращаемого значения в секции `RETURNING`:

```
SELECT id
FROM documents VIEW json_idx
WHERE JSON_VALUE(payload, '$.user.name' RETURNING Utf8) = "Alice"u;
```

Результат:

```
id
1
```

При проверке равенства в индекс попадает токен «путь + значение» (`$.user.name = "Alice"`), что обеспечивает наивысшую селективность.

Комбинация условий

Несколько вызовов `JSON_EXISTS` / `JSON_VALUE` на одной индексированной JSON-колонке можно объединять операторами `AND` и `OR`:

```
SELECT id
FROM documents VIEW json_idx
WHERE JSON_EXISTS(payload, '$.user.id')
  AND JSON_VALUE(payload, '$.active' RETURNING Bool);
```

Результат:

```
id
1
```

Подробнее

- JSON-индексы — полный обзор возможностей и ограничений.

- VIEW (JSON-индекс) — синтаксис запросов через `VIEW`.
- INDEX (CREATE TABLE) — синтаксис создания JSON-индекса.
- Каталог со вложенными атрибутами — пример каталога товаров с вложенными атрибутами.

Каталог со вложенными атрибутами

В этом рецепте показано, как использовать JSON-индекс для ускорения доступа к каталогу товаров, в котором характеристики товара хранятся в виде JSON-документа с произвольным набором полей. Такая схема удобна, когда:

- набор атрибутов товара заранее неизвестен или различается для разных категорий;
- добавлять новые атрибуты через `ALTER TABLE ADD COLUMN` нежелательно;
- нужно эффективно фильтровать товары по различным сочетаниям атрибутов.

JSON-индекс позволяет добиться эффективной выборки по любому пути и значению внутри JSON-документа без полного сканирования таблицы.

Создайте таблицу и индекс

```
CREATE TABLE products (
    sku_id Uint64,
    attrs JsonDocument,
    PRIMARY KEY (sku_id),
    INDEX attrs_json_idx GLOBAL USING json ON (attrs)
);
```

В этой схеме:

- `sku_id` — числовой идентификатор товара.
- `attrs` — атрибуты товара в формате `JsonDocument`. Хранение в виде `JsonDocument` экономит место и ускоряет десериализацию по сравнению с `Json`.
- `attrs_json_idx` — JSON-индекс по колонке `attrs`. Обновляется синхронно вместе с основной таблицей.

Загрузите тестовые данные

```
UPSERT INTO products (sku_id, attrs) VALUES
    (10, JsonDocument(@@{
        "brand": "ACME",
        "price": 49.90,
        "category": "tools",
        "warehouses": [{"id": 1, "stock": 12}, {"id": 2, "stock": 0}]
    }@@)),
    (11, JsonDocument(@@{
        "brand": "ACME",
        "price": 199.00,
        "category": "electronics",
        "warehouses": [{"id": 1, "stock": 3}]
    }@@)),
    (12, JsonDocument(@@{
        "brand": "Globex",
        "price": 25.00,
        "category": "tools",
        "warehouses": [{"id": 2, "stock": 0}]
    }@@));
```

Фильтр по бренду и диапазону цены

```
SELECT sku_id, attrs
FROM products VIEW attrs_json_idx
WHERE JSON_VALUE(attrs, '$.brand' RETURNING Utf8) = "ACME"u
  AND JSON_VALUE(attrs, '$.price' RETURNING Double) BETWEEN 10.0 AND 100.0;
```

Что происходит:

- Для условия `$.brand = "ACME"` в индекс попадает токен «путь + значение» — это даёт точное совпадение и максимальную селективность.
- Для условия `BETWEEN 10.0 AND 100.0` в индекс попадает только токен пути `$.price`. Это сужает множество строк до тех, у которых поле `price` присутствует, после чего движок исполнения запросов проверяет диапазон точным сравнением (пост-фильтр).
- Условия объединены через `AND` — это позволяет использовать индекс для проверки сразу обоих фрагментов, в результате чтение по индексу будет минимальным.

Результат:

```
sku_id attrs
10     {"brand":"ACME","price":49.9,...}
```

Поиск по вложенному массиву

JsonPath поддерживает обращение к элементам массивов и фильтры внутри пути. Например, можно найти товары, у которых хотя бы на одном складе есть остаток:

```
SELECT sku_id
FROM products VIEW attrs_json_idx
WHERE JSON_EXISTS(attrs, '$.warehouses ? (@.stock > 0)');
```

Для поиска по индексу используется токен пути `$.warehouses.stock`, а условие `@.stock > 0` проверяется пост-фильтром на каждой найденной записи. Это эффективно, когда соответствующее поле присутствует у сравнительно малой части документов.

Результат:

```
sku_id
10
11
```

Поиск по категории и наличию атрибута

Условия можно комбинировать любыми операторами `AND` и `OR`. Поиск товаров в категории `tools`, у которых указано поле `price`:

```
SELECT sku_id
FROM products VIEW attrs_json_idx
WHERE JSON_VALUE(attrs, '$.category' RETURNING Utf8) = "tools"u
  AND JSON_EXISTS(attrs, '$.price');
```

Результат:

```
sku_id
10
12
```

Подробнее

- Поддерживаемые предикаты JSON-индекса — полные правила того, какие выражения индексируются.
- Обработка AND и OR — нюансы комбинирования условий, в том числе с неиндексируемыми предикатами.
- Параметризованные запросы и переменные JsonPath — параметризованные варианты этих же запросов.

Параметризованные запросы и переменные JsonPath

В большинстве приложений входные данные запроса не подставляются в текст SQL, а передаются через параметры. Типичные варианты использования параметров для работы с JSON-индексами:

1. прямое сравнение результата `JSON_VALUE` с параметром;
2. проверка наличия результата в списке значений через выражение `IN`;
3. передача параметра в выражение JsonPath через секцию `PASSING`.

Подготовка

Ниже приведён пример таблицы, JSON-индекса и заполнения данных.

```
CREATE TABLE documents (
    id Uint64,
    payload JsonDocument,
    PRIMARY KEY (id),
    INDEX json_idx GLOBAL USING json ON (payload)
);

UPSERT INTO documents (id, payload) VALUES
    (1, JsonDocument(@@{
        "owner_id": 100,
        "tag": "active",
        "archived": false,
        "content": {"x": 1, "y": 1}
    }@@)),
    (2, JsonDocument(@@{
        "owner_id": 100,
        "tag": "draft",
        "archived": false,
        "content": {"x": 1, "y": 2}
    }@@)),
    (3, JsonDocument(@@{
        "owner_id": 101,
        "tag": "active",
        "archived": true,
        "content": {"x": 2, "y": 1}
    }@@)),
    (4, JsonDocument(@@{
        "owner_id": 102,
        "tag": "pending",
        "archived": false,
        "content": {"x": 2, "y": 2}
    }@@));
```

Прямое сравнение с параметром

Самый распространённый вариант — параметр подставляется как правый операнд сравнения, а левым операндом является вызов `JSON_VALUE` с явным `RETURNING`:

```
DECLARE $owner_id AS Int64;

SELECT id, payload
FROM documents VIEW json_idx
WHERE JSON_VALUE(payload, '$.owner_id' RETURNING Int64) = $owner_id
  AND JSON_EXISTS(payload, '$.archived ? (@ == false)');
```

В индекс попадает токен «путь + значение» (`$.owner_id = $owner_id`) — параметр учитывается как обычное значение. Это позволяет выполнить выборку с такой же селективностью, как и при сравнении с литералом.

Запуск с `$owner_id = 100` вернёт строки `1` и `2`.

Поиск по списку значений

Для поиска по нескольким значениям одного поля удобно использовать `IN`:

```
DECLARE $tags AS List<Utf8>;

SELECT id
FROM documents VIEW json_idx
WHERE JSON_VALUE(payload, '$.tag' RETURNING Utf8) IN $tags;
```

Поведение индекса различается в зависимости от того, литерал это или параметр:

- `IN ("active"u, "pending"u)` — список литералов превращается в `OR` нескольких токенов «путь + значение», по одному на каждое значение списка.
- `IN $tags` — для каждого значения параметра во время выполнения запроса формируется токен «путь + значение» (`$.tag` + значение из списка). На этапе компиляции значения параметра ещё неизвестны, поэтому в плане запроса (`EXPLAIN`) такое условие отображается как пара «путь + параметр» (`{"path": "$.tag", "param": "$tags"}`).

Параметром для `IN` может выступать любая коллекция скалярных значений: `List<T>`, `Tuple<T, ...>`, `Dict<K, V>` или `Set<T>`.

Запуск с `$tags = ["active"u, "pending"u]` вернёт строки `1`, `3` и `4`.

Параметры внутри JsonPath (PASSING)

Если параметр должен использоваться внутри фильтра JsonPath (`? (...)`), его передают в секции `PASSING`:

```
DECLARE $min_stock AS Int64;

SELECT id
FROM documents VIEW json_idx
WHERE JSON_EXISTS(
    payload,
    '$.content ? (@.y > $threshold)'
    PASSING $min_stock AS threshold
);
```

Для поиска по индексу используется токен пути `$.content.y`, а условие `@.y > $threshold` проверяется пост-фильтром.

Аналогично `PASSING` работает в `JSON_VALUE`:

```
DECLARE $v AS Int64;

SELECT id
FROM documents VIEW json_idx
WHERE JSON_VALUE(
    payload,
    '$.content ? (@.y == $val)'
    PASSING $v AS val
    RETURNING Int64
) = 10;
```

Поддерживаемые типы параметров

Для всех трёх способов поддерживаются параметры со следующими типами: `Int8` … `Int64`, `Uint8` … `Uint64`, `Float`, `Double`, `Bytes` (`String`), `Text` (`Utf8`), `Bool`. Опциональные типы (`Optional<T>`) в параметрах не поддерживаются.

Подробнее о типах см. в JSON_VALUE.

Подробнее

- JSON-индекс — быстрый старт — базовый сценарий использования JSON-индекса.
- Передача параметров в предикаты JSON-индекса — детальное описание всех вариантов.
- JsonPath — синтаксис языка JsonPath.

Проверка типа поля и наличия пути

Этот рецепт показывает, как JSON-индекс применяется для проверки структуры документов: наличие конкретного пути, тип значения по пути и т. п. Эти задачи часто встречаются при работе с разнородными JSON-документами, когда часть полей опциональна или может иметь разный тип.

Подготовка

```
CREATE TABLE documents (
    id Uint64,
    payload JsonDocument,
    PRIMARY KEY (id),
    INDEX json_idx GLOBAL USING json ON (payload)
);

UPSERT INTO documents (id, payload) VALUES
    (1, JsonDocument(@@{"archived": false, "value": 1,    "data": [1, 2, 3]}@@)),
    (2, JsonDocument(@@{"archived": false, "value": 2,    "data": "plain text"}@@)),
    (3, JsonDocument(@@{"archived": true,  "value": 3,    "data": {"nested": true}}@@)),
    (4, JsonDocument(@@{"archived": false, "value": null, "meta": "no data field"}@@));
```

Найти документы, у которых поле содержит массив

Метод JsonPath `.type()` возвращает строковое имя типа значения по указанному пути. Это позволяет фильтровать документы по типу содержимого:

```
SELECT id
FROM documents VIEW json_idx
WHERE JSON_VALUE(payload, '$.data.type()' RETURNING Utf8) = "array"u;
```

В индекс попадает токен пути `$.data` — метод `.type()` завершает построение токена пути. Точная проверка строкового значения `"array"` выполняется пост-фильтром.

Результат:

```
id
1
```

Полный список значений, возвращаемых методом `.type()`, приведён в описании синтаксиса JsonPath.

Найти документы, у которых поле является не пустым массивом

Метод `.size()` возвращает количество элементов массива (или 1 для скаляра, 0 для отсутствующего пути). В комбинации с фильтром по типу:

```
SELECT id
FROM documents VIEW json_idx
WHERE JSON_VALUE(payload, '$.data.type()' RETURNING Utf8) = "array"u
  AND JSON_VALUE(payload, '$.data.size()' RETURNING Int64) > 0;
```

Оба фрагмента индексируются по соответствующим путям, а точные значения проверяются пост-фильтром. Результат: `id = 1`.

Найти документы со значением `false` или `null`

Для проверки «значение равно `false`» или «значение равно `null`» используйте JsonPath внутри `JSON_EXISTS`, а не `JSON_VALUE(...) IS NULL`:

```
-- Значение поля 'archived' равно false
SELECT id
FROM documents VIEW json_idx
WHERE JSON_EXISTS(payload, '$.archived ? (@ == false)');

-- Значение поля 'value' равно null
SELECT id
FROM documents VIEW json_idx
WHERE JSON_EXISTS(payload, '$.value ? (@ == null)');
```

В операцию поиска по индексу попадает путь `$.archived` (или `$.value`) и значения `false` или `null`, соответственно.

> **Примечание**
>
> Указанные выше условия отбирают документы, в которых указанный атрибут явно установлен в значение false или null. Проверку отсутствия атрибута нельзя выполнить с помощью JSON-индекса.

Подробнее

- JSON_EXISTS — что допустимо в выражениях JsonPath.
- JSON_VALUE — что допустимо при извлечении значений.
- JsonPath: методы — список методов (`type`, `size`, `keyvalue`, ...) и предикатов JsonPath.

Рецепты: локальные индексы

Практические примеры по локальным индексам.

- Блум-индекс — быстрый старт
- Быстрый старт с min_max-индексом

Блум-индекс — быстрый старт

Колоночная (OLAP) таблица: bloom_filter

Ниже минимальный пример: колоночная таблица с первичным ключом и локальным индексом `bloom_filter` по колонке, которая часто используется в условиях фильтрации.

```
CREATE TABLE events (
    id Uint64 NOT NULL,
    resource_id Utf8 NOT NULL,
    payload String,
    message Utf8,
    PRIMARY KEY (id),
    INDEX idx_res LOCAL USING bloom_filter
        ON (resource_id)
        WITH (false_positive_probability = 0.01)
)
WITH (
    STORE = COLUMN
);
```

Расширение примера: n-граммный индекс

К той же таблице из примера выше можно добавить `bloom_ngram_filter` по строковой колонке (для колоночных таблиц):

```
ALTER TABLE events
  ADD INDEX idx_msg LOCAL USING bloom_ngram_filter
  ON (message)
  WITH (
    ngram_size = 3,
    false_positive_probability = 0.01,
    case_sensitive = true
  );
```

Запросы и эффект

После загрузки данных селективные запросы с условиями по проиндексированным колонкам могут читать меньше данных: при обходе хранилища Блум-индекс пропускает фрагменты, в которых искомое значение гарантированно отсутствует (по сравнению с полным чтением колонки без такого фильтра).

Пример данных и запросов к колоночной таблице из примеров выше:

```
INSERT INTO events (id, resource_id, payload, message) VALUES
    (1, "res-1", "{}", "started"),
    (2, "res-42", "{}", "error: timeout"),
    (3, "res-2", "{}", "done");
```

Фильтр по значению в колонке с `bloom_filter` — движок может отсечь лишние фрагменты при чтении `resource_id` и связанных колонок:

```
SELECT id, message
FROM events
WHERE resource_id = "res-42";
```

Поиск подстроки в колонке с `bloom_ngram_filter` — индекс по n-граммам помогает отбросить фрагменты без подходящих n-грамм в `message`:

```
SELECT id, message
FROM events
WHERE message LIKE '%timeout%';
```

Строковая (OLTP) таблица: префиксный фильтр Блума

В строковой таблице индекс `bloom_filter` строится по левому префиксу первичного ключа. Индексируемые колонки должны образовывать непрерывное ведущее подмножество колонок первичного ключа:

```
CREATE TABLE orders (
    customer_id Utf8 NOT NULL,
    order_id Utf8 NOT NULL,
    amount Decimal(10,2),
    PRIMARY KEY (customer_id, order_id),
    -- Префиксный фильтр Блума по первой колонке ключа
    INDEX idx_customer LOCAL USING bloom_filter
        ON (customer_id)
        WITH (false_positive_probability = 0.001),
    -- Префиксный фильтр Блума по всему первичному ключу
    INDEX idx_full_key LOCAL USING bloom_filter
        ON (customer_id, order_id)
);
```

Точечные чтения и сканы по диапазону, ограничивающие ведущие колонки ключа, могут пропускать нерелевантные фрагменты данных. Запрос с фильтром только по `customer_id` использует префиксный фильтр Блума `idx_customer`:

```
SELECT amount FROM orders WHERE customer_id = "cust-42";
```

Запрос с фильтром по всему первичному ключу использует `idx_full_key`:

```
SELECT amount FROM orders WHERE customer_id = "cust-42" AND order_id = "ord-1001";
```

Как убедиться в эффективности индекса

Чтобы проверить, что Блум-индекс действительно помогает, выполните один и тот же селективный запрос на таблице с достаточным объёмом данных до и после создания индекса и сравните время выполнения.

Дополнительные материалы:

- подробности и ограничения — в статье Блум-индексы;
- настройка параметров — раздел Настройка параметров;
- полный синтаксис — в ALTER TABLE ADD INDEX.

Быстрый старт с min_max-индексом

Создание таблицы с min_max-индексом

Ниже минимальный пример: колоночная таблица с первичным ключом и локальным индексом типа `min_max` на колонках, которые часто используются в фильтрах.

```
CREATE TABLE events (
    id Uint64 NOT NULL,
    created_at Timestamp,
    level Int32,
    resource_id Utf8,
    PRIMARY KEY (id),
    INDEX idx_created_at LOCAL USING min_max
        ON (created_at),
    INDEX idx_level LOCAL USING min_max
        ON (level)
)
WITH (
    STORE = COLUMN
);
```

Запросы и эффект

После загрузки данных селективные запросы с фильтрами по индексируемым колонкам могут читать меньше данных: во время сканирования хранилища min_max-индекс пропускает фрагменты, диапазон значений которых не может содержать подходящие строки.

Пример данных и запросов для таблицы выше:

```
INSERT INTO events (id, created_at, level, resource_id) VALUES
    (1, Timestamp("2024-01-01T00:00:00.000000Z"), 1, "res-1"),
    (2, Timestamp("2024-01-01T00:01:00.000000Z"), 3, "res-42"),
    (3, Timestamp("2024-01-02T00:00:00.000000Z"), 5, "res-2");
```

Диапазонный фильтр по колонке с временной меткой:

```
SELECT id, resource_id
FROM events
WHERE created_at >= Timestamp("2024-01-01T00:00:00.000000Z")
  AND created_at <  Timestamp("2024-01-02T00:00:00.000000Z");
```

Диапазонный фильтр по числовой колонке:

```
SELECT id, resource_id
FROM events
WHERE level BETWEEN 2 AND 4;
```

Как проверить эффективность индекса

Чтобы проверить, что min_max-индекс действительно помогает, выполните один и тот же селективный запрос на таблице с достаточным объёмом данных до и после создания индекса и сравните время выполнения.

Дополнительно:

- Подробности и ограничения: min_max-индекс
- Полный синтаксис: ALTER TABLE ADD INDEX

Рецепты работы с трансфером данных из топика в таблицу

В этом разделе собраны практические примеры настройки трансфера данных из топиков в таблицы.

Содержание:

- Трансфер — быстрый старт
- Трансфер — поставка access-логов NGINX в таблицу

Трансфер — быстрый старт

Эта статья поможет быстро начать работу с трансфером в YDB на простейшем модельном примере.

В статье рассматриваются следующие шаги работы с трансфером:

- создание топика, из которого будет читать трансфер;
- создание таблицы, в которую будут записываться данные;
- создание трансфера;
- заполнение топика данными;
- проверка содержимого таблицы.

Шаг 1. Создание топика

Сначала нужно создать топик в YDB, из которого трансфер будет читать данные. Это можно сделать с помощью SQL-запроса:

```
CREATE TOPIC `transfer_recipe/source_topic`;
```

Топик `transfer_recipe/source_topic` позволяет передавать любые неструктурированные данные.

Шаг 2. Создание таблицы

После создания топика следует создать таблицу, в которую будут поступать данные из топика `source_topic`. Это можно сделать с помощью SQL-запроса:

```
CREATE TABLE `transfer_recipe/target_table` (
  partition Uint32 NOT NULL,
  offset Uint64 NOT NULL,
  data String,
  PRIMARY KEY (partition, offset)
);
```

Таблица `transfer_recipe/target_table` имеет три столбца:

- `partition` — идентификатор партиции топика, из которой получено сообщение;
- `offset` — порядковый номер, идентифицирующий сообщение внутри партиции;
- `data` — тело сообщения.

Шаг 3. Создание трансфера

После создания топика и таблицы нужно добавить трансфер данных, который будет переносить сообщения из топика в таблицу. Это можно сделать с помощью SQL-запроса:

```
$transformation_lambda = ($msg) -> {
    return [
        <|
            partition: $msg._partition,
            offset: $msg._offset,
            data: $msg._data
        |>
    ];
};

CREATE TRANSFER `transfer_recipe/example_transfer`
  FROM `transfer_recipe/source_topic` TO `transfer_recipe/target_table`
  USING $transformation_lambda;
```

В этом примере:

- `$transformation_lambda` — это правило преобразования сообщения из топика в колонки таблицы. В данном случае сообщение из топика переносится в таблицу без изменений. Подробнее о настройке правил преобразования вы можете узнать в документации;
- `$msg` — переменная, содержащая обрабатываемое сообщение из топика.

Шаг 4. Заполнение топика данными

После создания трансфера можно записать в топик сообщения, например, с помощью YDB CLI.

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

```
echo "Message 1" | ydb --profile quickstart topic write 'transfer_recipe/source_topic'
echo "Message 2" | ydb --profile quickstart topic write 'transfer_recipe/source_topic'
echo "Message 3" | ydb --profile quickstart topic write 'transfer_recipe/source_topic'
```

Шаг 5. Проверка содержимого таблицы

После записи сообщении в топик `source_topic` спустя некоторое время появятся записи в таблице `transfer_recipe/target_table`. Проверить их наличие можно с помощью SQL-запроса:

```
SELECT *
```

```
FROM `transfer_recipe/target_table`;
```

Результат выполнения запроса:

partition	offset	data
0	0	Message 1
0	1	Message 2
0	2	Message 3

Строки в таблицу добавляются не для каждого сообщения, полученного из топика, а пакетно с буферизацией. По умолчанию данные записываются в таблицу каждые 60 секунд или при достижении объёма накопленных данных в 8 МБ. Эти параметры можно явно задать при создании трансфера или изменить их позже.

Заключение

Данная статья приводит простой пример работы с трансфером: создание топика, таблицы и трансфера, записи в топик и проверки результата работы трансфера.

Эти примеры призваны проиллюстрировать синтаксис при работе с трансфером. Более реалистичный пример см. в статье описывающей поставку access лога NGINX.

См. также:

- Трансфер данных
- Трансфер — поставка access-логов NGINX в таблицу

Трансфер — поставка access-логов NGINX в таблицу

Эта статья поможет настроить поставку access-логов NGINX в таблицу YDB для дальнейшего анализа. В статье рассматривается формат access-логов NGINX, используемый по умолчанию. Более подробно о формате логов NGINX и его настройке можно прочитать в документации NGINX.

Формат access-лога NGINX по умолчанию имеет вид:

```
$remote_addr - $remote_user [$time_local] "$request" $status $body_bytes_sent "$http_referer" "$http_user_agent"
```

Пример:

```
::1 - - [01/Sep/2025:15:02:47 +0500] "GET /favicon.ico HTTP/1.1" 404 181 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 YaBrowser/25.6.0.0 Safari/537.36"
::1 - - [01/Sep/2025:15:02:51 +0500] "GET / HTTP/1.1" 200 409 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 YaBrowser/25.6.0.0 Safari/537.36"
::1 - - [01/Sep/2025:15:02:51 +0500] "GET /favicon.ico HTTP/1.1" 404 181 "http://localhost/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 YaBrowser/25.6.0.0 Safari/537.36"
```

В статье рассматриваются следующие шаги:

- создание таблицы, в которую будут записываться данные;
- создание трансфера;
- проверка содержимого таблицы.

Пререквизиты

Для выполнения примеров из этой статьи понадобятся:

- Кластер YDB версии 25.2 или выше. Об установке простого одноузлового кластера YDB можно прочитать здесь. Рекомендации по развёртыванию YDB для промышленного использования см. здесь.
- Установленный HTTP-сервер NGINX с ведением access-логов или доступ к access-логам NGINX с другого сервера.
- Настроенная поставка access-логов NGINX из файла в топик `transfer_recipe/access_log_topic`, например, с помощью Kafka Connect с конфигурацией поставки данных из файла в топик.

Шаг 1. Создание таблицы

Добавьте таблицу, в которую будут поставляться данные из топика `transfer_recipe/access_log_topic`. Это можно сделать с помощью SQL-запроса:

```
CREATE TABLE `transfer_recipe/access_log` (
  partition Uint32 NOT NULL,
  offset Uint64 NOT NULL,
  line Uint64 NOT NULL,
  remote_addr String,
  remote_user String,
  time_local Timestamp,
  request_method String,
  request_path String,
  request_protocol String,
  status Uint32,
  body_bytes_sent Uint64,
  http_referer String,
  http_user_agent Utf8,
  PRIMARY KEY (partition, offset, line)
);
```

Эта таблица `transfer_recipe/access_log` имеет три служебных столбца:

- `partition` — идентификатор партиции топика, из которой получено сообщение;
- `offset` — порядковый номер, идентифицирующий сообщение внутри партиции;
- `line` — порядковый номер строки лога внутри сообщения.

Столбцы `partition`, `offset` и `line` однозначно идентифицируют строку файла access-лога.

Если требуется хранить данные access-логов ограниченное время, можно настроить автоматическое удаление старых строк таблицы. Например, для 24 часов это можно сделать с помощью SQL-запроса:

```
ALTER TABLE `transfer_recipe/access_log` SET (TTL = Interval("PT24H") ON time_local);
```

Шаг 2. Создание трансфера

После создания топика и таблицы следует добавить трансфер данных, который будет перекладывать сообщения из топика в таблицу. Это можно сделать с помощью SQL-запроса:

```
$transformation_lambda = ($msg) -> {
    -- Функция преобразования строки лога в строку таблицы
    $line_lambda = ($line) -> {
        -- Сначала разбиваем строку по символу " (двойные кавычки), чтобы выделить строки, которые могут содержать пробел.
        -- Сами строки символ " (двойные кавычки) содержать не могут — он будет заменён последовательностью символов \x22.
        $parts = String::SplitToList($line.1, '"');
        -- Каждую полученную часть, которая не соответствует экранированной строке, разбиваем по пробелу.
        $info_parts = String::SplitToList($parts[0], " ");
```

```
        $request_parts = String::SplitToList($parts[1], " ");
        $response_parts = String::SplitToList($parts[2], " ");
        -- Преобразуем дату в тип Datetime
        $dateParser = DateTime::Parse("%d/%b/%Y:%H:%M:%S");
        $date = $dateParser(SUBSTRING($info_parts[3], 1));

        -- Возвращаем структуру, каждое именованное поле которой соответствует столбцу таблицы.
        -- Важно: типы значений именованных полей должны соответствовать типам столбцов таблицы. Например, ес
ли столбец имеет тип Uint32,
        -- значение именованного поля должно быть типа Uint32. В противном случае требуется явное преобразова
ние с помощью CAST.
        -- Значения колонок NOT NULL должны быть извлечены из опционального типа с помощью функции Unwrap.
        return <|
            partition: $msg._partition,
            offset: $msg._offset,
            line: $line.0,
            remote_addr: $info_parts[0],
            remote_user: $info_parts[2],
            time_local: DateTime::MakeTimestamp($date),
            request_method: $request_parts[0],
            request_path: $request_parts[1],
            request_protocol: $request_parts[2],
            status: CAST($response_parts[1] AS Uint32),
            body_bytes_sent: CAST($response_parts[2] AS Uint64),
            http_referer: $parts[3],
            http_user_agent: CAST(String::CgiUnescape($parts[5]) AS Utf8) -- Явно преобразуем в Utf8, так как
столбец http_user_agent имеет тип Utf8, а не String
        |>;
    };


    $split = String::SplitToList($msg._data, "\n"); -- Если одно сообщение содержит несколько строк лога, раз
деляем его на отдельные строки
    $lines = ListFilter($split, ($line) -> { -- Фильтруем пустые строки, которые, например, могут появиться п
осле последнего символа \n
        return LENGTH($line) > 0;
    });

    -- Преобразуем каждую строку access лога в строку таблицы
    return ListMap(ListEnumerate($lines), $line_lambda);
};

CREATE TRANSFER `transfer_recipe/access_log_transfer`
  FROM `transfer_recipe/access_log_topic` TO `transfer_recipe/access_log`
  USING $transformation_lambda;
```

В этом примере:

- `$transformation_lambda` — это правило преобразования сообщения из топика в колонки таблицы. Каждая строка access лога, записанная в сообщение, обрабатывается отдельно при помощи `line_transformation_lambda`;
- `$line_lambda` — это правило преобразования одной строки access лога в строку таблицы;
- `$msg` — переменная, которая содержит обрабатываемое сообщение из топика.

Шаг 3. Проверка содержимого таблицы

После записи сообщений в топик `transfer_recipe/access_log_topic` спустя некоторое время появятся записи в таблице `transfer_recipe/access_log`. Проверить их наличие можно с помощью SQL-запроса:

```
SELECT *
FROM `transfer_recipe/access_log`;
```

Результат выполнения запроса:

#	partition	offset	line	remote_addr	remote_user	time_local	request_method	request_path	request_protocol	status	bo
1	0	2	0	::1	-	2025-09-01T15:02:51.000000Z	GET	/favicon.ico	HTTP/1.1	404	18
2	0	1	0	::1	-	2025-09-01T15:02:51.000000Z	GET	/	HTTP/1.1	200	40

3	0	0	0	::1	-	2025-09-01T15:02:47.000000Z	GET	/favicon.ico	HTTP/1.1	404	18

Строки в таблицу добавляются не для каждого сообщения, полученного из топика, а пакетно с буферизацией. По умолчанию данные записываются в таблицу каждые 60 секунд или при достижении объёма накопленных данных в 8 МБ. Эти параметры можно явно задать при создании трансфера или изменить их позже.

Заключение

В этой статье показан пример доставки access-логов NGINX в таблицу YDB. Логи любого другого текстового формата можно обрабатывать аналогичным образом: нужно создать таблицу для хранения необходимых данных из лога и правильно написать lambda-функцию, преобразующую строки лога в строки таблицы.

Смотрите также

- Трансфер данных
- Трансфер — быстрый старт
- Lambda-функция
- CREATE TABLE
- CREATE TOPIC
- CREATE TRANSFER
- Функция UNWRAP
- Функция COALESCE

Рецепты работы с потоковыми запросами

Этот раздел содержит рецепты работы с потоковыми запросами.

Содержание:

- Быстрый старт: чтение и запись в топики
- Отладочное чтение из топика

Быстрый старт: чтение и запись в топики

В этом руководстве вы создадите свой первый потоковый запрос.

Запрос будет:

- читать события из входного топика;
- отбирать только ошибки;
- подсчитывать количество ошибок по каждому серверу за 10 минут;
- записывать результат в выходной топик.

События поступают в формате JSON с полями: время, уровень логирования и имя сервера.

Вы выполните следующие шаги:

- создание топиков;
- создание потокового запроса;
- просмотр состояния запроса;
- заполнение входного топика данными;
- проверка содержимого выходного топика;
- удаление потокового запроса.

Предварительные условия

Для выполнения примеров вам потребуется:

- запущенная база YDB — см. quick start;
- включённый фича-флаг `enable_streaming_queries`.

Docker

```
docker run -d --rm --name ydb-local -h localhost \
  --platform linux/amd64 \
  -p 2135:2135 -p 2136:2136 -p 8765:8765 -p 9092:9092 \
  -v $(pwd)/ydb_certs:/ydb_certs \
  -e GRPC_TLS_PORT=2135 -e GRPC_PORT=2136 -e MON_PORT=8765 \
  -e YDB_FEATURE_FLAGS=enable_streaming_queries \
  ydbplatform/local-ydb:25.4
```

local_ydb

```
./local_ydb deploy \
  --ydb-working-dir=/absolute/path/to/working/directory \
  --ydb-binary-path=/path/to/kikimr/driver \
  --enable-feature-flag=enable_streaming_queries
```

Примечание

В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Шаг 1. Создание топиков

Создайте входной и выходной топики:

```
CREATE TOPIC input_topic;
CREATE TOPIC output_topic;
```

Проверьте, что топики созданы:

```
./ydb --profile quickstart scheme ls
```

Шаг 2. Создание потокового запроса

Создайте потоковый запрос с помощью CREATE STREAMING QUERY:

```
CREATE STREAMING QUERY query_example AS
DO BEGIN

$number_errors = SELECT
    Host,
    COUNT(*) AS ErrorCount,
    CAST(HOP_START() AS String) AS Ts  -- Время начала окна, соответствующего результату агрегации
FROM
    input_topic
WITH (
    FORMAT = json_each_row,
    SCHEMA = (
        Time String NOT NULL,
        Level String NOT NULL,
        Host String NOT NULL
    )
)
WHERE
```

```
    Level = "error"
GROUP BY
    HOP(CAST(Time AS Timestamp), "PT600S", "PT600S", "PT0S"),  -- Число ошибок на неперекрывающихся окнах дли
ной 10 минут
    Host;

INSERT INTO
    output_topic
SELECT
    ToBytes(Unwrap(Yson::SerializeJson(Yson::From(TableRow()))))  -- Сериализация всех колонок в JSON
FROM
    $number_errors;

END DO
```

Подробнее:

- Агрегация `GROUP BY HOP` и функция `HOP_START` — GROUP BY ... HOP.
- Запись данных в топик — Форматы при записи данных.
- Сериализация в JSON: TableRow, Yson::From, Yson::SerializeJson, Unwrap, ToBytes.

Шаг 3. Просмотр состояния запроса

Проверьте состояние запроса через системную таблицу streaming_queries:

```
SELECT
    Path,
    Status,
    Issues,
    Run
FROM
    `.sys/streaming_queries`
```

Убедитесь, что в поле `Status` значение `RUNNING`. В противном случае проверьте поле `Issues`.

Если запрос находится в статусе `SUSPENDED` или в поле `Issues` есть ошибки, обратитесь к разделу диагностика ошибок.

Шаг 4. Заполнение входного топика данными

Запишите тестовые сообщения в топик с помощью YDB CLI:

```
echo '{"Time": "2025-01-01T00:00:00.000000Z", "Level": "error", "Host": "host-1"}' | ./ydb --profile quicksta
rt topic write input_topic
echo '{"Time": "2025-01-01T00:04:00.000000Z", "Level": "error", "Host": "host-2"}' | ./ydb --profile quicksta
rt topic write input_topic
echo '{"Time": "2025-01-01T00:08:00.000000Z", "Level": "error", "Host": "host-1"}' | ./ydb --profile quicksta
rt topic write input_topic
echo '{"Time": "2025-01-01T00:12:00.000000Z", "Level": "error", "Host": "host-2"}' | ./ydb --profile quicksta
rt topic write input_topic
echo '{"Time": "2025-01-01T00:12:00.000000Z", "Level": "error", "Host": "host-1"}' | ./ydb --profile quicksta
rt topic write input_topic
```

Результат появится в выходном топике после закрытия 10-минутного окна агрегации.

Шаг 5. Проверка содержимого выходного топика

Прочитайте данные из выходного топика:

```
./ydb --profile quickstart topic read output_topic --partition-ids 0 --start-offset 0 --limit 10 --format new
line-delimited
```

Ожидаемый результат:

```
{"ErrorCount":1,"Host":"host-2","Ts":"2025-01-01T00:00:00Z"}
{"ErrorCount":2,"Host":"host-1","Ts":"2025-01-01T00:00:00Z"}
```

Шаг 6. Удаление запроса

Удалите запрос с помощью DROP STREAMING QUERY:

```
DROP STREAMING QUERY query_example;
```

Что дальше

- Изучите форматы данных, поддерживаемые в потоковых запросах.
- Узнайте, как обогащать данные справочником из локальной таблицы или из S3.
- Научитесь записывать результаты в таблицы.

См. также

- Потоковые запросы;
- Форматы данных при чтении/записи из топиков.

Отладочное чтение из топика

При разработке потоковых запросов бывает полезно быстро посмотреть, какие данные поступают в топик, без создания полноценного потокового запроса. Для этого выполните обычный `SELECT` из топика.

Подробное описание чтения из топика — в статье YQL-запросы к топикам.

> **Важно**
>
> Чтение через `SELECT` предназначено только для отладки. Для промышленного использования создавайте потоковые запросы через CREATE STREAMING QUERY.

См. также

- YQL-запросы к топикам
- Чтение данных из топика
- Форматы данных при чтении/записи из топиков

Диагностика проблем

В данном разделе описана диагностика проблем, которые могут возникнуть с базами данных YDB и приложениями, работающими с этими базами данных.

- Диагностика проблем с производительностью

Диагностика проблем с производительностью

Решение проблем с производительностью баз данных требует комплексного подхода, который включает в себя оптимизацию запросов, правильную конфигурацию аппаратных ресурсов, а также грамотную архитектуру базы данных и использующих её приложений. Регулярный мониторинг и обслуживание базы данных являются необходимыми условиями для диагностики и решения таких проблем.

Инструменты для диагностики проблем с производительностью

Для диагностики проблем с производительностью YDB используются следующие инструменты:

- YDB метрики.

  Диагностика большинства проблем с производительностью включает анализ дашбордов в Grafana, которые используют метрики YDB, собираемые с помощью Prometheus. Описание установки Grafana и Prometheus см. в разделе Настройка мониторинга кластера YDB;
- Логи YDB;
- Трассировка;
- YDB CLI;
- YDB UI;
- План выполнения запроса;
- Параметризованные запросы и повторная компиляция;
- Сторонние инструменты мониторинга.

Классификация проблем с производительностью YDB

Проблемы с производительностью YDB можно разделить на несколько категорий. Данный раздел документации предоставляет общее описание этих категорий, начиная с самого низкого уровня системы и поднимаясь до уровня клиента. Сами инструкции по диагностике проблем см. ниже.

Проблемы с инфраструктурой

- **Сетевые проблемы**. Перегрузка сети внутри датацентра и особенно между датацентрами, может значительно повлиять на производительность YDB.
- **Сбои в работе датацентров**. Нарушения работы датацентров могут привести к недоступности сервиса или данных. Для решения данной проблемы кластер YDB можно настроить на работу в трёх датацентрах или зонах доступности. Однако при этом также необходимо учитывать вопрос производительности.
- **Сервисные работы и учения датацентров**. Плановые сервисные работы или учения для подготовки персонала к потенциальным чрезвычайным ситуациям и сбоям также могут влиять на производительность выполнения запросов. В зависимости от масштаба сервисных работ или сценария учений серверы YDB могут стать недоступными, что повлечёт за собой те же последствия, что и сбой в работе датацентра.
- **Аппаратные неисправности серверов**. Неисправный процессор, модули памяти или сетевые карты значительно влияют на производительность базы данных или приводят к полной недоступности сервера.

Проблемы с нехваткой аппаратных ресурсов

Такие проблемы возникают, когда нагрузка на базу данных требует больше аппаратных ресурсов — таких как процессор, память, дисковое пространство или пропускная способность сети — чем было выделено. В некоторых случаях неоптимальное выделение ресурсов, например неправильная настройка контрольных групп (cgroups) или пулов ресурсов акторной системы, может привести к нехватке аппаратных ресурсов для YDB и увеличить задержки запросов, даже если для сервера баз данных было выделено достаточно аппаратных ресурсов.

- **Недостаточное быстродействие процессора**. Высокая нагрузка на процессор может привести к медленному выполнению запросов и увеличению задержек. В условиях ограниченного ресурса процессора база данных может с трудом справляться со сложными запросами или интенсивным потоком транзакционных запросов.
- **Недостаточное дисковое пространство**. Нехватка места на диске может привести к невозможности сохранения новых данных, когда база переходит в режим только для чтения. Эта проблема может также приводить к замедлению работы, когда система пытается освободить дисковое пространство, активнее приводя данные к более компактному виду в фоне.
- **Недостаточный объём памяти (RAM)**. Обработка запросов требует памяти для временного хранения промежуточных данных. Недостаток свободной памяти может негативно повлиять на производительность базы данных.
- **Недостаточная пропускная способность**. Высокая скорость операций чтения/записи может перегрузить дисковую систему и приводить к увеличению задержек доступа к данным. Когда распределённое хранилище не может читать или записывать данные с достаточной скоростью, запросы к базе данных, требующие доступа к диску, могут замедляться.

Проблемы на уровне операционной системы

- **Расхождение системного времени на серверах**. Увеличение расхождения системного времени на серверах YDB приводит к росту задержек распределённых транзакций. Большая разница во времени приводит к невозможности выполнения распределённых транзакций.

Проблемы на уровне конфигурации YDB

- **Обновления YDB**. Следует учитывать два основных аспекта: рестарт всех узлов YDB в относительно короткий промежуток времени и отличия в поведении разных версий YDB.
- Ошибки конфигурации пулов ресурсов акторной системы.

Проблемы, связанные со схемами таблиц

- **Перегруженные таблетки data shard**. Таблетки data shard, обслуживающие строчные таблицы, могут быть перегружены по нескольким причинам. Такая перегрузка приводит к росту задержек при выполнении запросов, выполняемых этими таблетками.
- **Избыточные разделения и слияния партиций таблиц**. YDB поддерживает автоматическое разделение и слияние таблеток data shard, что позволяет легко адаптироваться к изменениям в рабочих нагрузках. Однако эти операции не являются бесплатными и могут оказать кратковременное негативное влияние на задержки запросов.

Проблемы, связанные с клиентскими приложениями

- **Проблемы с запросами**. Неэффективные запросы к базе данных могут выполняться значительно медленнее.
- **Проблемы использования SDK**. Проблемы, связанные с неоптимальным или неправильным использованием YDB SDK.

Инструкции

При диагностике проблем с производительностью YDB следует рассматривать каждую потенциальную причину проблемы как гипотезу. Систематично проанализируйте список гипотез и проверьте, применимы ли они к вашей ситуации. Каждая статья содержит описание проблемы, шаги по диагностике и рекомендации по решению проблемы, если гипотеза подтвердилась.

Если что-либо поменялось в системе, и время изменения совпадает с временем возникновения проблемы, связанные с изменением потенциальные причины следует рассмотреть в первую очередь. В ином случае рассмотрите возможные причины проблем с производительностью в приведённом ниже порядке, основанном на частотности проблем в крупных кластерах YDB:

1. Перегруженные шарды и ошибки `overloaded`;
2. Разделение и слияние партиций таблиц;
3. Частые переезды таблеток между узлами;
4. Недостаточные аппаратные ресурсы:
    - Недостаточная пропускная способность ввода-вывода;
    - Недостаточное дисковое пространство;
    - Недостаточное быстродействие процессора;
    - Недостаточный объём памяти (RAM);
5. Аппаратные неисправности и Сбои в работе датацентров;
6. Сетевые проблемы;
7. Процедура последовательного перезапуска;
8. Расхождение системного времени между серверами;
9. Инвалидация блокировок транзакций;
10. Сервисные работы и учения датацентров.

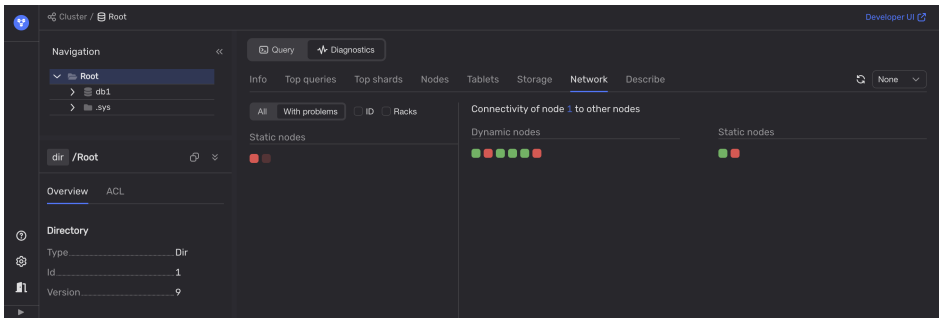
Сетевые проблемы

Сетевые проблемы, такие как ограниченная пропускная способность, потеря пакетов и нестабильное соединение, могут сильно повлиять на производительность базы данных, замедлив скорость обработки запросов или приводя к временным сбоям, например ошибкам превышения срока ожидания (timeout).

Диагностика

Для диагностики сетевых проблем используйте опцию healthcheck во YDB UI:

1. Откройте YDB UI:
   1.1. Перейдите во вкладку **Databases** и выберите необходимую базу данных.
   1.2. На вкладке **Navigation** убедитесь, что выбрана нужная база данных.
   1.3. Откройте вкладку **Diagnostics**.
   1.4. На вкладке **Network** включите фильтр **With problems**.



2. Используйте другие доступные инструменты для мониторинга таких метрик сети как задержки, джиттер, потеря пакетов, пропускная способность и др.

Рекомендации

Обратитесь к ответственным за сетевую инфраструктуру кластера YDB. В больших организациях это может быть своя команда. В других случаях обратитесь в поддержку облачного сервиса или хостинга.

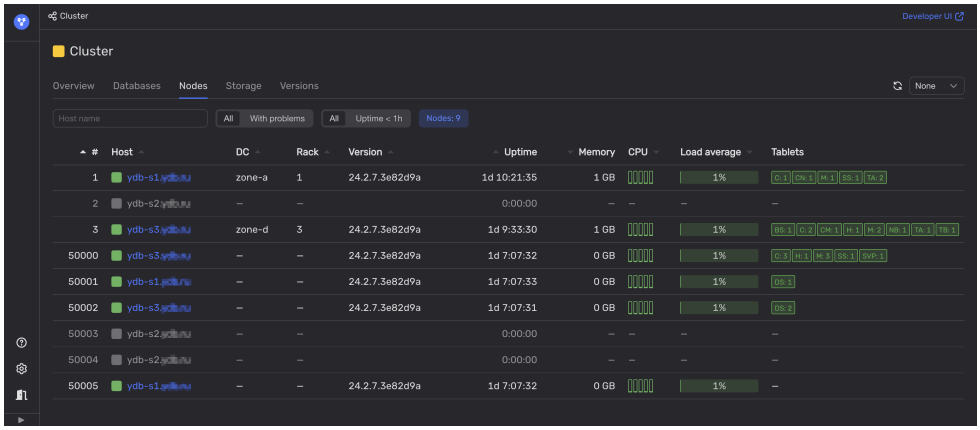
Сбои в работе датацентров

Сбои в работе датацентров могут привести к недоступности сервиса или данных, но YDB разработана специально для решения таких проблем. Разные факторы могут вызвать сбои датацентра — сбои в подаче электроэнергии, стихийные бедствия, кибератаки и пр. Распространённая отказоустойчивая топология кластера YDB включает три датацентра или зоны доступности. В этом случае YDB может продолжать работу, даже если будет отключён один датацентр и одна серверная стойка в другом датацентре. Тем не менее, данная ситуация приведёт к переезду таблеток из отключенного датацентра на рабочие узлы, что временно приводит к увеличению задержек в выполнении запросов.

Диагностика

Чтобы установить недоступность одного из датацентров кластера YDB, выполните следующие шаги:

1. Откройте YDB UI.
2. На вкладке **Nodes** проанализируйте индикаторы состояния в колонках **Host** и **DC**.



   Если все узлы в одном из датацентров недоступны, то в этом датацентре, скорее всего, произошёл сбой.

   В противном случае, посмотрите на колонку **Rack** и проверьте статус узлов YDB в каждой серверной стойке. Эта информация поможет определить недоступность отдельных серверных стоек, что можно классифицировать как частичный сбой в работе датацентра.

Рекомендации

Обратитесь в поддержку. В больших организациях это может быть своя команда, отвечающая за работу низлежащей инфраструктуры. В других случаях обратитесь в поддержку облачного провайдера или хостинга. Также по возможности проверьте статус датацентра, если таковая страница существует.

Важно учитывать возможные сбои в работе датацентров при планировании мощностей, требуемых для работы базы данных. Узлы YDB в каждом датацентре должны иметь достаточно резервных аппаратных ресурсов для нормальной работы при полной нагрузке в случае сбоев в работе датацентров.

Сервисные работы и учения датацентров

Плановые сервисные работы или учения для подготовки персонала к потенциальным чрезвычайным ситуациям и сбоям также могут влиять на производительность выполнения запросов. В зависимости от масштаба сервисных работ или сценария учений серверы YDB могут стать недоступными, что повлечёт за собой те же последствия, что и сбой в работе датацентра.

Диагностика

Проверьте расписание запланированных сервисных работ и учений, не совпадает ли оно с начавшимися проблемами с производительностью. Если не совпадает, см. рекомендации в статье Сбои в работе датацентров.

Рекомендации

Обратитесь к ответственному за выполняемые сервисные работы или учения. Если влияние этих работ на производительность базы данных слишком велико, обсудите возможность их отмены или завершения раньше срока.

Аппаратные неисправности

Вышедшие из строя диски и сетевые карты значительно влияют на производительность базы данных или приводят к полной недоступности сервера. Проблемы с процессором могут приводить к отказам сервера и более высокой нагрузке на остальные узлы YDB.

Диагностика

Используйте инструменты аппаратного мониторинга, предоставляемые операционной системой и датацентром, для диагностики аппаратных неисправностей.

Также используйте опцию **Healthcheck** во YDB UI для диагностики некоторых аппаратных проблем:

- **Проблемы с дисками**
  1. На вкладке **Storage** выберите фильтр **Degraded**, чтобы показать группы хранения или узлы, содержащие повреждённые или вышедшие из строя диски.
  2. Проверьте в Grafana, демонстрируют ли графики на панелях мониторинга **Distributed Storage Overview** и **PDisk Device single disk** признаки деградации производительности системы хранения данных.
- **Сетевые проблемы**

  См. Сетевые проблемы.

Рекомендации

Обратитесь к инженерам, ответственным за аппаратное обеспечение, для решения проблемы. В больших организациях это может быть внутренняя команда. В других случаях обратитесь в поддержку облачного провайдера или хостинга.

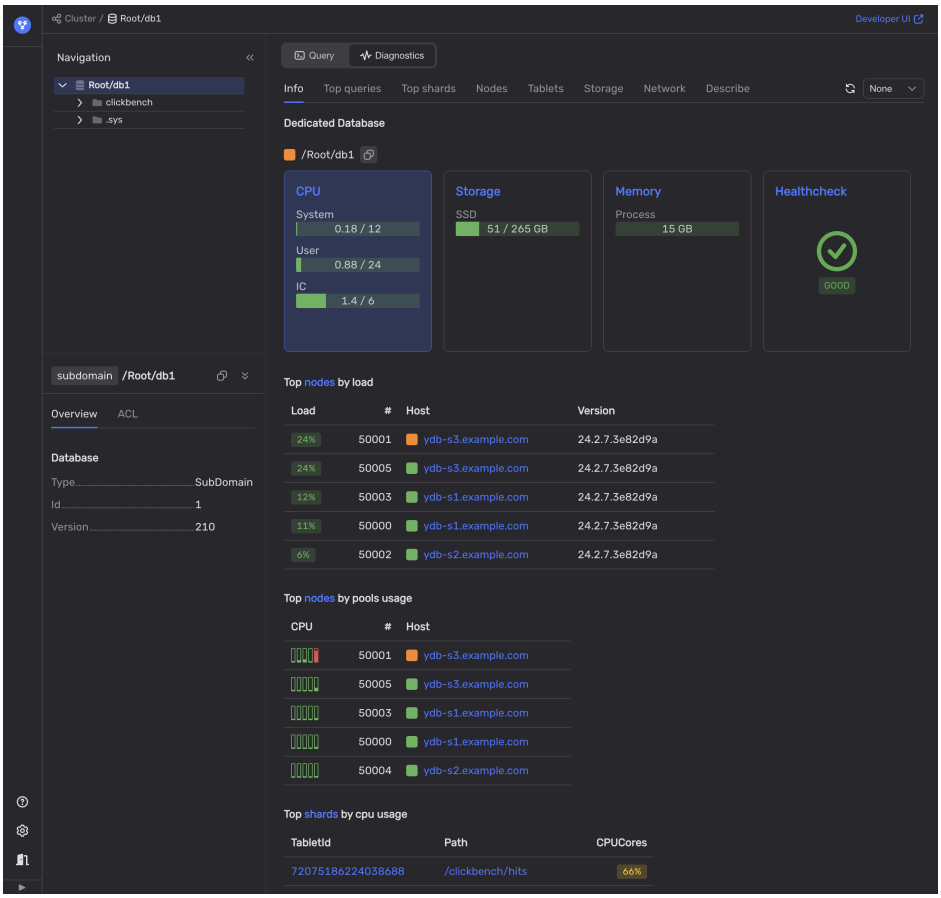
Недостаточное быстродействие процессора

Высокая нагрузка на процессор может привести к медленному выполнению запросов и увеличению задержек. В условиях ограниченного ресурса процессора база данных может с трудом справляться со сложными запросами или высоконагруженными транзакционными сценариями использования.

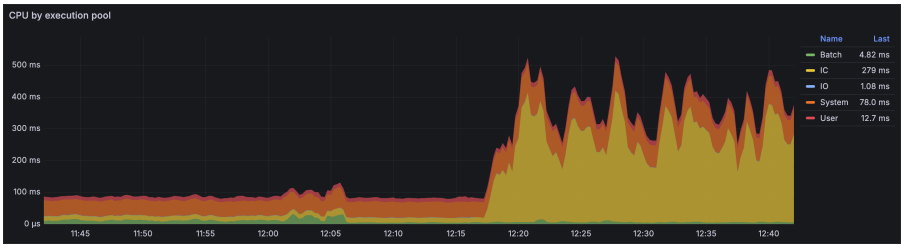
Узлы YDB в основном используют ресурсы процессора на выполнение акторов. На каждом узле акторы выполняются с использованием ресурсов одного из пулов акторной системы. Потребление ресурсов каждого пула измеряется отдельно, что позволяет точнее отслеживать изменения в потреблении ресурсов.

Диагностика

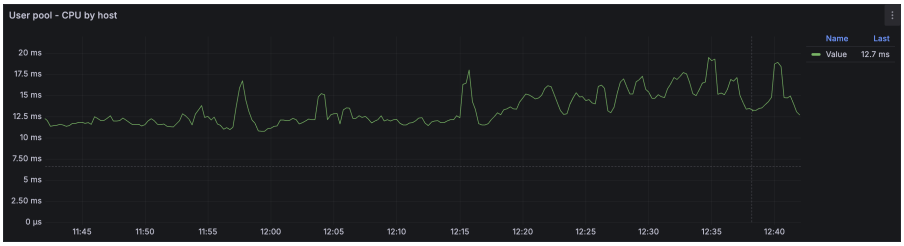
1. Используйте вкладку **Diagnostics** во YDB UI для анализа загрузки процессора во всех пулах ресурсов:
   1.1. Откройте YDB UI, перейдите на вкладку **Databases** и нажмите на требуемую базу данных.
   1.2. На вкладке **Navigation** убедитесь, что требуемая база данных выбрана.
   1.3. Откройте вкладку **Diagnostics**.
   1.4. На вкладке **Info** нажмите на кнопку **CPU** и проверьте уровни загрузки процессора во всех пулах ресурсов.



2. Проанализируйте загрузку процессора во всех пулах ресурсов на графиках Grafana:
   2.1. Откройте панель мониторинга **CPU** в Grafana.
   2.2. Проверьте наличие скачков на следующих графиках:
   - **CPU by execution pool**



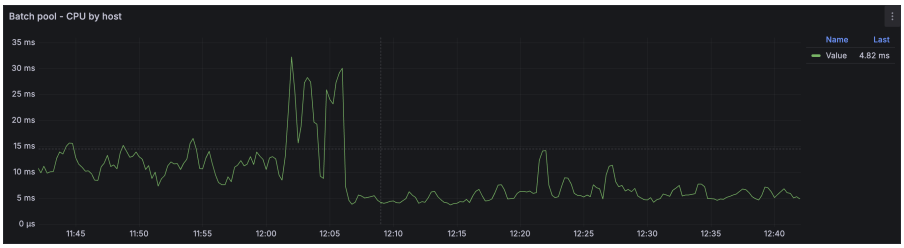
   - **User pool - CPU by host**



   - **System pool - CPU by host**



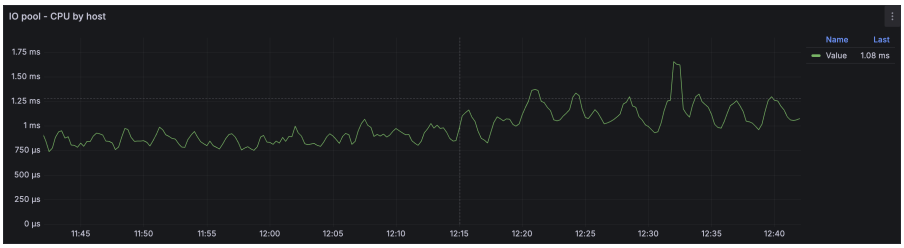
- **Batch pool - CPU by host**



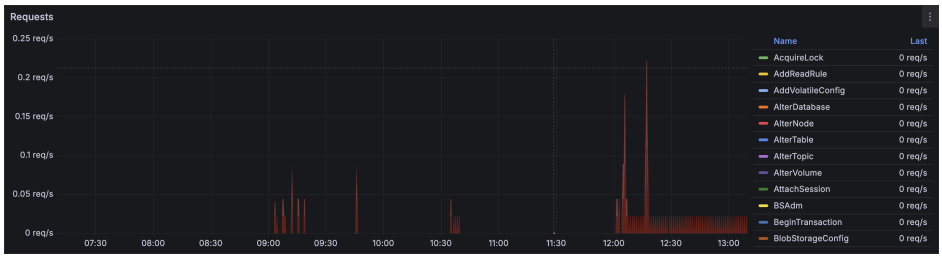
- **IC pool - CPU by host**



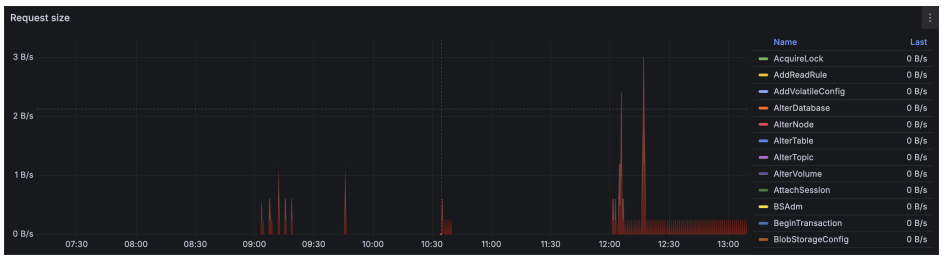
- **IO pool - CPU by host**



3. Если скачки потребления ресурсов процессора обнаружены в пользовательском пуле ресурсов (user pool), проанализируйте изменения пользовательской нагрузки, которые могли бы вызвать недостаток ресурсов процессора. Проверьте следующие графики на панели мониторинга **DB overview** в Grafana:
   - **Requests**



   - **Request size**



   - **Response size**



Также проверьте все графики в секции **Operations** на панели мониторинга **DataShard**.

4. Если скачки потребления ресурсов процессора обнаружены в пакетном пуле ресурсов (batch pool), проверьте, не запущены ли процессы резервного копирования (backups).

Рекомендации

Добавьте дополнительные узлы базы данных в кластер или выделите больше процессорных ядер существующим узлам. Если это невозможно, рассмотрите возможность перераспределения ядер процессора между пулами ресурсов.

Недостаточный объём памяти (RAM)

Если swap (paging of anonymous memory) отключён на сервере YDB, недостаточный объём памяти активирует механизм ядра системы OOM killer, который завершает процессы, потребляющие больше всего оперативной памяти. Часто это сама база данных. Этот механизм также взаимодействует с cgroups, если сконфигурировано несколько cgroups.

Если swap включён, недостаточный объём памяти может приводить к сильной зависимости скорости базы данных от пропускной способности дисковой системы, которая значительно медленнее, чем доступ к данным в памяти.

> **Важно**
>
> Если узлы YDB выполняются на серверах с включённым swap, необходимо отключить swap. YDB – это распределённая система, поэтому если узел перезапускается из-за недостатка памяти, клиент просто переподключится к другому узлу и продолжит работать с данными как будто ничего не случилось. Swap позволяет запросам выполняться на том же узле, но медленнее из-за увеличенной нагрузки на пропускную способность дисковой системы, что обычно ещё хуже.

Хотя причины и механизмы снижения производительности из-за нехватки памяти могут различаться, во всех случаях наблюдаются похожие симптомы: увеличение времени отклика при выполнении запросов и получении данных.

Кроме того, может быть важно, какие компоненты процесса YDB потребляют память.

Диагностика

1. Определите, не перезапускались ли недавно по неизвестной причине какие-либо узлы YDB. Исключите случаи обновления версии YDB и другого планового обслуживания. Это может помочь обнаружить узлы, которые были завершены из-за нехватки памяти и перезапущены системой `systemd`.
   1.1. Откройте YDB UI.
   1.2. На вкладке **Nodes** обратите внимание на узлы с низким значением uptime.
   1.3. Выберите недавно перезапущенный узел и войдите на сервер, на котором он размещён. Выполните команду `dmesg` и проверьте, активировался ли недавно механизм OOM killer.

   Ищите подобные строки:

   ```
   [ 2203.393223] oom-kill:constraint=CONSTRAINT_NONE,nodemask=(null),cpuset=user.slice,mems_allowed=0,
   global_oom,task_memcg=/user.slice/user-1000.slice/session-1.scope,task=ydb,pid=1332,uid=1000
   [ 2203.393263] Out of memory: Killed process 1332 (ydb) total-vm:14219904kB, anon-rss:1771156kB, fil
   e-rss:0kB, shmem-rss:0kB, UID:1000 pgtables:4736kB oom_score_adj:0
   ```

   Кроме того, изучите логи `ydbd` для получения более подробной информации.
2. Определите, используется ли память на 100%.
   2.1. Откройте панель мониторинга **DB overview** в Grafana.
   2.2. Проанализируйте графики в секции **Memory**.
3. Определите, увеличилась ли пользовательская нагрузка на YDB. Проанализируйте следующие графики на панели мониторинга **DB overview** в Grafana:
   - **Requests**;
   - **Request size**;
   - **Response size**.
4. Определите, выходили ли новые версии ваших приложений, работающих с YDB, или менялись ли методы доступа к данным в этих приложениях.

Рекомендации

Рассмотрите следующие решения проблемы нехватки памяти:

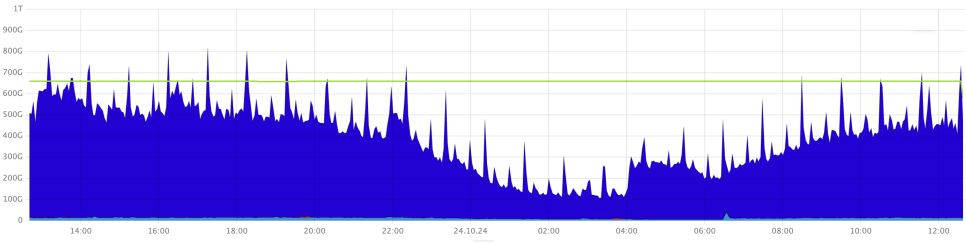
- Если нагрузка на YDB увеличилась из-за новых методов использования или возросшей частоты запросов, попробуйте оптимизировать приложение, чтобы снизить нагрузку на YDB, или добавьте больше узлов YDB.
- Если нагрузка на YDB не изменилась, но узлы всё равно перезапускаются, рассмотрите возможность добавления большего количества узлов YDB или увеличения жёсткого лимита памяти для узлов. Дополнительную информацию об управлении памятью в YDB см. в статье memory_controller_config.

Недостаточная пропускная способность ввода-вывода

Высокая скорость операций чтения/записи может перегрузить дисковую систему и приводить к увеличению задержек доступа к данным. Когда распределённое хранилище не может читать или записывать данные с достаточной скоростью, запросы к базе данных, требующие доступа к диску, могут замедляться.

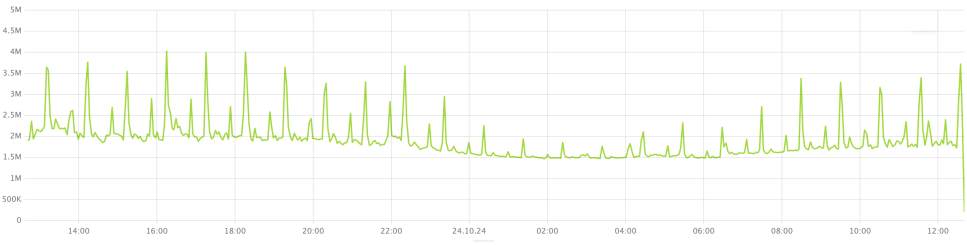
Диагностика

1. Откройте панель мониторинга **Distributed Storage Overview** в Grafana.
2. На графике **DiskTimeAvailable and total Cost relation** проверьте, пересекают ли всплески **Total Cost** уровень **DiskTimeAvailable**.



   Этот график показывает ориентировочную суммарную пропускную способность системы хранения в условных единицах (зелёный) и суммарную стоимость использования в условных единицах (синий). Когда суммарная стоимость использования системы хранения превышает суммарную пропускную способность, система хранения YDB перегружается, и задержки выполнения запросов растут.

3. На графике **Total burst duration** проверьте наличие всплесков в системе хранения. Этот график показывает микровсплески нагрузки на систему хранения, в микросекундах.



> **Примечание**
>
> Этот график может выявить микровсплески нагрузки, которые не проявляются на графике со средней стоимостью использования **Cost and DiskTimeAvailable relation**.

Рекомендации

Добавьте в базу данных дополнительные группы хранения.

В случае с частыми микровсплесками нагрузки может помочь балансировка нагрузки по группам хранения.

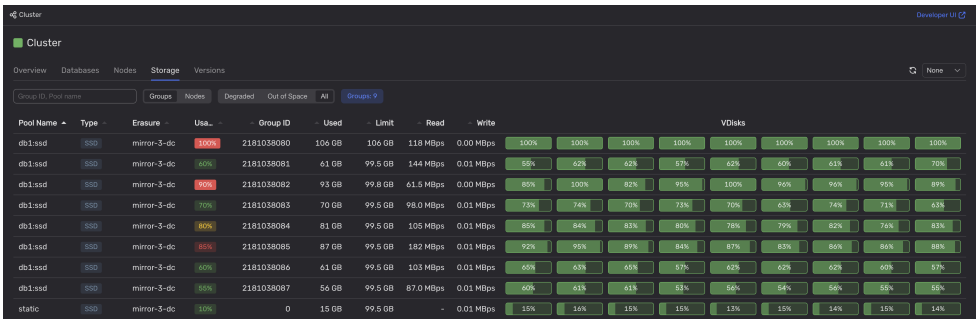
Недостаточное дисковое пространство

Нехватка места на диске может привести к невозможности сохранения новых данных, когда база переходит в режим только для чтения. Эта проблема может также приводить к замедлению работы, когда система пытается освободить дисковое пространство, активнее приводя данные к более компактному виду в фоне.

Диагностика

1. Проверьте наличие скачков на графиках панели мониторинга **DB overview > Storage** в Grafana.
2. Во YDB UI на вкладке **Storage** проанализируйте список доступных групп хранения и их потребление места на диске.

> **Совет**
>
> Используйте фильтр **Out of Space**, чтобы отображать только группы хранения с заполненными дисками.



> **Примечание**
>
> Чтобы получить эту информацию, можно также использовать Healthcheck API.

Рекомендации

Добавьте больше групп хранения в базу данных.

Если у кластера нет свободных групп хранения, необходимо их предварительно сконфигурировать. При необходимости добавьте дополнительные узлы хранения.

Расхождение системного времени между серверами

Синхронизированное время на серверах баз данных имеет важное значение для распределённых баз данных. Если системные часы на серверах YDB будут сильно расходиться, это приведёт к увеличению задержек в распределённых транзакциях.

> **Внимание**
>
> Важно поддерживать системное время на серверах YDB в синхронном состоянии, чтобы избегать больших задержек.

Симптомы и пороги

Основные проявления расхождения времени между узлами:

- **Замедление распределённых транзакций** — время выполнения может увеличиваться примерно на величину расхождения часов между узлами (узел с более «быстрыми» часами ждёт согласования с узлами с отстающими часами).
- **Рост задержек и ошибок по таймаутам** — при рассинхронизации возможны ложные срабатывания дедлайнов и таймаутов запросов.
- **Проблемы с аутентификацией** — при сильном сдвиге времени возможны ошибки из-за несогласованности сроков действия токенов и проверок на стороне сервисов.

При мониторинге расхождения времени между узлами ориентируйтесь на пороги (типичные уровни для наблюдаемости в кластере):

- **Менее 1 миллисекунды** — нормальный уровень для штатной работы кластера.
- **1 — 5 миллисекунд** — зона, в которой уже имеет смысл искать причину рассинхронизации и усиливать наблюдение.
- **Более 5 миллисекунд** — критический уровень: требуется срочная диагностика и выравнивание времени.
- **25 миллисекунд и выше** — нарастающая деградация производительности, пользователи получают массовые задержки и таймауты при выполнении распределённых транзакций.
- **30 секунд** — распределённые транзакции перестают выполняться (см. ниже).

Если системное время узлов, на которых запущены таблетки-координаторы, заметно отличается друг от друга, задержки транзакций увеличиваются на величину разницы во времени между самыми быстрыми и самыми отстающими системными часами. Это происходит потому, что транзакция, запланированная на узле с более быстрыми системными часами, может быть выполнена только после того, как узел с самыми отстающими часами достигнет того же времени.

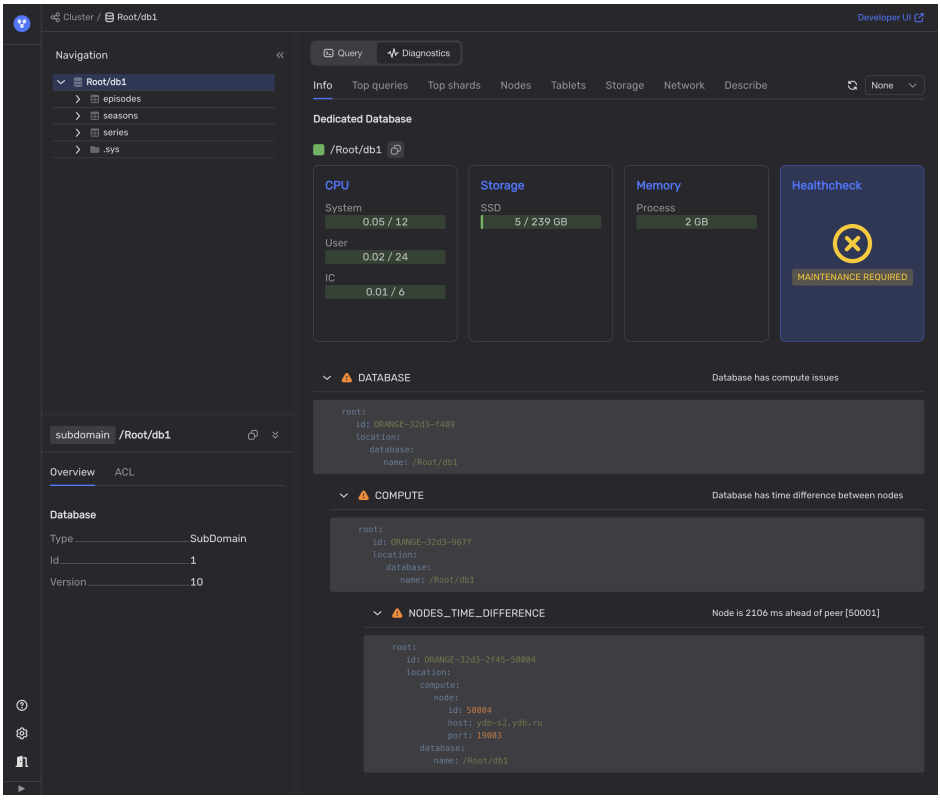
Более того, если отклонение во времени превысит 30 секунд, система YDB откажется обрабатывать распределённые транзакции. Перед тем как координаторы приступят к планированию транзакции, задействованные data shards определяют допустимый диапазон временных меток для транзакции. Начало этого диапазона — текущее системное время таблетки-медиатора, а конец определяет тайм-аут планирования в 30 секунд. Если системное время координатора выходит за пределы этого временного диапазона, он не может запланировать распределённую транзакцию, что приводит к ошибкам в таких запросах.

Диагностика

Чтобы диагностировать расхождение в системном времени серверов YDB, используйте следующие методы:

1. Используйте **Healthcheck** во YDB UI:
   1.1. Во YDB UI перейдите на вкладку **Databases** и нажмите на наименование базы данных.
   1.2. На вкладке **Navigation** убедитесь, что требуемая база данных выбрана.
   1.3. Откройте вкладку **Diagnostics**.
   1.4. На вкладке **Info** нажмите на кнопку **Healthcheck**.

   Если кнопка **Healthcheck** отображает статус `MAINTENANCE REQUIRED`, возможно, в кластере YDB возникли проблемы, например, расхождение системных часов. Все выявленные неполадки будут перечислены в разделе **DATABASE** под кнопкой **Healthcheck**.
   1.5. Чтобы увидеть диагностированные неполадки, раскройте раздел **DATABASE**.



Проблемы с расхождением системного времени отображаются в разделе `NODES_TIME_DIFFERENCE`.

> **Примечание**
>
> Для получения дополнительной информации см. Health Check API

2. Откройте страницу Interconnect overview во YDB UI. По метрикам interconnect (в т.ч. связанным с рассинхронизацией часов между узлами) можно оценить масштаб проблемы вместе с общей картиной задержек и ошибок соединений.

> **Примечание**
>
> Увеличение расхождения системного времени по данным мониторинга интерконнекта (как отображаемым в YDB UI, так и собираемым через метрики кластера) может быть связано как с фактическим расхождением часов, так и с исчерпанием ресурсов в процессорном пуле интерконнекта, либо с перегрузкой сетевого оборудования.

3. Используйте такие инструменты, как `pssh` или `ansible`, чтобы выполнить команду (например, `date +%s%N`) на всех узлах YDB и отобразить значение системных часов.

> **Важно**
>
> На результаты будут влиять сетевые задержки между хостом, на котором запущены `pssh` или `ansible`, и хостами YDB.

Если вы используете утилиты синхронизации времени, вы также можете запросить их статус вместо запроса текущих временных меток. Например:

```
chronyc sources -v
```

Дополнительно полезно смотреть на метрики мониторинга кластера (если они у вас собираются): задержки транзакций, долю успешных gRPC-запросов — на фоне рассинхронизации времени они могут деградировать совместно с показателями interconnect (в частности, метрикой `interconnect.ClockSkewMicrosec`).

Рекомендации

1. Вручную синхронизируйте системные часы серверов, на которых работают узлы YDB. Например, используйте `pssh` или `ansible`, чтобы выполнить команду синхронизации часов на всех узлах.
2. Убедитесь, что системные часы на всех серверах YDB регулярно синхронизируются с помощью `ntpd`, `chrony` или аналогичного инструмента. Используйте один и тот же логический источник времени (одинаковый набор NTP-серверов или иерархию NTP) для всех серверов в кластере YDB, и подключите несколько независимых NTP-источников.

> **Важно**
>
> Использование сервиса **systemd-timesyncd** не рекомендовано, поскольку этот тип клиента NTP не обеспечивает достаточную точность синхронизации системного времени. Вместо него следует использовать другие инструменты, например `chrony` или `ntpd`.

Примеры конфигурации NTP

Ниже — примеры настроек для `chrony`. Список серверов замените на актуальный для вашей среды и политики безопасности.

chrony (`/etc/chrony/chrony.conf`):

```
server ntp1.example.net iburst
server ntp2.example.net iburst
server ntp3.example.net iburst
server ntp4.example.net iburst
maxslewrate 10000
local stratum 10
```

Мониторинг и алертинг

Настройте оповещения по расхождению времени между узлами и по связанным симптомам (в том числе по данным YDB UI и внешнего мониторинга). Ориентиры по порогам см. в разделе «Симптомы и пороги».

Аварийное восстановление

При критическом расхождении времени может потребоваться принудительная синхронизация на узлах (выполняйте осознанно и по согласованной процедуре для вашего кластера):

```
# Пример для chrony: остановка службы и однократная подстройка
sudo systemctl stop chrony
sudo chronyd -q 'server ntp1.example.net iburst'
sudo systemctl start chrony
```

Рекомендуемый порядок действий на кластере:

1. Убедиться, что используются корректные и доступные NTP-серверы.
2. Выровнять время на всех узлах (без рассогласования конфигурации между узлами).
3. Проверить сходимость времени через Healthcheck и мониторинг.
4. Зафиксировать причину сбоя синхронизации (сеть, DNS, блокировки исходящего NTP и т.д.).

Профилактика

- Регулярно проверяйте метрики и отчёты Healthcheck на предмет роста расхождения времени.
- Периодически проверяйте отказоустойчивость цепочки синхронизации (несколько NTP-серверов, доступность порта UDP 123 при необходимости).
- Держите конфигурацию используемого клиента NTP согласованной на всех узлах кластера.

Процедура последовательного перезапуска

Кластеры YDB могут обновляться без простоев, поскольку обычно они содержат избыточные компоненты и поддерживает процедуру последовательного перезапуска. Чтобы обеспечить постоянную доступность данных, YDB включает в себя систему управления кластером (CMS), которая отслеживает все сбои в работе и узлы, отключённые для обслуживания, например при перезапуске. CMS останавливает новые запросы на обслуживание, если они могут поставить под угрозу доступность данных.

Однако, даже если данные всегда доступны, перезапуск всех узлов за относительно короткий промежуток времени может оказать заметное влияние на общую производительность. Каждая таблетка, которая выполнялась на перезапускаемом узле, будет запущена снова на другом узле. Перемещение таблетки между узлами требует времени и может повлиять на задержки запросов, связанных с ней. Смотрите рекомендации по процедуре последовательного перезапуска.

Кроме того, новая версия YDB может обрабатывать запросы по-другому. Хотя производительность, как правило, повышается с каждым обновлением, в некоторых пограничных случаях производительность может снижаться. Смотрите рекомендации по производительности разных версий.

Диагностика

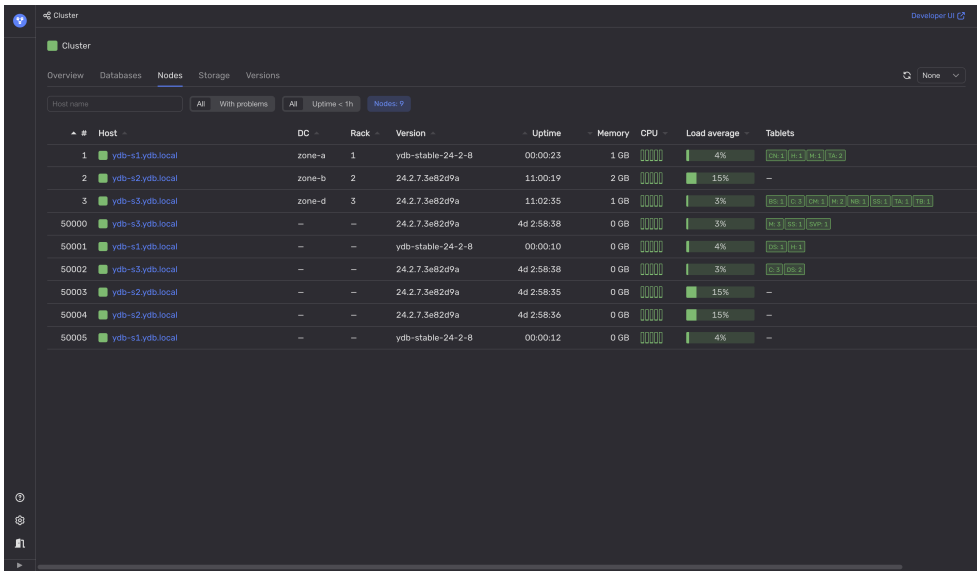
Важно

Диагностика последовательных перезапусков и обновлений YDB основана только на вторичных признаках. Чтобы быть абсолютно уверенным в том, что происходит обновление или последовательный перезапуск узлов, обратитесь к администратору вашей базы данных.

Чтобы проверить, не обновляется ли YDB кластер:

1. Откройте YDB UI.
2. На вкладке **Nodes** проверьте, отличаются ли версии YDB на узлах.

   Также проверьте, имеют ли узлы с новой версией YDB более низкие значения времени с момента запуска (uptime).



Внимание

Низкий uptime узла YDB может также указывать на другие проблемы. Например, см. Недостаточный объём памяти (RAM).

Рекомендации

В случае последовательного перезапуска

Если из-за процесса последовательного перезапуска кластера YDB задержки приложения вырастают выше критического уровня, подумайте о том, чтобы замедлить процесс перезапуска:

1. Если узлы перезапускаются группами, сократите размер групп, вплоть до одного узла за раз.
2. Разнесите во времени перезагрузки в каждом датацентре и/или серверной стойке.
3. Вставьте дополнительные паузы между перезагрузками.

В случае отличий в работе разных версий

Целью является как можно более раннее обнаружение негативного влияния новой версии YDB на скорость выполнения определённых запросов в конкретной пользовательской нагрузке:

1. Просмотрите список изменений YDB сервера, обращая особое внимание на изменения, связанные с производительностью и относящиеся к вашей рабочей нагрузке.
2. Используйте выделенный кластер для тестирования YDB под нагрузкой, которая настолько точно соответствует вашей рабочей нагрузке, насколько это возможно. Всегда сначала развёртывайте новую версию YDB на таких кластерах, чтобы оценить влияние новой версии в тестовом окружении. Отслеживайте задержки как на стороне клиента, так и на стороне сервера, чтобы выявить любые потенциальные проблемы с производительностью.
3. Используйте канареечное развертывание (canary deployment), обновив сначала только один узел кластера и проанализировав любые изменения в его поведении. Если всё выглядит стабильно, постепенно распространите обновление на большее количество узлов, например, на всю серверную стойку или датацентр, и повторяйте проверки на наличие аномалий. Если возникнут какие-либо проблемы, немедленно вернитесь к предыдущей версии и попытайтесь воспроизвести проблему в тестовой среде.

Сообщайте о любых выявленных проблемах с производительностью на GitHub YDB. Предоставьте контекст и все детали, которые могут помочь воспроизвести проблему.

Частые переезды таблеток между узлами

YDB автоматически распределяет нагрузку, перемещая таблетки с перегруженных узлов на другие узлы. Этот процесс управляется компонентом Hive. Когда Hive перемещает таблетки, запросы, затрагивающие эти таблетки, могут выполняться дольше из-за ожидания инициализации таблетки на новом узле.

Для балансировки нагрузки между узлами YDB учитывает использование следующих аппаратных ресурсов:

- процессор;
- оперативная память;
- сетевой трафик;
- Counter[14].

Балансировка выполняется автоматически в следующих случаях:

- **Дисбаланс потребления аппаратных ресурсов**

  Для оценки сбалансированности потребления аппаратных ресурсов YDB использует метрику **Scatter**. Подробнее о логике расчёта метрики **Scatter** и условиях запуска балансировки см. в разделе Дисбаланс потребления ресурсов.

- **Перегруженность узла**

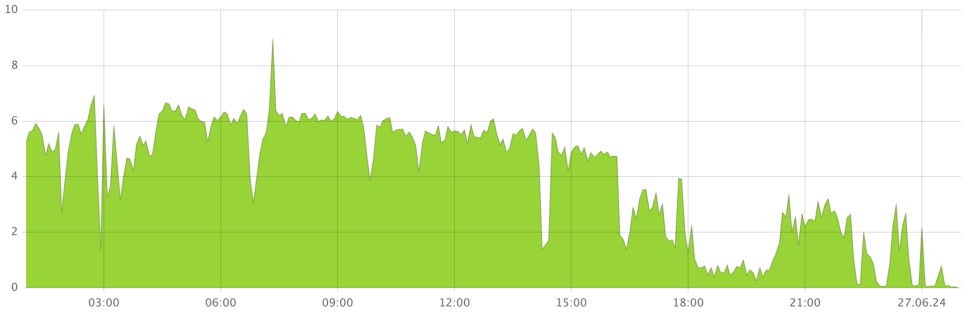
  Hive инициирует балансировку при сильной асимметрии загрузки (например, > 90% на одном узле и < 70% на другом). Подробнее см. Перегруженность узла.

- **Неравномерное распределение таблеток одного объекта**

  Для таблеток, у которых отсутствует явное потребление ресурсов, Hive использует фиктивный ресурс **Counter** для контроля равномерности распределения. При нарушении равномерности запускается балансировка. Подробнее см. Равномерное распределение конкретного объекта.

Диагностика

1. Посмотрите, отображаются ли всплески на графике **Tablets moved by Hive** на панели мониторинга Grafana **DB status**.



   На этой диаграмме отображаются данные о количестве таблеток, перемещаемых между узлами YDB в секунду.

2. Проанализируйте статистику балансировщика Hive.
   2.1. Откройте YDB UI.
   2.2. Нажмите на ссылку **Developer UI** в правом верхнем углу YDB UI.
   2.3. В **Developer UI** перейдите на страницу **Tablets > Hive > App**.

   Проанализируйте статистику балансировщика в правом верхнем углу.

Balancer	Runs	Moves	Last run	Last moves	Progress
Counter	0	0			
CPU	14	14	2024-10-18T12:05:44.342466Z	1	
Memory	0	0			
Network	0	0			
Emergency	0	0			
Spread	0	0			
Scatter	0	0			
Manual	0	0			
Storage	0	0			

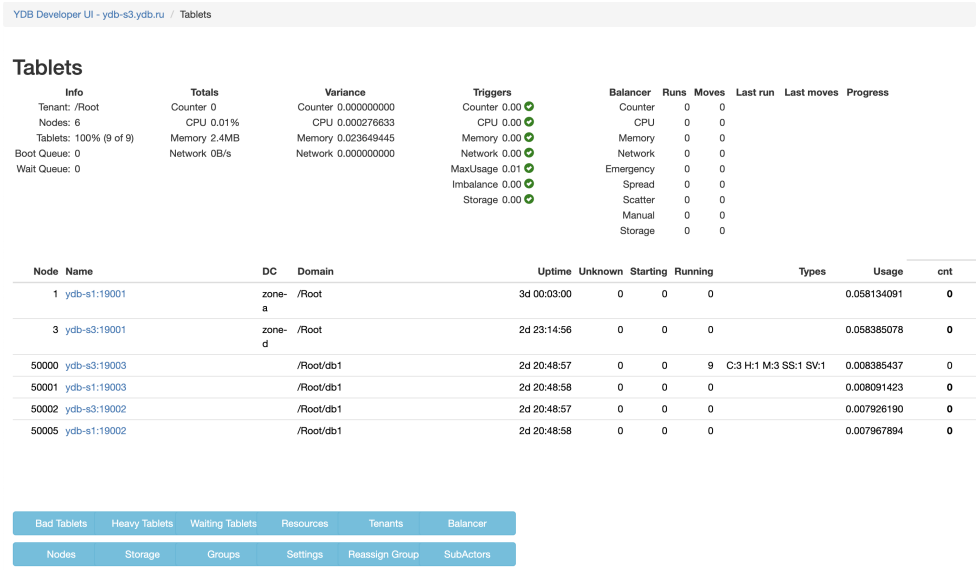
   2.4. Кроме того, чтобы просмотреть недавно перемещённые таблетки, нажмите на кнопку **Balancer**.

   Откроется окно **Balancer**, в котором список недавно перемещённых таблеток отображается в секции **Latest tablet moves**.

Рекомендации

Измените настройки балансировщика Hive:

1. Откройте YDB UI.
2. Нажмите на ссылку **Developer UI** в правом верхнем углу YDB UI.
3. В **Developer UI** перейдите на страницу **Tablets > Hive > App**.



4. Нажмите на **Settings**.
5. Чтобы снизить вероятность слишком частой балансировки, увеличьте следующие пороговые значения балансировщика Hive:

Параметр	Описание	Значение по умолчанию
MinCounterScatterToBalance	Пороговое значение для значения разброса ресурса counter. При достижении этого значения Hive начинает балансировать нагрузку	0.02
MinCPUScatterToBalance	Пороговое значение для значения разброса ресурса процессора. При достижении этого значения Hive начинает балансировать нагрузку	0.5
MinMemoryScatterToBalance	Пороговое значение для значения разброса ресурса оперативной памяти. При достижении этого значения Hive начинает балансировать нагрузку	0.5
MinNetworkScatterToBalance	Пороговое значение для значения разброса ресурса сетевого трафика. При достижении этого значения Hive начинает балансировать нагрузку	0.5
MaxNodeUsageToKick	Пороговое значение использования ресурсов узла YDB. При достижении этого значения Hive начинает срочную балансировку	0.9
ObjectImbalanceToBalance	Пороговое значение метрики дисбаланса объекта базы данных	0.02

Примечание

В этих параметрах используются относительные значения, где 1.0 соответствует 100% и фактически отключает балансировку. Если общее значение аппаратных ресурсов может превышать 100%, скорректируйте значение настройки соответствующим образом.

[14] Counter - это фиктивный ресурс, который представляет собой количество таблеток одного типа, запущенных на узле, который может быть использован для равномерного распределения таблеток между узлами.

Перегруженные таблетки data shard

Таблетки data shard, обслуживающие строковые таблицы, могут быть перегружены по следующим причинам:

- Таблица создана без указания параметра AUTO_PARTITIONING_BY_LOAD.

  В этом случае YDB не разбивает перегруженные таблетки data shard.

  Таблетки data shard являются однопоточными и обрабатывают запросы последовательно. Каждая таблетка data shard может принимать на выполнение до 10000 операций. Принятые запросы ожидают своей очереди на выполнение. Таким образом, чем длиннее очередь, тем выше задержка.

  Если таблетка data shard уже содержит 10000 операций в своей очереди, новые запросы будут возвращать ошибку «overloaded». Повторите такие запросы, используя экспоненциально растущий перерыв, см. Ошибки «overloaded».
- Таблица создана с параметром AUTO_PARTITIONING_MAX_PARTITIONS_COUNT и уже достигла лимита на число партиций.
- Неэффективный первичный ключ, который вызывает дисбаланс в распределении запросов по таблеткам data shard. Типичным примером является использование монотонно увеличивающегося первичного ключа при загрузке данных, что может привести к перегрузке «последней» партиции. Например, это может произойти с автоматически увеличивающимся первичным ключом, использующим тип данных Serial.

Диагностика

1. Используйте YDB UI или Grafana, чтобы проверить, не перегружены ли узлы YDB:
   - На панели мониторинга Grafana **DB overview** проанализируйте диаграмму **Overloaded shard count**.

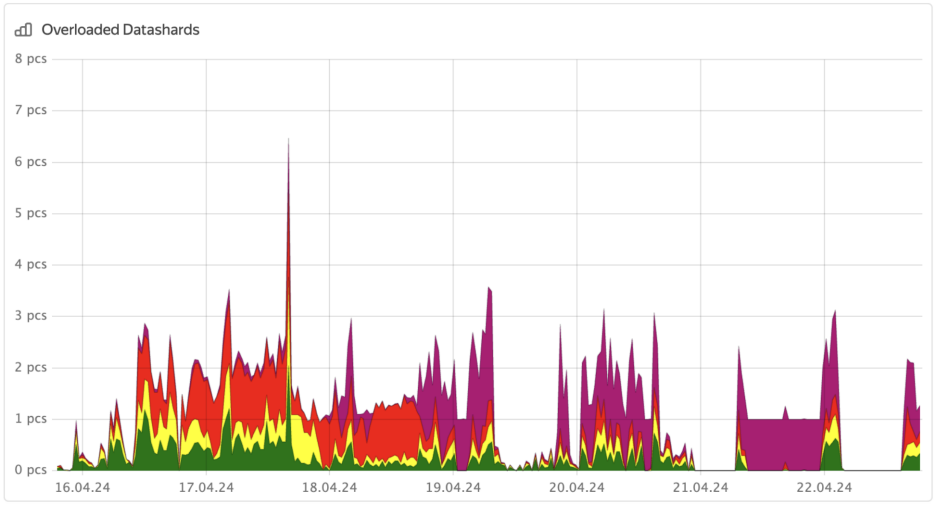
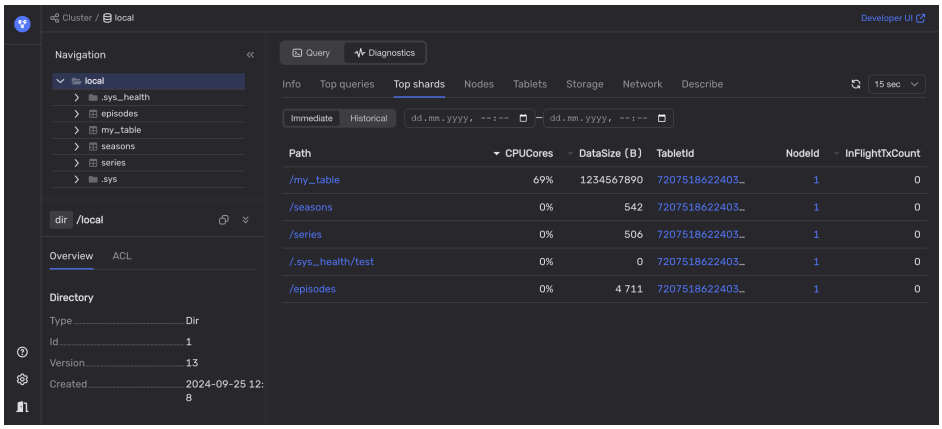


     Диаграмма отображает перегруженные таблетки data shard в кластере YDB, но она не показывает таблицы, в которых есть перегруженные таблетки.

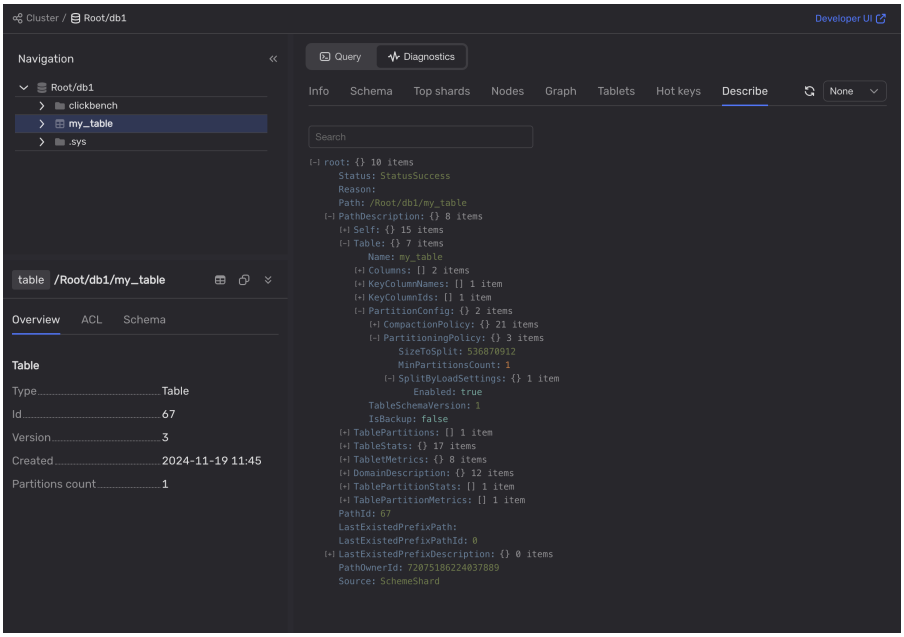
     > **Совет**
     >
     > Настройте уведомления в Grafana о перегрузках таблеток data shard.

   - Во YDB UI:
     1. Перейдите на вкладку **Databases** и выберите базу данных.
     2. На вкладке **Navigation** убедитесь, что база данных выбрана.
     3. Откройте вкладку **Diagnostics**.
     4. Откройте вкладку **Top shards**.
     5. На вкладках **Immediate** и **Historical** отсортируйте таблетки по колонке **CPUCores** и проанализируйте информацию.



     Кроме того, информация о перегруженных таблетках представлена в виде системной таблицы. Дополнительные сведения см. в разделе История перегруженных партиций.
2. Чтобы точно определить проблему со схемой, используйте YDB UI или YDB CLI:
   - Во YDB UI:
     1. На вкладке **Databases** нажмите на базу данных.

2. На вкладке **Navigation** выберите требуемую базу данных.
3. Откройте вкладку **Diagnostics**.
4. На вкладке **Describe** перейдите на страницу `root > PathDescription > Table > PartitionConfig > PartitioningPolicy`.



5. Проанализируйте значения **PartitioningPolicy**:
   - `SizeToSplit`
   - `SplitByLoadSettings`
   - `MaxPartitionsCount`

   Если в таблице не отображаются вышеперечисленные параметры, см. Рекомендации по конфигурации таблиц.

> **Примечание**
>
> Эта информация также отображается на вкладке **Diagnostics > Info**.

- В YDB CLI:
  1. Чтобы получить информацию о проблемной таблице, выполните следующую команду:

     ```
     ydb scheme describe <table_name>
     ```

  2. В выводе команды проанализируйте **Auto partitioning settings**:
     - `Partitioning by size`
     - `Partitioning by load`
     - `Max partitions count`

     Если в таблице не указаны эти параметры, см. Рекомендации по конфигурации таблиц.

3. Проанализируйте, монотонно ли увеличиваются значения первичного ключа:
   - Проверьте тип данных столбца первичного ключа. Типы данных `Serial` используются для автоматического увеличения значений.
   - Проверьте логику приложения.
   - Вычислите разницу между минимальным и максимальным значениями столбца первичного ключа. Затем сравните это значение с количеством строк в данной таблице. Если эти значения совпадают, возможно первичный ключ в этой таблице увеличивается монотонно.

   Если значения первичного ключа действительно увеличиваются монотонно, см. Рекомендации для несбалансированного первичного ключа.

Рекомендации

Для конфигурации таблиц

Рассмотрите следующие решения для устранения перегрузки таблеток data shard:

- Если в проблемной таблице не включено партиционирование по нагрузке, включите его.

> **Совет**
>
> Партиционирование по нагрузке отключено, если на вкладке **Diagnostics > Info** во **YDB UI** или в выводе команды `ydb scheme describe` отображается строка `Partitioning by load: false`.

- Если количество партиций в таблице достигло максимального лимита, увеличьте максимальный лимит партиций таблицы.

Совет

Чтобы определить количество партиций в таблице, см. значение `PartCount` на вкладке **Diagnostics > Info** во **YDB UI**.

Обе операции можно выполнить с помощью запроса `ALTER TABLE ... SET`.

Для несбалансированного первичного ключа

Рассмотрите возможность изменения первичного ключа, чтобы равномерно распределить нагрузку по партициям таблицы. Вы не можете изменить первичный ключ существующей таблицы. Для этого вам нужно будет создать новую таблицу с изменённым первичным ключом, а затем перенести данные в новую таблицу.

Примечание

Также рассмотрите возможность изменения логики вашего приложения для генерации значений первичного ключа для новых строк. Например, используйте хэши значений вместо самих значений.

Избыточные разделения и слияния партиций таблиц

> **Важно**
>
> Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Каждая партиция строковой таблицы в YDB обрабатывается таблеткой data shard. YDB поддерживает автоматическое разделение и слияние таблеток data shard, что позволяет легко адаптироваться к изменениям в рабочих нагрузках. Однако эти операции не являются бесплатными и могут оказать кратковременное негативное влияние на задержки запросов.

Когда YDB разбивает партицию, исходная партиция заменяется двумя новыми партициями, охватывающими тот же диапазон первичных ключей. Теперь две таблетки data shard обрабатывают диапазон первичных ключей, который ранее обрабатывался одной таблеткой data shard, тем самым добавляя больше вычислительных ресурсов для таблицы.

По умолчанию YDB разделяет партицию таблицы, когда её размер достигает 2 ГБ. Однако рекомендуется также включить разделение по загрузке, что позволит YDB разделять перегруженные партиции, даже если их размер меньше 2 ГБ.

У scheme shard уходит примерно 15 секунд на принятие решения о разделении таблетки data shard. По умолчанию пороговое значение потребления процессора для разделения таблетки data shard установлено в 50%.

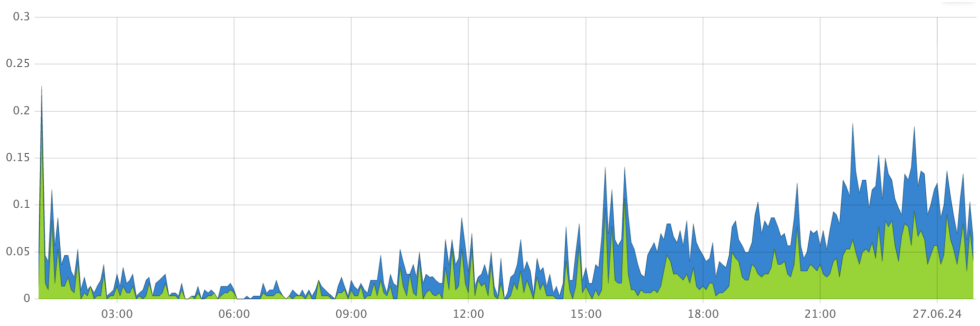
Когда YDB объединяет соседние партиции в строковой таблице, они заменяются одной партицией, которая охватывает их диапазон первичных ключей. Соответствующие таблетки data shard также объединяются в одну таблетку для управления новой партицией.

Для того чтобы произошло слияние, таблетки data shard должны существовать не менее 10 минут, а их загрузка процессора за последний час не должна превышать 35%.

При настройке партиционирования таблицы вы можете также установить лимиты на минимальное и максимальное количество партиций. Если разница между минимальным и максимальным пределами превышает 20%, а загрузка таблицы значительно меняется с течением времени, Hive может начать разделять перегруженные таблицы, а затем объединять их обратно в периоды низкой загрузки.

Диагностика

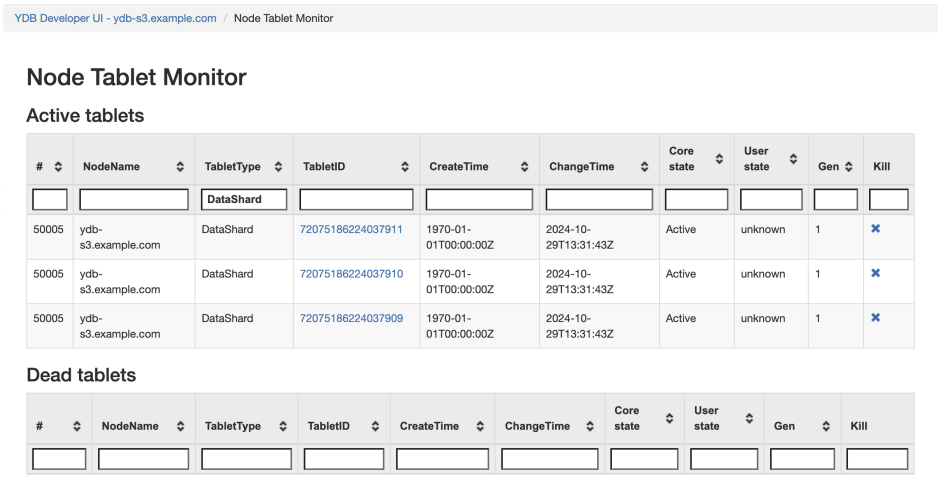
1. Посмотрите, есть ли всплески на графике **Split / Merge partitions** на панели мониторинга Grafana **DB status**.



   На этой диаграмме отображаются временные ряды следующих данных:

   - количество разделений партиций таблиц в секунду (синий);
   - количество слияний партиций таблиц в секунду (зелёный).

2. Проверьте, не увеличилась ли пользовательская нагрузка, когда был замечен всплеск количества разделений и слияний таблеток.
   - Просмотрите диаграммы на панели мониторинга **DataShard** в Grafana на предмет любых изменений в объёме данных, считываемых или записываемых запросами.
   - Изучите диаграмму **Requests** на панели мониторинга **Query engine** в Grafana на предмет увеличения количества запросов.
3. Чтобы определить недавно разделённые или слитые таблетки, выполните следующие шаги:
   3.1. Во YDB UI нажмите на ссылку **Developer UI** в правом верхнем углу.
   3.2. Перейдите на страницу **Node Table Monitor** > **All tablets of the cluster**.
   3.3. Для отображения только таблеток data shard в фильтре **TabletType** укажите `DataShard`.



3.4. Отсортируйте таблетки по колонке **ChangeTime**. Обратите внимание на таблетки, значения времени изменения которых совпадают с пиковыми значениями на диаграмме **Split / Merge partitions**.

3.5. Чтобы определить таблицу, связанную с таблеткой data shard, в строке data shard нажмите на ссылку в столбце **TabletID**.

3.6. На странице **Tablets** откройте ссылку **App**.

Информация о таблице отображается в секции **User table <имя таблицы>**.

4. Чтобы определить, связана ли проблема с неправильной схемой таблицы, выполните следующие шаги:

4.1. Получите информацию о проблемной таблице с помощью YDB CLI. Выполните следующую команду:

```
ydb scheme describe <имя_таблицы>
```

4.2. В выводе команды проанализируйте **Auto partitioning settings**:

- `Partitioning by load`
- `Max partitions count`
- `Min partitions count`

Рекомендации

Если пользовательская нагрузка на YDB не изменилась, рассмотрите возможность изменения интервала между минимальным и максимальным лимитами на количество партиций таблицы до рекомендуемой разницы в 20%. Используйте инструкцию YQL `ALTER TABLE имя_таблицы SET (ключ = значение)` для обновления параметров `AUTO_PARTITIONING_MIN_PARTITIONS_COUNT` и `AUTO_PARTITIONING_MAX_PARTITIONS_COUNT`.

Если вы хотите избежать разделения и слияния таблеток data shard, вы можете выставить одинаковые значения для параметров `AUTO_PARTITIONING_MIN_PARTITIONS_COUNT` и `AUTO_PARTITIONING_MAX_PARTITIONS_COUNT` или отключить разделение по загрузке.

При снижении значения `AUTO_PARTITIONING_PARTITION_SIZE_MB` ниже текущего размера многих партиций это может поставить в очередь на разделение сразу множество партиций и кратковременно увеличить задержки. Операции из очереди выполняются с ограниченным параллелизмом. Снижайте порог постепенно и контролируйте метрики.

Инвалидация блокировок транзакций

Инвалидация блокировок транзакций (TLI) происходит, когда одна транзакция (нарушитель) записывает данные и ломает оптимистичные блокировки другой транзакции (жертвы). Жертва обнаруживает это при коммите и получает ошибку `transaction locks invalidated`. Транзакцию-жертву необходимо повторить. Частые повторы замедляют приложение.

> **Примечание**
>
> YDB SDK предоставляет встроенный механизм повтора временных ошибок. Подробнее см. Обработка ошибок.

Предотвращение конфликтов

- **Сокращайте длительность транзакций.** Чем дольше транзакция удерживает блокировки, тем выше вероятность конфликта. По возможности избегайте интерактивных транзакций: лучший подход — один YQL-запрос с `BEGIN;` и `COMMIT;` для чтения, изменения и коммита данных. Если интерактивные транзакции необходимы, выполняйте `COMMIT` в последнем запросе.
- **Сокращайте пересечение данных между транзакциями.** Чем меньше строк читает транзакция, тем меньше блокировок она удерживает и тем ниже вероятность конфликта. Избегайте чтения заведомо ненужных данных. Если несколько транзакций конкурируют за одни и те же строки — пересмотрите схему данных.
- **Используйте режимы транзакций только на чтение.** Транзакции в режиме `Snapshot Read-Only` читают данные из консистентного снапшота и не устанавливают оптимистичных блокировок — такая транзакция никогда не станет жертвой TLI. Если транзакция не изменяет данные, явно задавайте этот режим через SDK.

Пример: конфликт транзакций при резервировании товара

В интернет-магазине на распродаже сотни покупателей одновременно покупают один товар. Транзакция покупки читает остаток, потом приложение рассчитывает скидки, ждёт подтверждения оплаты, и только после этого списывает:

```
SELECT available FROM stock WHERE sku = $sku AND warehouse_id = $warehouse_id;
-- приложение проверяет: available > 0
-- приложение рассчитывает скидки, ожидает оплаты ...
UPDATE stock SET available = available - 1 WHERE sku = $sku AND warehouse_id = $warehouse_id;
COMMIT;
```

`SELECT` устанавливает оптимистичную блокировку на строку. Пока транзакция ожидает оплаты, другая транзакция успевает изменить ту же строку и закоммититься — блокировка ломается, и первая транзакция получает TLI при коммите.

Решение — сразу резервировать товар одной короткой транзакцией, а всё медленное (скидки, оплата, внешние сервисы) выполнять после:

```
BEGIN;
UPDATE stock SET available = available - 1, reserved = reserved + 1
WHERE sku = $sku AND warehouse_id = $warehouse_id AND available > 0;

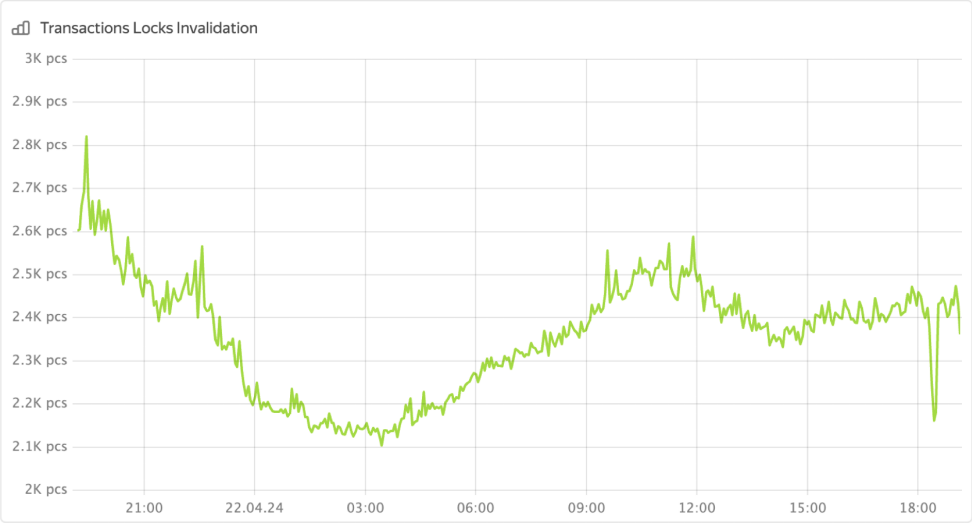
UPSERT INTO orders (order_id, sku, qty, status)
VALUES ($order_id, $sku, 1, "RESERVED");
COMMIT;
-- после: расчёт скидок, оплата, доставка
```

Если оплата прошла — отдельной транзакцией ставим `status = "PAID"`. Если нет — возвращаем товар из `reserved` обратно в `available` и ставим `"CANCELLED"`. Транзакция резерва выполняется быстро и минимизирует время удержания блокировки.

Диагностика

Мониторинг в Grafana

1. Откройте панель мониторинга **DB overview** в Grafana.
2. Проверьте, есть ли всплески количества ошибок на диаграмме **Transaction Lock Invalidation**.



Эта диаграмма отображает количество запросов в секунду, возвращаемых с ошибкой «transaction locks invalidated».

Диагностика через логи TLI

Когда транзакция завершается с ошибкой TLI, сообщение об ошибке содержит идентификатор запроса-жертвы:

```
Transaction locks invalidated. ... VictimQuerySpanId: 1111111111111111.
```

По этому `VictimQuerySpanId` можно найти в логах сервера полный контекст конфликта: какой запрос установил блокировки и какой их сломал. Подробнее о включении логирования, формате записей и корреляции событий, а также об утилите `find_tli_chain` для автоматического анализа логов см. в Логирование TLI.

Анализ через системные представления

Для анализа конфликтов блокировок доступны следующие системные представления:

Анализ на уровне запросов

Для выявления запросов с наибольшим числом конфликтов используйте системное представление `.sys/query_metrics_one_minute`:

```
SELECT QueryText, LocksBrokenAsBreaker, LocksBrokenAsVictim
FROM `.sys/query_metrics_one_minute`
WHERE LocksBrokenAsBreaker > 0 OR LocksBrokenAsVictim > 0
ORDER BY LocksBrokenAsBreaker + LocksBrokenAsVictim DESC;
```

Колонка	Описание
`LocksBrokenAsBreaker`	Сколько раз этот запрос сломал чужие блокировки
`LocksBrokenAsVictim`	Сколько раз блокировки этого запроса были сняты

Запросы с высоким `LocksBrokenAsBreaker` — нарушители: именно они вызывают откаты других транзакций. Запросы с высоким `LocksBrokenAsVictim` — жертвы.

Анализ на уровне партиций

Для анализа сломанных блокировок на уровне партиций таблиц используйте следующие системные представления:

- `.sys/partition_stats` — текущая статистика по партициям, содержит кумулятивное поле `LocksBroken`
- `.sys/top_partitions_by_tli_one_minute` — топ-10 партиций с ненулевым числом сломанных блокировок за минутный интервал
- `.sys/top_partitions_by_tli_one_hour` — топ-10 партиций с ненулевым числом сломанных блокировок за часовой интервал

Пример запроса для поиска партиций с наибольшим числом сломанных блокировок:

```
SELECT
    Path,
    SUM(LocksBroken) as TotalLocksBroken
FROM `.sys/partition_stats`
GROUP BY Path
ORDER BY TotalLocksBroken DESC
LIMIT 10;
```

Пример запроса для просмотра истории сломанных блокировок по партициям:

```
SELECT
    IntervalEnd,
    LocksBroken,
    Path
FROM `.sys/top_partitions_by_tli_one_hour`
WHERE IntervalEnd BETWEEN Timestamp("2000-01-01T00:00:00Z") AND Timestamp("2099-12-31T00:00:00Z")
ORDER BY IntervalEnd DESC, LocksBroken DESC
LIMIT 100;
```

Логирование TLI

Логирование инвалидации блокировок транзакций (TLI) позволяет установить, у какого запроса были сломаны блокировки **(жертва)** и какой запрос их сломал **(нарушитель)**.

Включение логирования

Для получения детальных логов о конфликтах блокировок установите уровень `INFO` (числовое значение `6`) для компонента `TLI`.

Добавьте в конфигурацию кластера:

```
log_config:
  entry:
    - component: "TLI"
      level: 6  # INFO
```

Параметр `log_config` поддерживает динамическое обновление без перезапуска узлов.

После включения сервер записывает логи при каждой сломанной блокировке, а `VictimQuerySpanId` начинает появляться в сообщении об ошибке SDK.

> **Примечание**
>
> Базовый размер лога для одного TLI-события составляет 10–20 КБ. Фактический объем зависит от размера текста логируемых запросов (виновника и жертвы) и транзакций, частью которых они являются, поскольку все эти данные записываются в лог.
> Если конфликты редки, нагрузка на логирование незначительна. При высокой частоте TLI суммарный объём логов может быть существенным.
>
> Чтобы исключить таблицы с ожидаемыми конфликтами из TLI-диагностики, используйте параметр `tli_config.ignored_table_regexes`.

Структура логов

Когда происходит TLI, сервер записывает в лог записи от каждого компонента (`DataShard` или `SessionActor`) для каждой из сторон конфликта (жертвы или нарушителя). Количество записей зависит от количества запросов в транзакциях.

Пример сценария:

Устанавливается две сессии к СУБД: жертва и нарушитель.

Время	QuerySpanId	Роль	Запрос
T1	`1111111111111111`	Жертва	`SELECT * FROM Orders WHERE OrderId = 42` (устанавливает блокировку)
T2	`2222222222222222`	Нарушитель	`UPDATE Orders SET Status = 'done' WHERE OrderId = 42` (ломает блокировку)
T3	`3333333333333333`	Жертва	`UPDATE Orders SET Amount = 100 WHERE OrderId = 42` (коммит — падает)

> **Примечание**
>
> В приводимых далее примерах содержимого логов порядок следования отдельных полей внутри записей может отличаться.

Лог нарушителя (DataShard):

```
component=DataShard tabletId=<tablet-id> message="Write transaction broke other locks" breakerQuerySpanId=222
2222222222222 victimQuerySpanId=1111111111111111
```

Лог нарушителя (SessionActor):

```
component=SessionActor message="Query had broken other locks" breakerTxSpanId=2222222222222222 querySpanId=22
22222222222222 queryText="UPDATE Orders SET Status = 'done' WHERE OrderId = 42"
```

Лог жертвы (DataShard):

```
component=DataShard tabletId=<tablet-id> message="Write transaction was a victim of broken locks" victimQuery
SpanId=1111111111111111
```

Лог жертвы (SessionActor):

В этом примере SessionActor записывает две записи — по одной на каждый запрос транзакции-жертвы:

```
component=SessionActor message="Query was a victim of broken locks" victimTxSpanId=1111111111111111 querySpan
Id=1111111111111111 queryText="SELECT * FROM Orders WHERE OrderId = 42"

component=SessionActor message="Query was a victim of broken locks" victimTxSpanId=1111111111111111 querySpan
Id=3333333333333333 queryText="UPDATE Orders SET Amount = 100 WHERE OrderId = 42"
```

Поля логов

Поле	Описание	Где встречается
`component`	Источник записи: `DataShard` или `SessionActor`	Все логи TLI
`message`	Тип события (см. примеры выше)	Все логи TLI
`tabletId`	Идентификатор таблета DataShard	Логи DataShard
`victimQuerySpanId`	Идентификатор запроса, чьи блокировки были сломаны	Логи жертвы DataShard, логи нарушителя DataShard
`breakerQuerySpanId`	Идентификатор запроса, который сломал блокировки	Логи нарушителя DataShard
`victimTxSpanId`	Идентификатор запроса-жертвы, чьи блокировки были сломаны. Совпадает с victimQuerySpanId в логах DataShard.	Логи жертвы SessionActor
`breakerTxSpanId`	Идентификатор запроса-нарушителя, который сломал блокировки. Совпадает с breakerQuerySpanId в логах DataShard.	Логи нарушителя SessionActor
`querySpanId`	Идентификатор конкретного запроса в записи SessionActor	Логи SessionActor
`queryText`	SQL конкретного запроса в записи SessionActor	Логи SessionActor

Анализ логов

По `VictimQuerySpanId` из сообщения об ошибке SDK можно найти все связанные события:

1. **Поиск идентификатора запроса-жертвы**: в логе жертвы поле `victimQuerySpanId` содержит идентификатор SELECT-запроса, который установил сломанные блокировки.
2. **Поиск текста запроса-жертвы**: в логе жертвы есть запись, у которой `victimTxSpanId` совпадает с `querySpanId`. Поле `queryText` этой записи содержит текст SELECT-запроса, который установил сломанные блокировки.
3. **Поиск идентификатора запроса-нарушителя**: в логе нарушителя DataShard есть запись, у которой поле `victimQuerySpanId` содержит то же самое значение (идентификатор запроса-жертвы). Из этой записи берётся значение `breakerQuerySpanId`, которое является идентификатором запроса-нарушителя.
4. **Поиск текста запроса-нарушителя**: в логе нарушителя есть запись, у которой `breakerTxSpanId` содержит идентификатор запроса-нарушителя и в то же время совпадает с `querySpanId`. Поле `queryText` этой записи содержит текст запроса-нарушителя.
5. **Получение полного контекста транзакций**: все записи SessionActor с одним и тем же `victimTxSpanId` содержат сведения о запросах транзакции-жертвы. Аналогично, все записи SessionActor с одним и тем же `breakerTxSpanId` содержат сведения о запросах транзакции-нарушителя.

Утилита find_tli_chain

Для автоматического анализа логов TLI в репозитории YDB есть утилита find_tli_chain.py.

Данная утилита принимает на вход параметры:

- `VictimQuerySpanId` - идентификатор сломанного запроса из сообщения об ошибке SDK;
- `LogFile` - имя лог-файла для анализа логов.

Подразумевается, что логирование TLI было предварительно включено в конфигурации, лог файлы собраны с узлов базы данных и объединены в один файл.

```
python3 find_tli_chain.py <VictimQuerySpanId> <LogFile>
```

Пример:

```
python3 find_tli_chain.py 1111111111111111 ydb.log
```

```
================================================
  TLI Chain
================================================

VictimQuerySpanId: 1111111111111111

VictimQueryText: SELECT * FROM Orders WHERE OrderId = 42

BreakerQuerySpanId: 2222222222222222

BreakerQueryText: UPDATE Orders SET Status = 'done' WHERE OrderId = 42

================================================
  VictimTx
================================================

SELECT * FROM Orders WHERE OrderId = 42

UPDATE Orders SET Amount = 100 WHERE OrderId = 42

================================================
  BreakerTx
================================================

UPDATE Orders SET Status = 'done' WHERE OrderId = 42
```

Утилита выводит:

- **TLI Chain** — `VictimQuerySpanId`, `VictimQueryText`, `BreakerQuerySpanId`, `BreakerQueryText`;
- **VictimTx** — все запросы транзакции-жертвы;
- **BreakerTx** — все запросы транзакции-нарушителя.

Дополнительные параметры:

Параметр	Описание
`--window-sec N`	Временное окно поиска вокруг события (по умолчанию 10 с)
`--no-color`	Отключить цветной вывод

Примечание

Утилита корректно обрабатывает несортированные и смешанные лог-файлы: фильтрация основана на значениях временных меток, а не на позициях строк.

Ошибки «overloaded»

В статье описано, при каких условиях запрос к YDB завершается ошибкой `OVERLOADED`, и что делать дальше.

Ниже перечислены типичные причины. Проверить их можно по инструкции в разделе Диагностика. Как повторять запросы, не разгоняя нагрузку, описано в конце статьи.

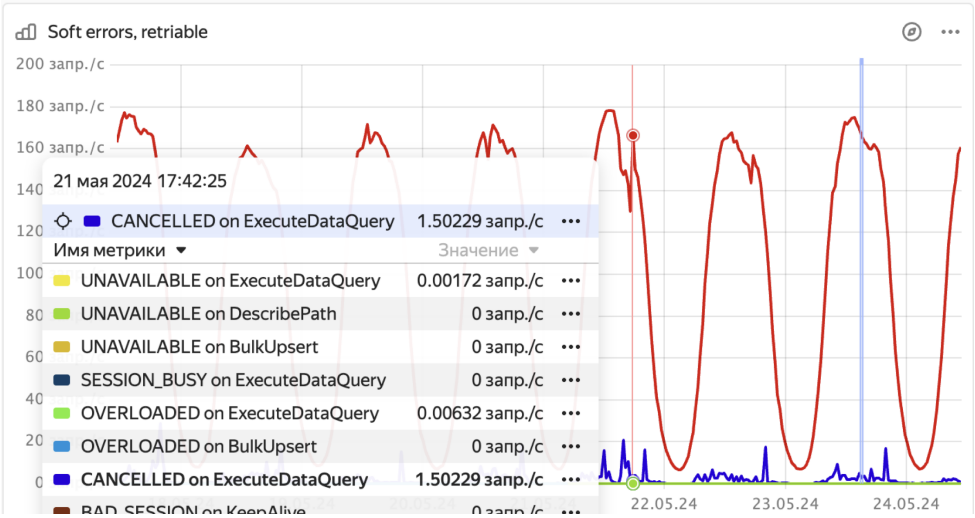
YDB возвращает ошибки `OVERLOADED` в следующих случаях:

- Из-за перегруженной партиции таблицы:
  1. На партиции превышен лимит числа одновременно обрабатываемых транзакций. Для data-транзакций это обычно проявляется как превышение порога в 15000 in-flight запросов на шард.
  2. Шард отстаёт по завершению уже принятых транзакций. Это происходит, когда у него накапливается хвост незавершённых операций и новые запросы временно отклоняются, чтобы не увеличивать отставание.
  3. Локальная база данных шарда перегружена: например, слишком много транзакций уже находится в обработке, in-memory данные слишком выросли или compaction не успевает за нагрузкой. В этом случае шард может начать отклонять часть новых запросов, даже если лимит по числу in-flight транзакций на уровне data shard ещё не достигнут.

  Как правило, эти причины возникают из-за одних и тех же первопричин: неравномерного распределения нагрузки, слишком высокой частоты записи или недостатка ресурсов.
- Из-за того, что партиция таблицы временно находится не в рабочем состоянии:
  1. Партиция может разделяться или объединяться, перезапускаться, ещё не завершить служебный переход в рабочее состояние или находиться в другом промежуточном состоянии, в котором новые запросы временно не принимаются.
- Из-за перегруженного CDC:
  1. Превышен лимит размера выходной очереди CDC: 10000 элементов или 125 МБ.
- Из-за перегруженного узла:
  1. Количество открытых сессий с узлом YDB достигло лимита в 1000. Это может указывать на проблему в логике приложения.

Диагностика

1. Откройте панель мониторинга Grafana **DB overview**.
2. В разделе **API details** проверьте, есть ли всплески частоты запросов со статусом `OVERLOADED` на диаграмме **Soft errors (retriable)**.



3. Чтобы проверить, не связаны ли всплески ошибок `OVERLOADED` с превышением лимита в 15000 запросов на партицию таблицы:
   - Через UI:
     1. Во YDB UI перейдите на вкладку **Databases** и нажмите на базу данных.
     2. На вкладке **Navigation** убедитесь, что требуемая база данных выбрана.
     3. Откройте вкладку **Diagnostics**.
     4. Откройте вкладку **Top shards**.
     5. На вкладках **Immediate** и **Historical** отсортируйте таблетки по столбцу **InFlightTxCount** и проверьте, не превышают ли максимальные значения лимит в 15000 запросов.
   - Через систему мониторинга:
     1. Найдите дашборд, отображающий метрики базы данных.
     2. Выберите `category = app`, `type = DataShard`, `sensor = SUM(DataShard/ImmediateTxInFly)`.
     3. Проверьте, не растут ли его значения до величин, близких к 15000 или превышающих этот порог.
4. Чтобы проверить, не связаны ли всплески ошибок `OVERLOADED` с отставанием принятых транзакций:
   4.1. Найдите дашборд, отображающий метрики базы данных.
   4.2. Выберите `category = app`, `type = DataShard`, `sensor = MAX(DataShard/TxCompleteLag)`.
   4.3. Проверьте, не растут ли его значения до 300 секунд или превышающих этот порог.
5. Чтобы проверить, не связаны ли всплески ошибок `OVERLOADED` с перегрузкой локальной базы данных шарда:
   5.1. Найдите дашборд, отображающий метрики базы данных.
   5.2. Выберите `category = executor`, `type = DataShard`, `sensor = MAX(RejectProbability)`.
   5.3. Проверьте, не появляются ли у него значения выше нуля. Рост этого сенсора означает, что local DB начала вероятностно отклонять часть новых запросов из-за перегрузки.
6. Чтобы проверить, не связаны ли всплески ошибок `OVERLOADED` со слишком частыми слияниями и разделениями таблеток, см. Избыточные разделения и слияния партиций таблиц.

7. Чтобы проверить, не связаны ли всплески ошибок `OVERLOADED` с превышением лимита в 1000 открытых сессий, см. диаграмму **Session count by host** на панели мониторинга Grafana **DB status**.
8. См. статью Перегруженные таблетки data shard.

Встроенные механизмы обработки ошибок

Если запрос к YDB возвращает ошибку `OVERLOADED`, повторите операцию с экспоненциальной задержкой (пауза между повторами всё длиннее — обычно в несколько раз) и джиттером (случайным разбросом интервала), чтобы не создавать синхронные волны повторных запросов. YDB SDK реализует политики повторов для временных ошибок; обзор и настройка — в разделе Обработка ошибок, примеры — в Выполнение повторных попыток.

Не переносите на `OVERLOADED` ту же тактику, что и при **таймауте на клиенте**: при обрыве по таймауту запрос на сервере может еще выполняться, а частые повторы усугубляют очередь. Подробнее — в статье Каскадный эффект при частых повторных попытках.

Каскадный эффект при частых повторных попытках

В статье разобран типичный антипаттерн: **агрессивные повторы запроса при клиентском таймауте**, когда предыдущие экземпляры запроса все еще работают на сервере и конкурируют за ресурсы.

Такой сценарий часто сочетается с ростом задержек и ответами `OVERLOADED` при перегрузке шардов; см. Ошибки «overloaded» и Перегруженные таблетки data shard.

Статья объясняет, почему «уменьшить таймаут и повторять запросы чаще» при перегрузке ухудшает ситуацию, а не улучшает её.

Каскадный эффект возникает, когда клиент начинает часто повторять запросы, у которых время выполнения превышает установленный таймаут.

Механизм возникновения

Время выполнения запроса превышает таймаут клиента. Клиент начинает повторять запросы, но оригинальные запросы продолжают выполняться на сервере. Каждый следующий повторный запрос выполняется дольше предыдущего, так как сервер обрабатывает больше запросов одновременно. В какой-то момент задержка запросов постоянно превышает таймаут клиента. Клиент продолжает повторять запросы, увеличивая нагрузку и ещё больше повышая задержку.

Особенности YDB

В акторной модели YDB отмена запросов, которые уже выполняются, имеет ограничения. Особенно это проявляется при клиентских таймаутах — когда клиент закрывает gRPC стрим до получения ответа, запрос может продолжать выполняться на сервере.

Восстановление нормальной работы

Шард может восстановить нормальную работу только при временном полном снятии нагрузки, перезапуске шарда или разделении шарда по нагрузке.

Рекомендации

- **После таймаута на клиенте** не запускайте тот же запрос в цикле с коротким интервалом: серверная работа может продолжаться, очередь растет, задержка и число таймаутов увеличиваются.
- **После явной временной ошибки** (в том числе `OVERLOADED`) действуйте по рекомендациям в Ошибки «overloaded» о повторах без разгона нагрузки. Убедитесь по метрикам или нагрузке, что система не в режиме устойчивой перегрузки.
- **Для политик retry в SDK** см. Обработка ошибок и Выполнение повторных попыток.

Устранение неполадок спиллинга

Этот раздел предоставляет информацию по устранению неполадок для распространенных проблем со спиллингом в YDB. Спиллинг — это механизм управления памятью, который временно сохраняет промежуточные вычислительные данные на диск при нехватке оперативной памяти. Эти ошибки могут возникать во время выполнения запросов, когда система пытается использовать функциональность спиллинга, и могут наблюдаться в логах и ответах запросов.

Частые проблемы

- Permission denied - Недостаточные права доступа к директории спиллинга
- Spilling Service not started - Попытка использования спиллинга при выключенном Spilling Service
- Total size limit exceeded - Превышен максимальный суммарный размер файлов спиллинга

См. также

- Конфигурация спиллинга
- Концепция спиллинга
- Конфигурация контроллера памяти
- Мониторинг YDB
- Диагностика производительности

Permission denied

Недостаточные права доступа к директории спиллинга не позволяют YDB записывать данные на диск во время операций спиллинга. Это может привести к сбою запросов, если для их выполнения требуется спиллинг для обработки больших объемов данных.

Диагностика

Проверьте, существует ли директория спиллинга и имеет ли она правильные права доступа:

- Убедитесь, что директория спиллинга существует (см. Конфигурация спиллинга для получения информации о том, как найти директорию спиллинга).
- Убедитесь, что директория имеет права на чтение и запись для пользователя, под которым запущен `ydbd`.
- Проверьте права доступа к директории спиллинга.
- Убедитесь, что пользователь, под которым запускается `ydbd`, может читать и записывать данные в директорию.
- Если статические и динамические узлы запускаются под разными пользователями ОС и используют один `root`, убедитесь, что у обоих пользователей есть права на чтение и запись директории.

Рекомендации

Если права доступа заданы неправильно:

1. Измените владельца директории на пользователя, под которым запускается `ydbd`.
2. Убедитесь, что установлены права на чтение и запись для владельца директории. Если статические и динамические узлы работают под разными пользователями ОС, выдайте обоим права на чтение и запись (например, `chmod 1777`).
3. Перезапустите процесс `ydbd` для применения изменений.

> **Примечание**
>
> Директория спиллинга автоматически создается YDB при запуске процесса. Если директория не существует, проверьте, что параметр `root` в конфигурации спиллинга установлен правильно.

Spilling Service not started

Спиллинг выключен, но при выполнении запроса закончилась память. Когда спиллинг выключен, YDB не может выгрузить промежуточные данные на диск, что приводит к сбою запроса при исчерпании доступной памяти.

Диагностика

Проверьте конфигурацию сервиса спиллинга:

- Убедитесь, что `table_service_config.spilling_service_config.local_file_config.enable` установлен в `true`.

Рекомендации

Для исправления этой ошибки включите спиллинг:

1. Установите `table_service_config.spilling_service_config.local_file_config.enable`: `true` в вашей конфигурации.

Подробнее о включении спиллинга см. в разделе Включение.

> **Примечание**
>
> Подробнее об архитектуре спиллинга см. в разделе Архитектура спиллинга в YDB.

Total size limit exceeded

Превышен максимальный суммарный размер файлов спиллинга (параметр `max_total_size`). Это происходит, когда общий размер всех файлов спиллинга достигает настроенного лимита, что препятствует новым операциям спиллинга.

Диагностика

Проверьте текущее использование спиллинга:

- Отслеживайте общий размер файлов спиллинга в директории спиллинга
- Проверьте текущее значение параметра `max_total_size`
- Просмотрите доступное дисковое пространство в расположении директории спиллинга
- Проверьте, есть ли зависшие файлы спиллинга, которые должны были быть очищены

Рекомендации

Для решения этой проблемы:

1. **Увеличьте лимит размера спиллинга:**
   - Если на диске достаточно свободного места, увеличьте параметр `max_total_size` в конфигурации
   - Рекомендуется увеличить значение на 20-50% от текущего
2. **Расширьте дисковое пространство:**
   - Если свободного места на диске недостаточно, добавьте дополнительное дисковое пространство
   - Убедитесь, что директория спиллинга находится на диске с достаточным объемом
3. **Попробуйте повторить запрос:**
   - Дождитесь завершения других ресурсоемких запросов
   - Повторите выполнение запроса в менее загруженное время

Пример диагностики перегруженных шардов

В этой статье рассматривается пример диагностики перегруженных шардов и решения этой проблемы.

Дополнительную информацию о перегруженных шардах и причинах их перегрузки см. в статье Перегруженные таблетки data shard.

Статья начинается с описания возникшей проблемы. Затем мы проанализируем графики в Grafana и информацию на вкладке **Diagnostics** в YDB UI, чтобы найти решение, и проверим его эффективность.

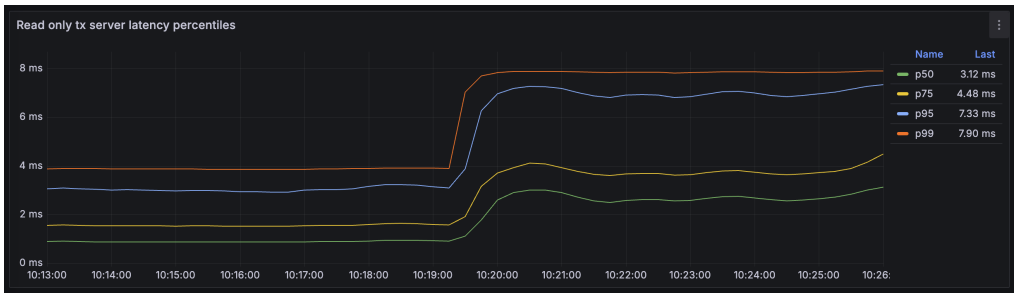
В конце статьи приводятся шаги по воспроизведению проблемы.

Описание проблемы

Вас уведомили о задержках при обработке пользовательских запросов в вашей системе.

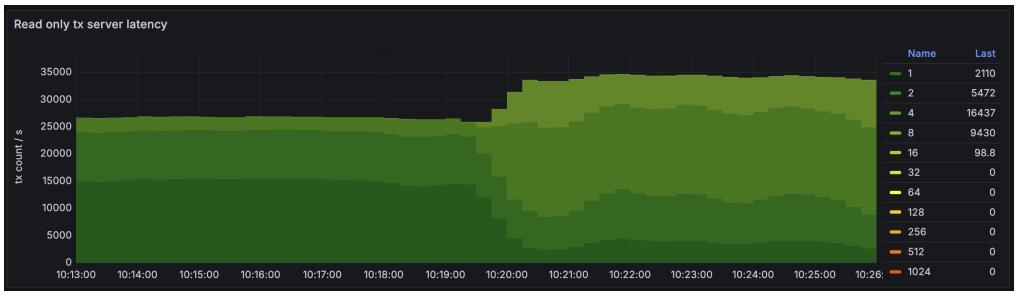
> **Примечание**
>
> Речь идёт о запросах к строковой таблице, управляемой data shard'ом.

Рассмотрим графики **Latency** на панели мониторинга Grafana DB overview и определим, имеет ли отношение наша проблема к кластеру YDB:



См. описание графика

> График отображает процентили задержек транзакций. Примерно в `10:19:30` эти значения выросли в два-три раза.



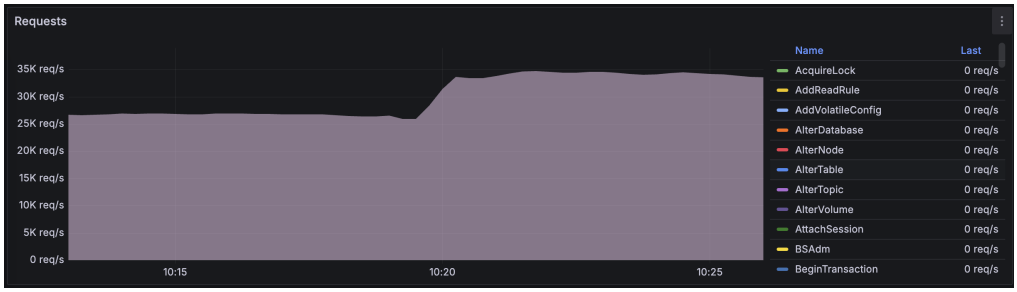
См. описание графика

> График отображает тепловую карту (heatmap) задержек транзакций. Транзакции группируются на основании их задержек, каждая группа (bucket) окрашивается в свой цвет. Таким образом, этот график показывает как количество транзакций, обрабатываемых YDB в секунду (по вертикальной оси), так и распределение задержек среди транзакций (цветовая дифференциация).
>
> К `10:20:30` доля транзакций с минимальными задержками (`Группа 1`, тёмно-зелёный) упала в четыре-пять раз. `Группа 4` выросла примерно в пять раз, а также выделилась новая группа транзакций с ещё более высокими задержками — `Группа 8`.

Таким образом, мы видим, что задержки действительно выросли. Теперь нам необходимо локализовать проблему.

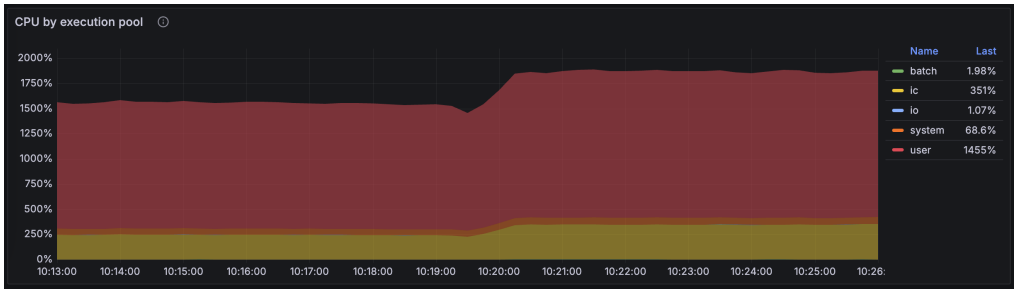
Диагностика

Давайте определим причину роста задержек. Могли ли они увеличиться из-за возросшей нагрузки? Посмотрим на график **Requests** в секции **API details** панели мониторинга Grafana DB overview:



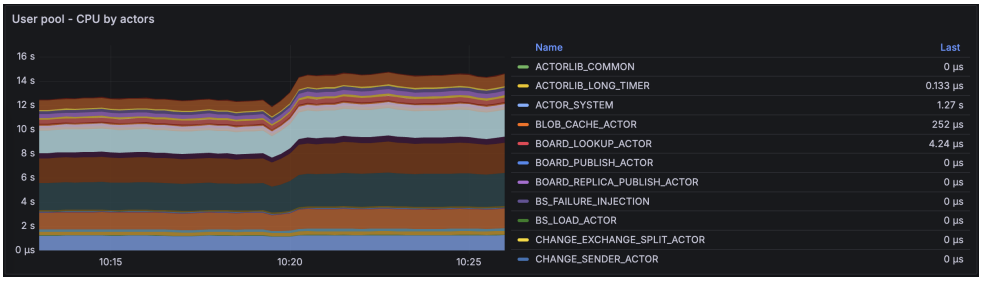
Количество пользовательских запросов выросло приблизительно с 27 000 до 35 000 в `10:20:00`. Но может ли YDB справиться с увеличившейся нагрузкой без дополнительных аппаратных ресурсов?

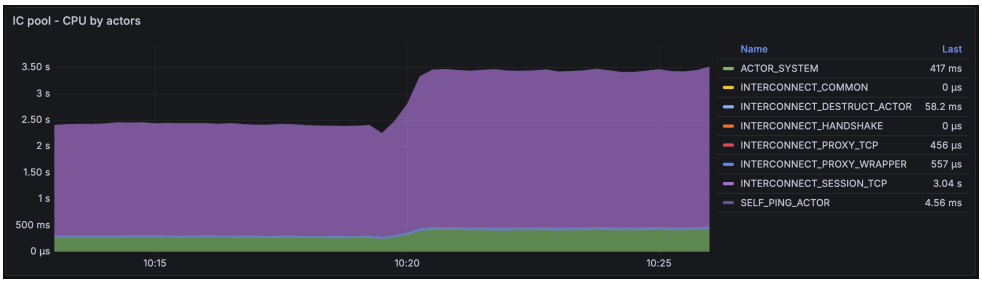
Загрузка CPU увеличилась, что видно на графике **CPU by execution pool**.

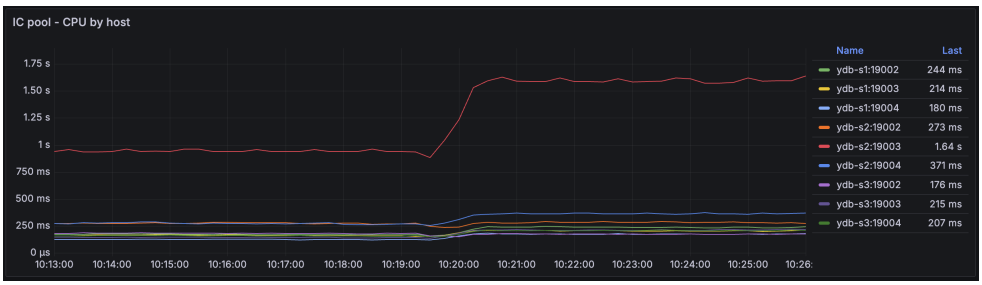


См. графики на панели мониторинга Grafana **CPU**

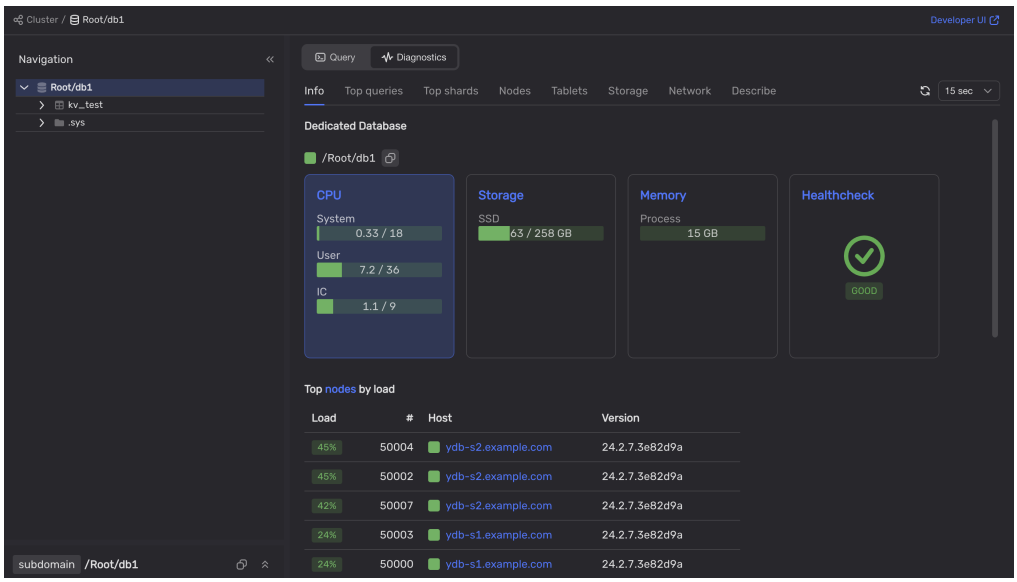
Графики на панели мониторинга Grafana **CPU** показывают рост нагрузки на CPU в пуле ресурсов пользователей и интерконнекта:





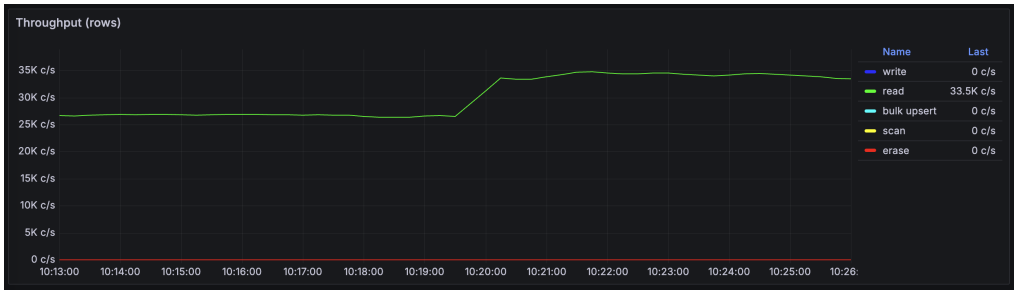


Мы также можем взглянуть на общее использование CPU на вкладке **Diagnostics** в YDB UI:



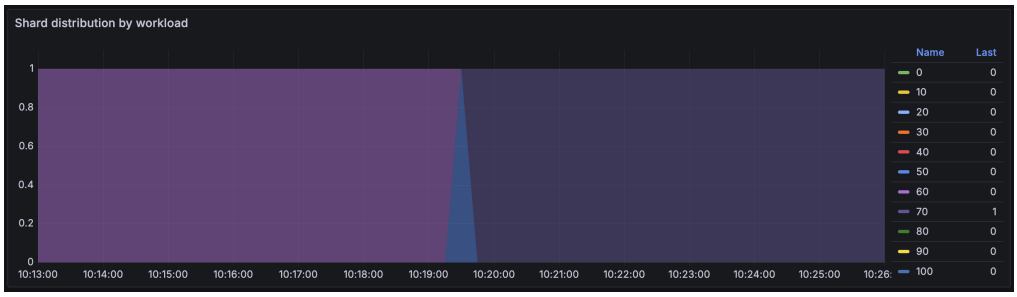
Кластер YDB не использует все ресурсы CPU.

Взглянув на секции **DataShard** и **DataShard details** на панели мониторинга Grafana DB overview, мы увидим, что после роста нагрузки на кластер один из data shard'ов был перегружен.



См. описание графика

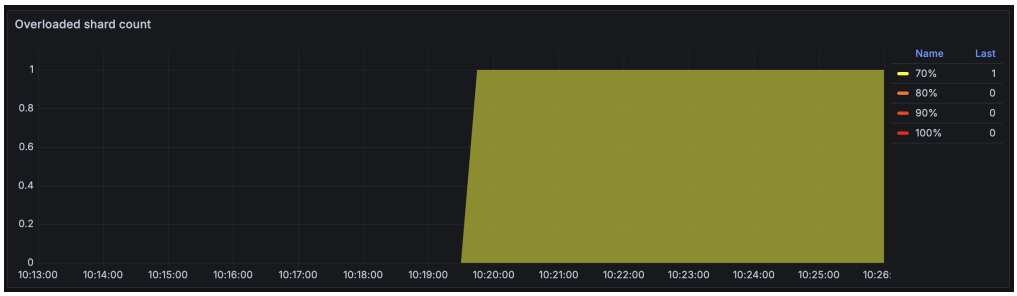
Этот график показывает, что количество читаемых строк в базе данных YDB увеличилось с ~26 000 до ~33 500 строк в секунду.



См. описание графика

Этот график отображает тепловую карту распределения data shard'ов по нагрузке. Каждый data shard потребляет от 0% до 100% ядра CPU. Data shard'ы делятся на десять групп по занимаемой ими доле ядра CPU — 0-10%, 10-20% и т.д. Эта тепловая карта показывает количество data shard'ов в каждой группе.

График показывает только один data shard, нагрузка на который изменилась примерно в `10:19:30` — data shard перешёл в `Группу 70`, содержащую шарды, нагруженные на 60–70%.

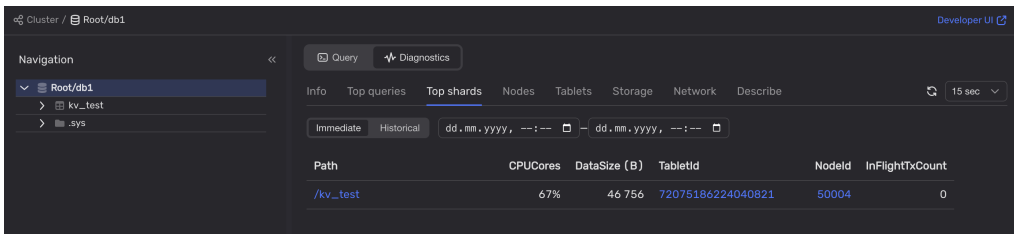


См. описание графика

По аналогии с предыдущим графиком, **Overloaded shard count** — это тепловая карта распределения data shard'ов по нагрузке. Однако этот график отображает только data shard'ы с нагрузкой, превышающей 60%.

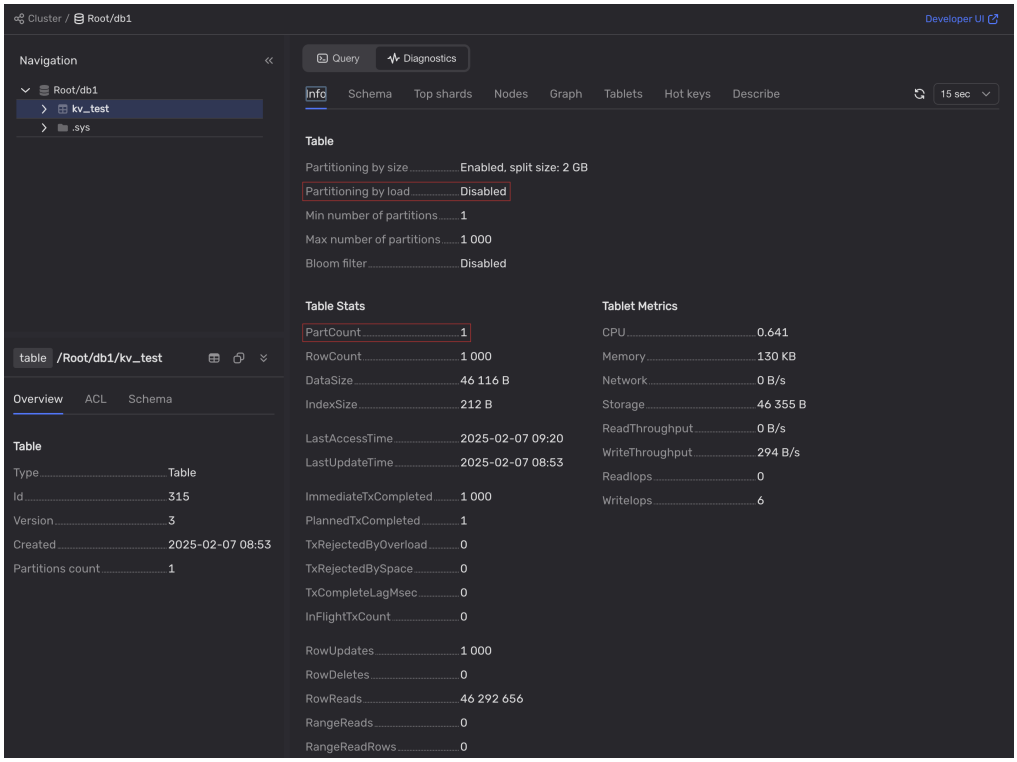
График показывает, что нагрузка на один data shard увеличилась до 70% примерно в `10:19:30`.

Чтобы определить, какую таблицу обслуживает перегруженный data shard, откроем вкладку **Diagnostics > Top shards** в YDB UI:



Мы видим, что один из data shard'ов, обслуживающих таблицу `kv_test`, нагружен на 67%.

Далее давайте взглянем на информацию о таблице `kv_test` на вкладке **Info**:



Важно

Таблица `kv_test` была создана с отключённым партиционированием по нагрузке и содержит только одну партицию.

Это означает, что все запросы к этой таблице обрабатывает один data shard. Учитывая, что data shard'ы — это однопоточные компоненты, обрабатывающие за раз только один запрос, такой подход неэффективен.

Решение

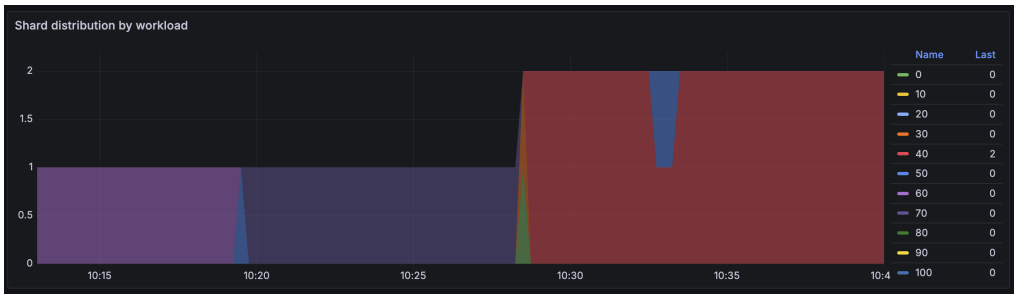
Нам необходимо включить партиционирование по нагрузке для таблицы `kv_test`:

1. В YDB UI выберите базу данных.
2. Откройте вкладку **Query**.
3. Выполните следующий запрос:

```
ALTER TABLE kv_test SET (
    AUTO_PARTITIONING_BY_LOAD = ENABLED
);
```

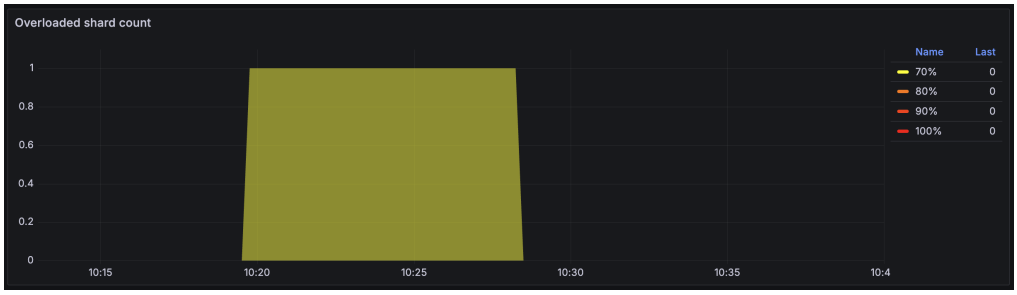
Результат

После включения автоматического партиционирования для таблицы `kv_test` перегруженный data shard разделился на два.



См. описание графика

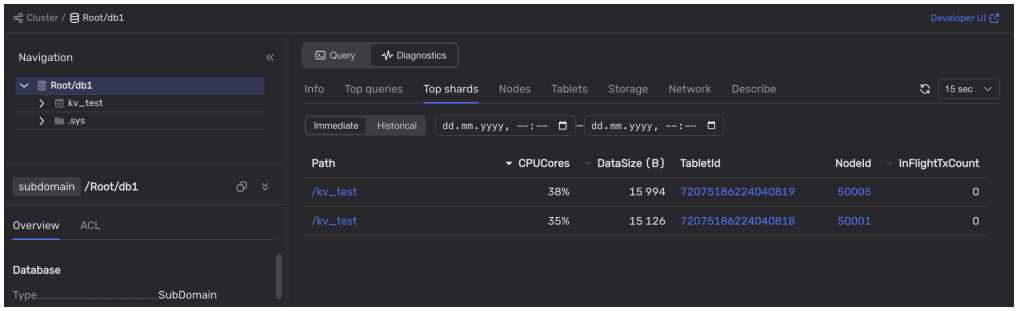
> График показывает, что количество data shard'ов выросло примерно в `10:28:00`. Судя по цвету групп, их нагрузка не превышает 40%.



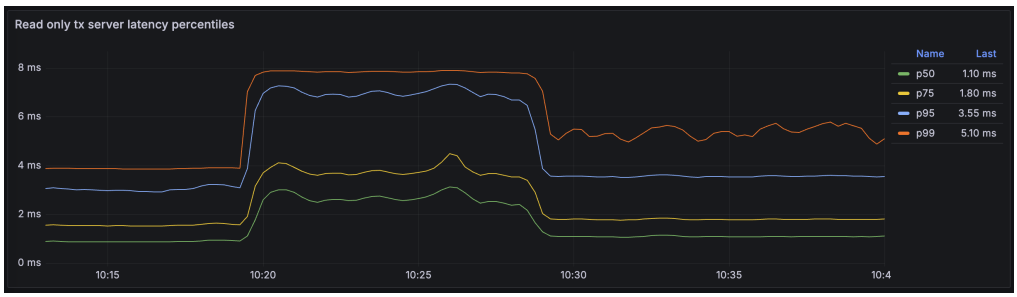
См. описание графика

Перегруженный шард исчез с графика примерно в `10:28:00`.

Теперь два data shard'а обрабатывают запросы к таблице `kv_test`, и ни один из них не перегружен:

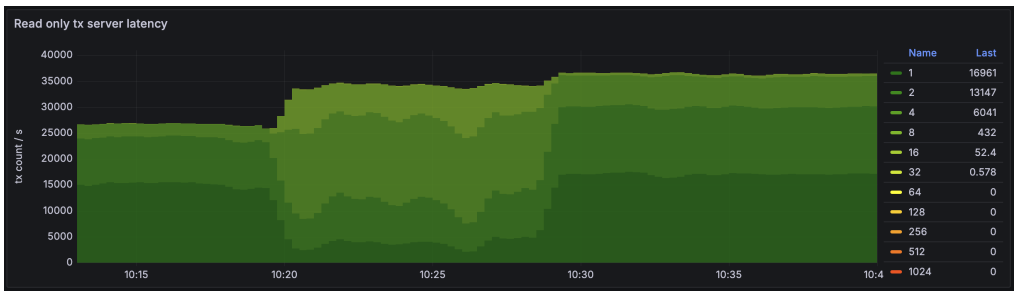


Давайте убедимся, что задержки транзакций вернулись к прежним значениям:



См. описание графика

Примерно в `10:28:00` процентили задержек p50, p75 и p95 упали практически до прежних значений. Задержки p99 сократились не настолько значительно, но всё же уменьшились в два раза.



См. описание графика

Транзакции на этом графике теперь распределены по шести группам. Примерно половина транзакций вернулась в `Группу 1`, то есть их задержки не превышают одной миллисекунды. Более трети транзакций находятся в `Группе 2` с задержками от одной до двух миллисекунд. Одна шестая транзакций — в `Группе 4`. Размеры остальных групп незначительны.

Задержки практически вернулись к уровню до увеличения нагрузки. При этом мы не увеличили расходы на приобретение дополнительных аппаратных ресурсов, а просто включили автоматическое партиционирование по нагрузке, что позволило более эффективно использовать доступные ресурсы.

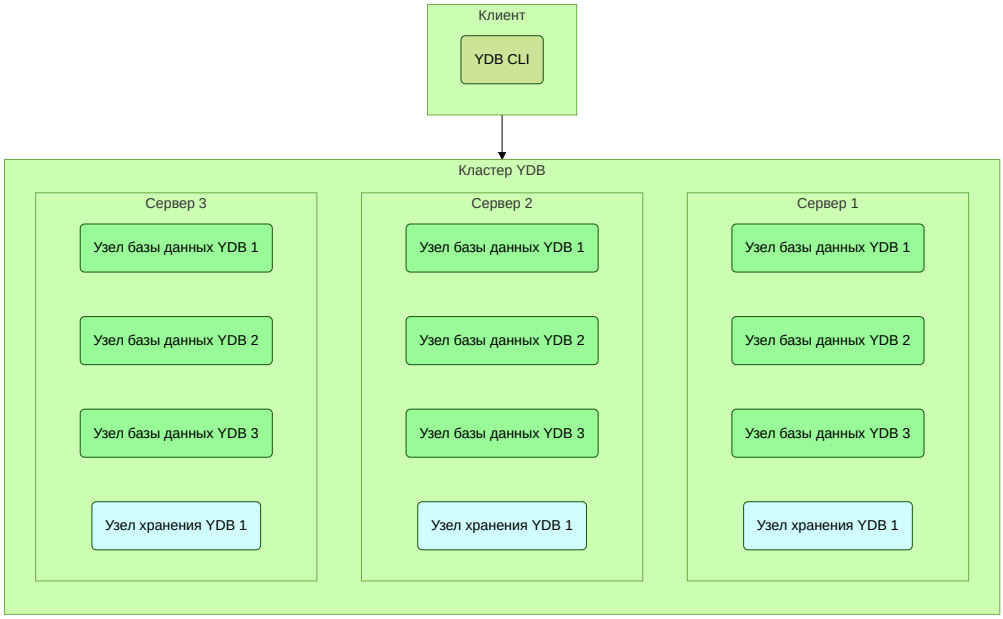
Имя группы	Задержки, мс	Один перегруженный data shard, транзакций в секунду	Несколько data shard'ов, транзакций в секунду
1	0-1	2110	▲ 16961
2	1-2	5472	▲ 13147
4	2-4	16437	▼ 6041
8	4-8	9430	▼ 432

16	8-16	98.8	▼ 52.4
32	16-32	—	▲ 0.578

Тестовый стенд

Топология

Для этого примера мы использовали кластер YDB из трёх серверов на Ubuntu 22.04 LTS. На каждом сервере был запущен один узел хранения и три узла баз данных, обслуживающих одну и ту же базу данных.



Аппаратная конфигурация

Аппаратные ресурсы серверов (виртуальных машин) приведены ниже:

- Платформа: Intel Broadwell
- Гарантированный уровень производительности vCPU: 100%
- vCPU: 28
- Память: 32 GB
- Диски:
  - 3 × 93 GB SSD на каждом узле YDB
  - 20 GB HDD для операционной системы

Тест

Нагрузка на кластер YDB была запущена с помощью команды CLI `ydb workload`. Дополнительную информацию см. в статье Нагрузочное тестирование.

Чтобы воспроизвести нагрузку, выполните следующие шаги:

1. Проинициализируйте таблицы для нагрузочного тестирования:

   ```
   ydb workload kv init --min-partitions 1 --auto-partition 0
   ```

   Мы намеренно отключаем автоматическое партиционирование для создаваемых таблиц используя опции `--min-partitions 1 --auto-partition 0`.

2. Воспроизведите стандартную нагрузку на кластер YDB:

   ```
   ydb workload kv run select -s 600 -t 100
   ```

   Мы запустили простую нагрузку, используя базу данных YDB как Key-Value хранилище. Точнее, мы использовали нагрузку `select` для выполнения `SELECT`-запросов, возвращающих строки по точному совпадению primary ключа.

   Параметр `-t 100` используется для запуска нагрузочного тестирования в 100 потоков.

3. Создайте перегрузку на кластере YDB:

   ```
   ydb workload kv run select -s 1200 -t 250
   ```

   Как только первый тест завершился, мы немедленно запустили тот же самый тест в 250 потоков, чтобы создать перегрузку.

Смотрите также

- Диагностика проблем с производительностью
- Перегруженные таблетки data shard
- Строковые таблицы

Пример диагностики перегруженных шардов и нехватки ресурса CPU

Как и в предыдущей статье, здесь мы рассматриваем пример диагностики перегруженных шардов и решения этой проблемы. Но в этом случае ситуация осложняется недостатком ресурсов CPU.

Дополнительную информацию о проблемах, диагностируемых в этой статье, см. в следующих материалах:

- Перегруженные таблетки data shard;
- Недостаточное быстродействие процессора.

Статья начинается с описания возникшей проблемы. Затем мы проанализируем графики в Grafana и информацию на вкладке **Diagnostics** в YDB UI, чтобы найти решение, и проверим его эффективность.

В конце статьи приводятся шаги по воспроизведению проблемы.

Описание первоначальной проблемы

Вас уведомили о задержках при обработке пользовательских запросов в вашей системе.

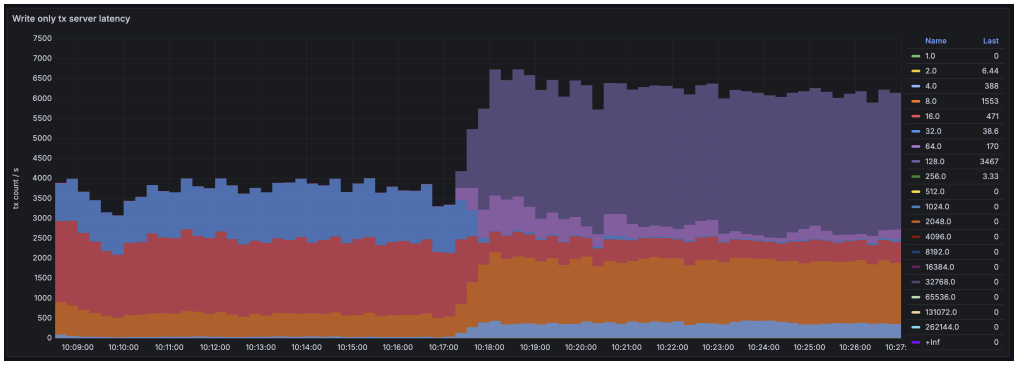
> **Примечание**
>
> Речь идёт о запросах к строковой таблице, управляемой data shard'ом.

Рассмотрим графики **Latency** на панели мониторинга Grafana DB overview и определим, имеет ли отношение наша проблема к кластеру YDB:



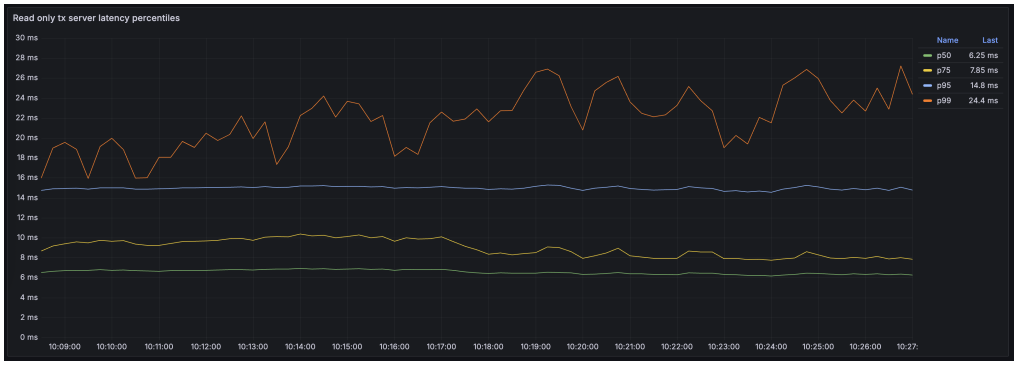
См. описание графика

> График отображает процентили задержек транзакций на запись данных. Примерно в `10:17:00` эти значения выросли в три-четыре раза.



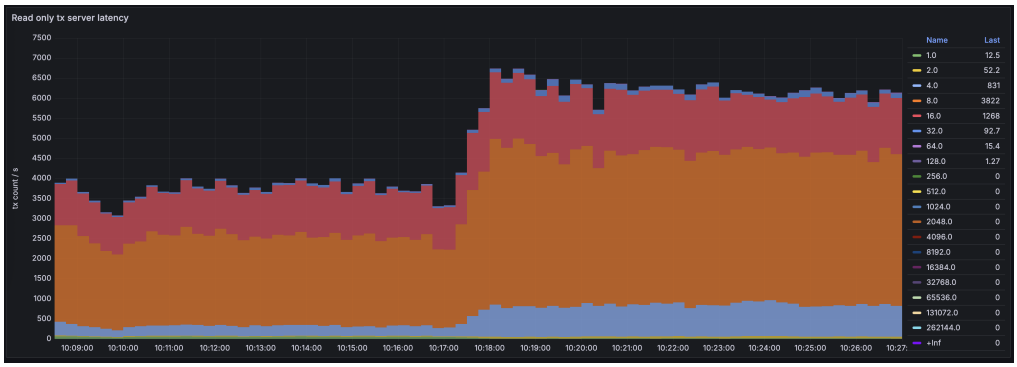
См. описание графика

> График отображает тепловую карту (heatmap) задержек транзакций на запись. Транзакции группируются на основании их задержек, каждая группа (bucket) окрашивается в свой цвет. Таким образом, этот график показывает как количество транзакций, обрабатываемых YDB в секунду (по вертикальной оси), так и распределение задержек среди транзакций (цветовая дифференциация).
>
> К `10:18:00` доля транзакций с задержками от 2 до 4 секунд (`Группа 4.0`, голубой) выросла примерно в десять раз. Доля транзакций с задержками от 4 до 8 секунд (`Группа 8.0`, жёлтый) увеличилась в три раза. Доля транзакций с задержками от 8 до 16 секунд (`Группа 16.0`, красный) уменьшилась в три раза. Доля транзакций с задержками от 16 до 32 секунд (`Группа 32.0`, синий) практически исчезла. А также выделились новые группы транзакций с ещё более высокими задержками — `Группа 64.0` и `Группа 128.0`.



См. описание графика

> График отображает процентили задержек транзакций на чтение данных и демонстрирует рост значений p99.



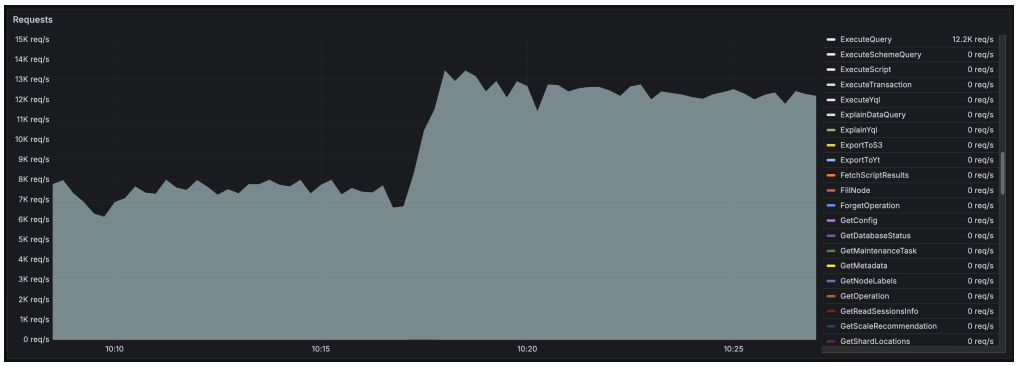
См. описание графика

> График отображает тепловую карту (heatmap) задержек транзакций на чтение данных. Транзакции группируются на основании их задержек, каждая группа (bucket) окрашивается в свой цвет. Таким образом, этот график показывает как количество транзакций, обрабатываемых YDB в секунду (по вертикальной оси), так и распределение задержек среди транзакций (цветовая дифференциация).
>
> К `10:18:00` мы видим рост количества транзакций на чтение начиная с `Группы 4.0`, но соотношения между группами транзакций остаются примерно одинаковыми.

Таким образом, мы видим, что задержки действительно выросли. Теперь нам необходимо локализовать проблему.

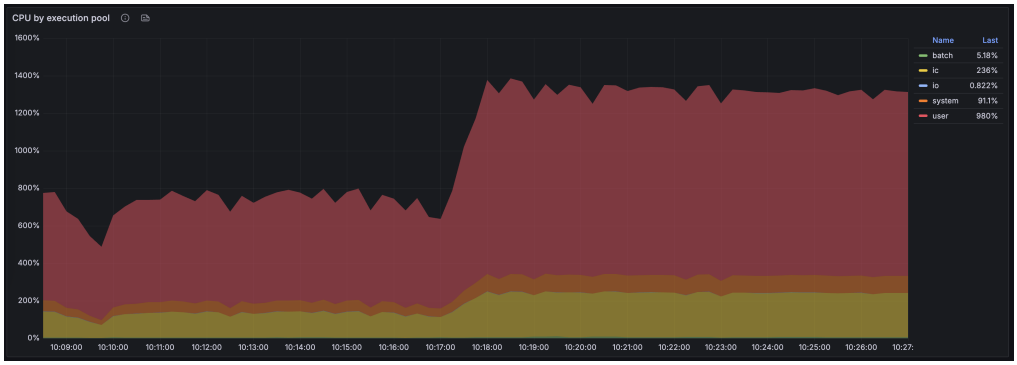
Диагностика

Давайте определим причину роста задержек. Могли ли они увеличиться из-за возросшей нагрузки? Посмотрим на график **Requests** в секции **API details** панели мониторинга Grafana DB overview:



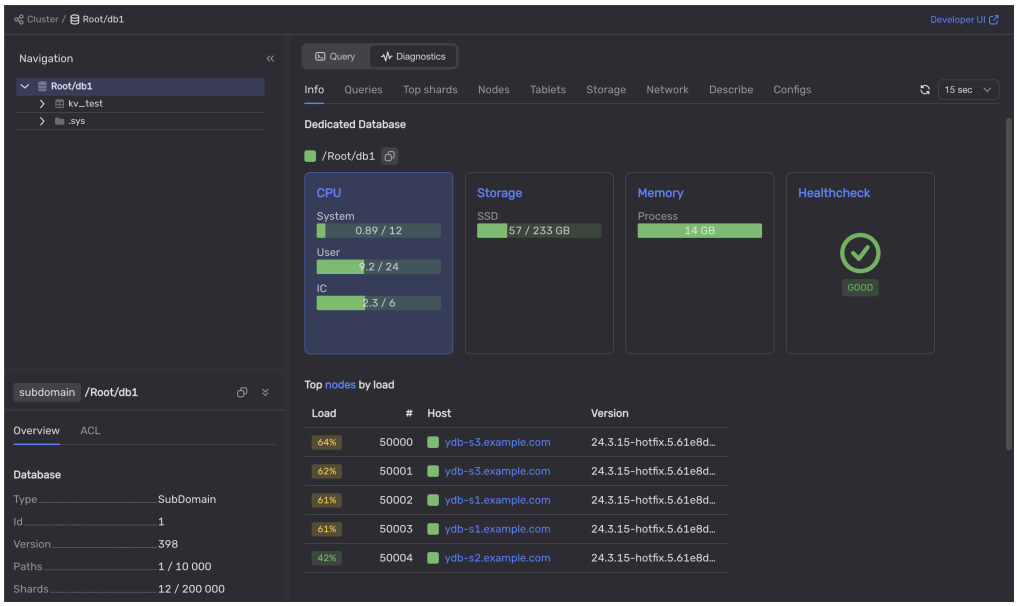
Количество пользовательских запросов выросло приблизительно с 8 000 до 13 000 в `10:20:00`. Но может ли YDB справиться с увеличившейся нагрузкой без дополнительных аппаратных ресурсов?

Загрузка CPU увеличилась, что видно на графике **CPU by execution pool**.



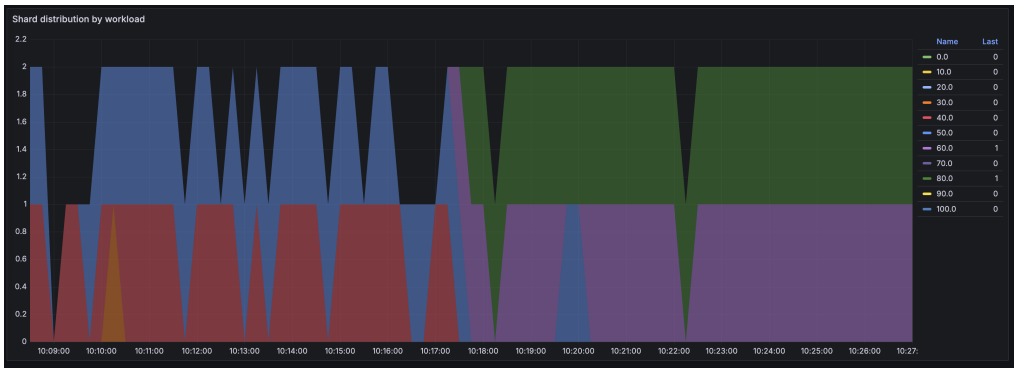
Мы видим рост нагрузки на CPU в пуле ресурсов пользователей (красный) и интерконнекта (жёлтый).

Мы также можем взглянуть на общее использование CPU на вкладке **Diagnostics** в YDB UI:



Кластер YDB не использует все ресурсы CPU.

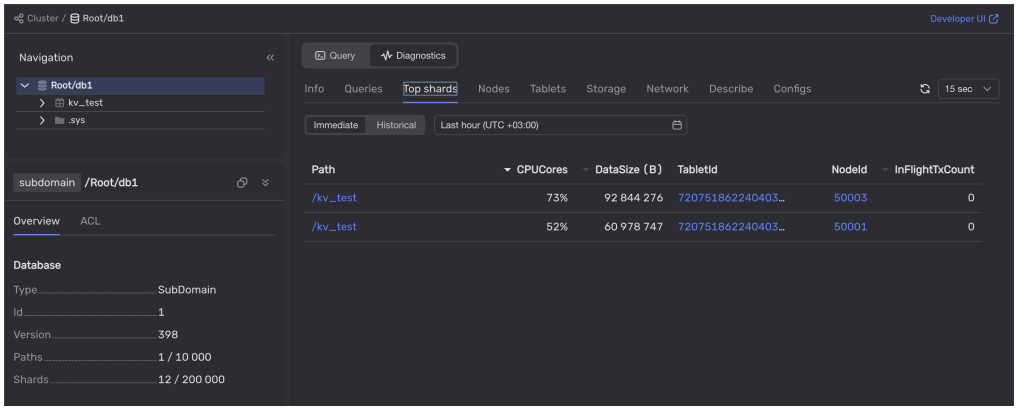
Взглянув на секцию **DataShard details** на панели мониторинга Grafana DB overview, мы увидим, что после роста нагрузки на кластер один из data shard'ов был перегружен (нагрузка на него выросла до 73%).



См. описание графика

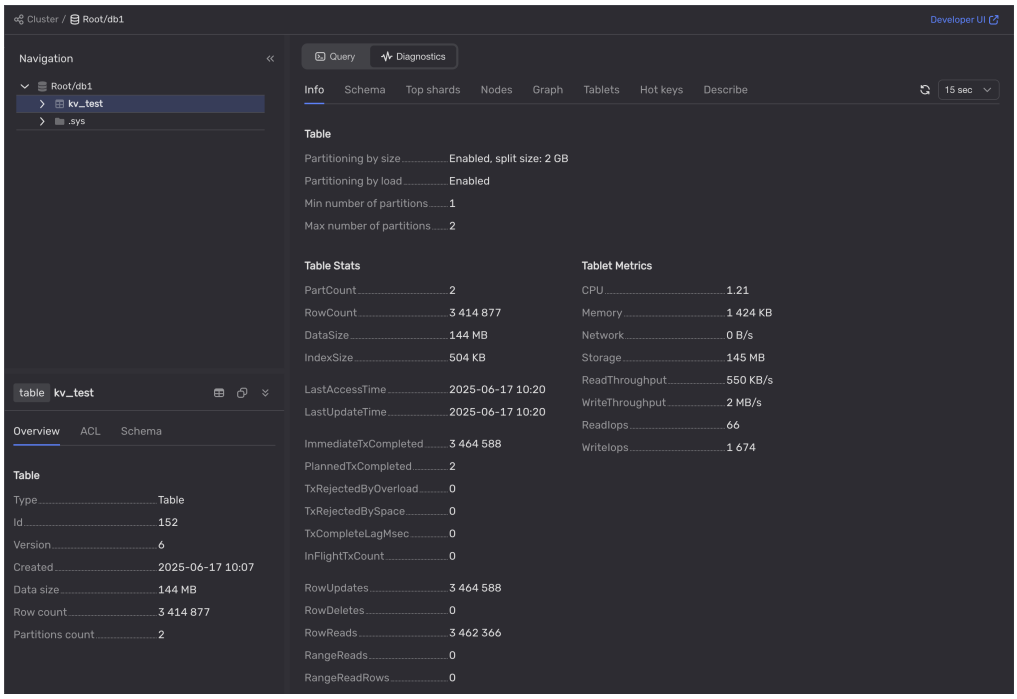
> Этот график отображает тепловую карту распределения data shard'ов по нагрузке. Каждый data shard потребляет от 0% до 100% ядра CPU. Data shard'ы делятся на десять групп по занимаемой ими доле ядра CPU — 0-10%, 10-20% и т.д. Эта тепловая карта показывает количество data shard'ов в каждой группе.
>
> График показывает два data shard'а, нагрузка на которые изменилась примерно в `10:18:00` — один data shard перешёл в `Группу 80`, содержащую шарды, нагруженные на 70–80%, а второй — в `Группу 60`.

Чтобы определить, какую таблицу обслуживает перегруженный data shard, откроем вкладку **Diagnostics > Top shards** в YDB UI:



Мы видим, что один из data shard'ов, обслуживающих таблицу `kv_test`, нагружен на 73%.

Далее давайте взглянем на информацию о таблице `kv_test` на вкладке **Info**:



> **Важно**
>
> Таблица `kv_test` была создана с включённым партиционированием по нагрузке, но максимальное количество партиций ограничено двумя.
>
> Учитывая, что data shard'ы являются однопоточными компонентами и обрабатывают за раз только один запрос, такой подход неэффективен.

Решение

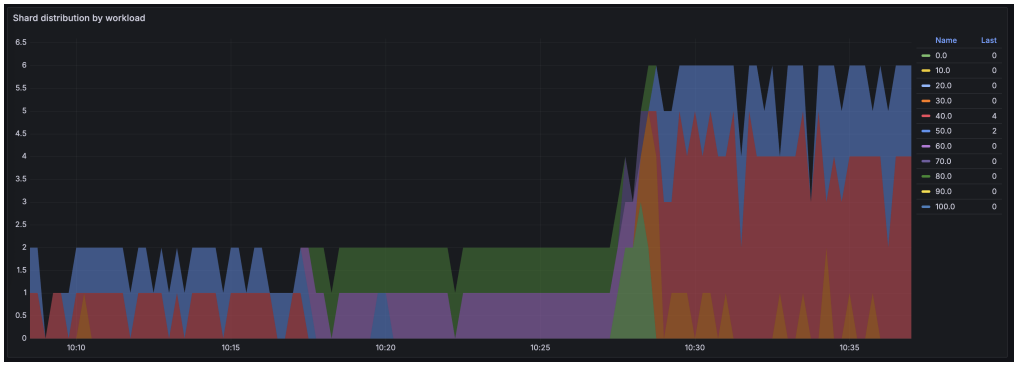
Нам необходимо увеличить лимит на максимальное количество партиций для таблицы `kv_test`:

1. В YDB UI выберите базу данных.
2. Откройте вкладку **Query**.
3. Выполните следующий запрос:

```
ALTER TABLE `kv_test` SET (
    AUTO_PARTITIONING_MAX_PARTITIONS_COUNT = 1000
);
```

Результат

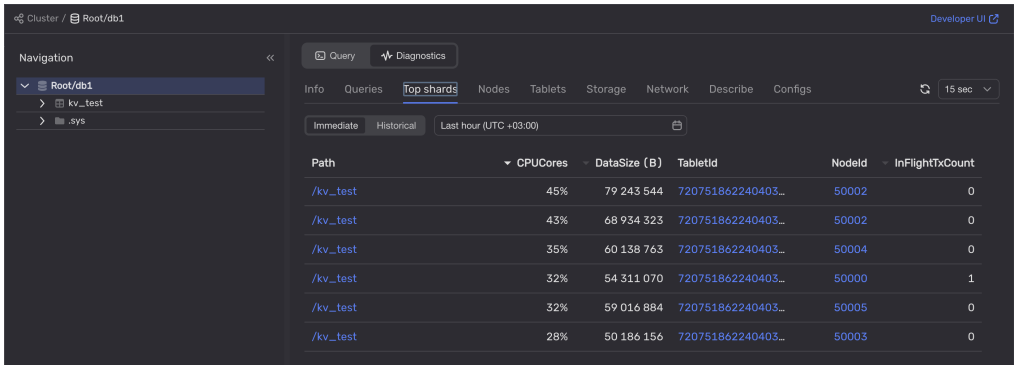
После увеличения максимального лимита на количество партиций для таблицы `kv_test` перегруженные data shard'ы начали делиться и их количество выросло до шести.



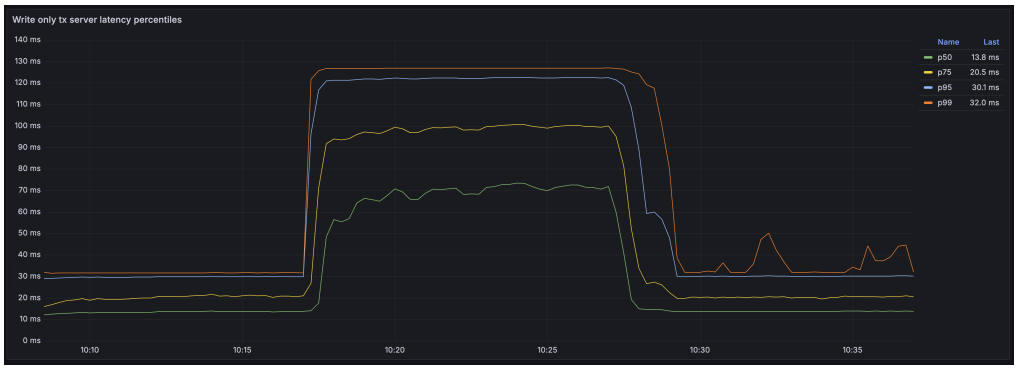
См. описание графика

График показывает, что количество data shard'ов выросло примерно в `10:29:00`. Судя по цвету групп, их нагрузка не превышает 50%.

Теперь шесть data shard'ов обрабатывают запросы к таблице `kv_test`, и ни один из них не перегружен:

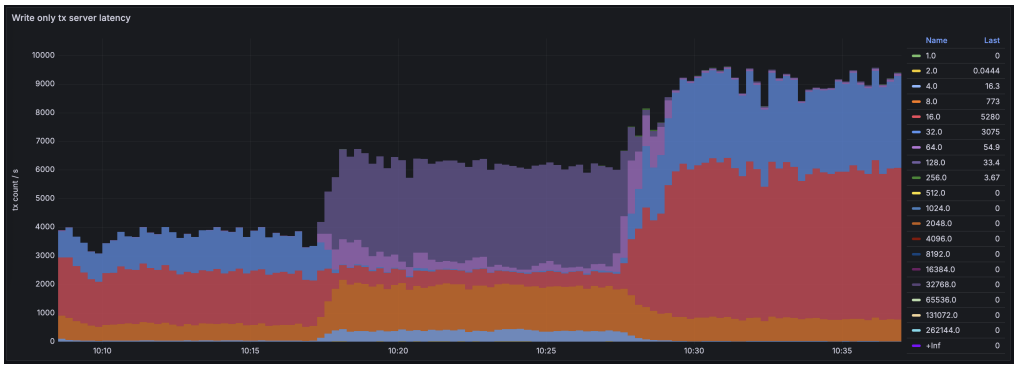


Давайте убедимся, что задержки транзакций вернулись к прежним значениям:



См. описание графика

Примерно в `10:29:00` процентили задержек упали практически до прежних значений.

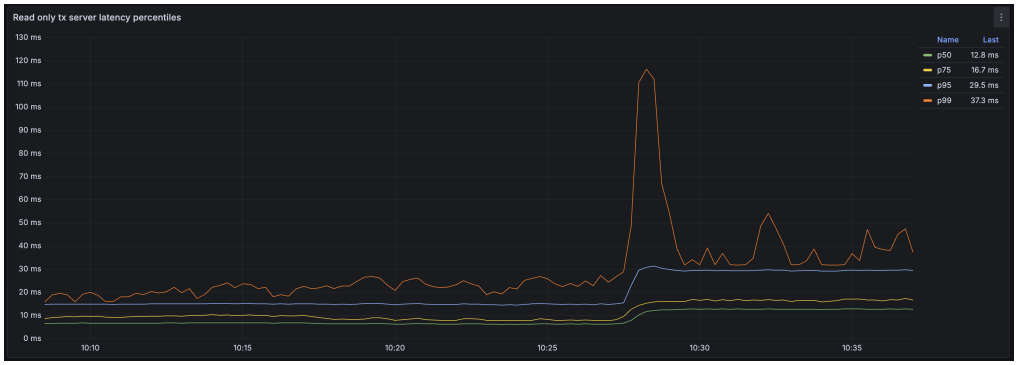


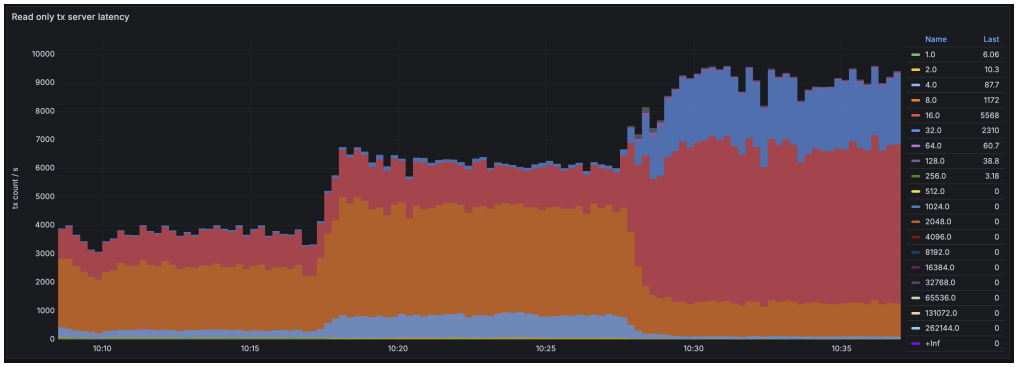
См. описание графика

Транзакции на этом графике теперь распределены по трём основным группам:

- `Группа 8.0` — примерно 8% транзакций выполняется с задержкой от 4 до 8 миллисекунд.
- `Группа 16.0` — примерно 58% транзакций выполняется с задержкой от 8 до 16 миллисекунд.
- `Группа 32.0` — примерно 34% транзакций выполняется с задержкой от 16 до 32 миллисекунд.

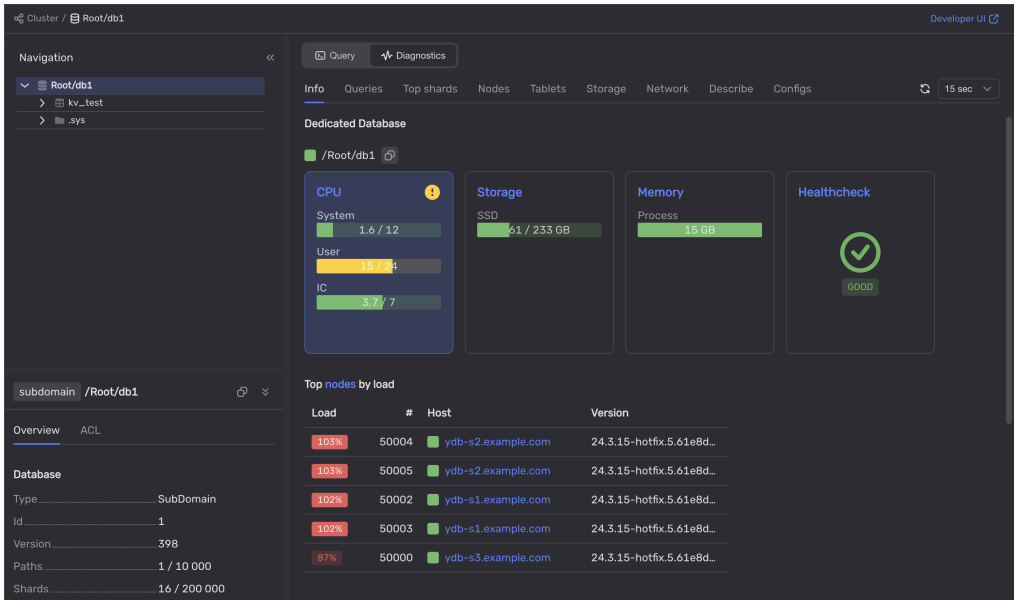
Размеры остальных групп незначительны.



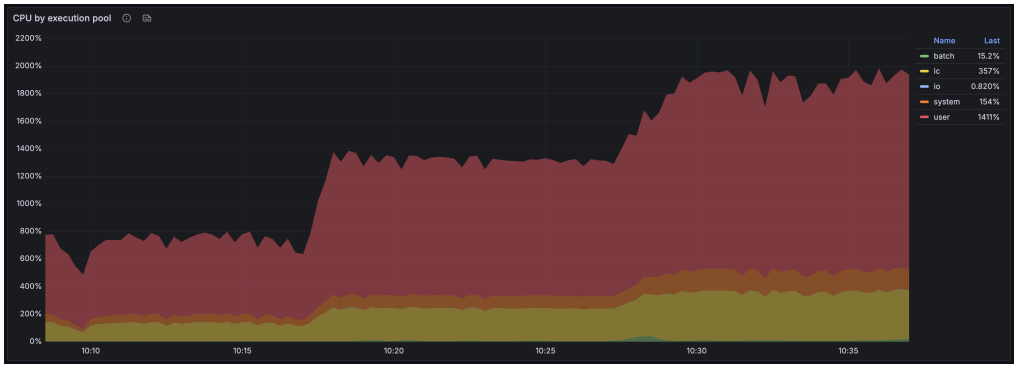


Проблема с недостаточным количеством ядер CPU

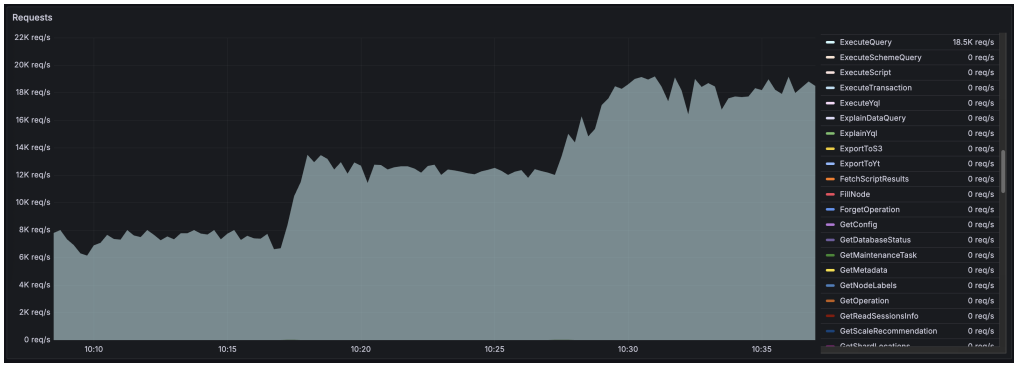
Однако, если мы откроем диагностику в YDB UI, то мы увидим предупреждение о том, что кластеру YDB не хватает ресурсов процессора в пользовательском пуле:



Загрузка CPU еще раз увеличилась, что видно на графике **CPU by execution pool**.



Количество пользовательских запросов выросло приблизительно с 13 000 до 18 000 в 10:30:00.



Решение проблемы с нехваткой ресурсов CPU

Проблему недостатка ресурсов CPU можно решить несколькими способами:

- Увеличить количество ядер для серверов, на которых запущены узлы YDB.
- Увеличить количество ядер, доступных акторной системе узлов кластера YDB, в конфигурации кластера.
- Добавить новые узлы в кластер YDB.

В нашем примере мы увеличим количество ядер CPU на серверах YDB и затем увеличим количество ядер в пуле ресурсов узлов кластера YDB на каждом сервере:

1. Увеличить количество ядер CPU выделенных серверам YDB. В нашем случае мы увеличили количество ядер с 16 до 24.
2. На каждом сервере YDB изменить динамическую конфигурацию кластера:
   2.1. Открыть файл конфигурации `/opt/ydb/cfg/ydbd-config-dynamic.yaml` в текстовом редакторе:

```
sudo vim /opt/ydb/cfg/ydbd-config-dynamic.yaml
```

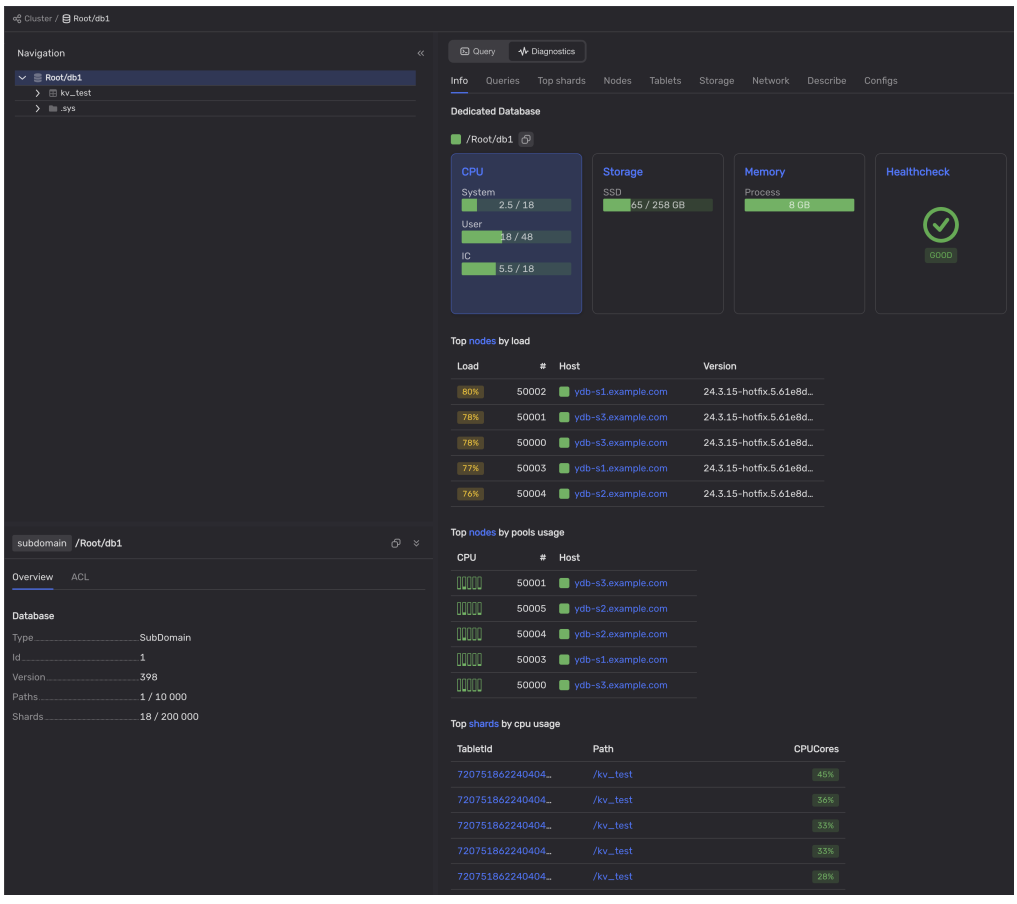
   2.2. Увеличить количество ядер в параметре `cpu_count` раздела `actor_system_config`:

```
actor_system_config:
    use_auto_config: true
    node_type: COMPUTE
cpu_count: 24
```

3. Чтобы применить новую конфигурацию, перезапустить узлы YDB с помощью утилиты ydbops, ansible или вручную.

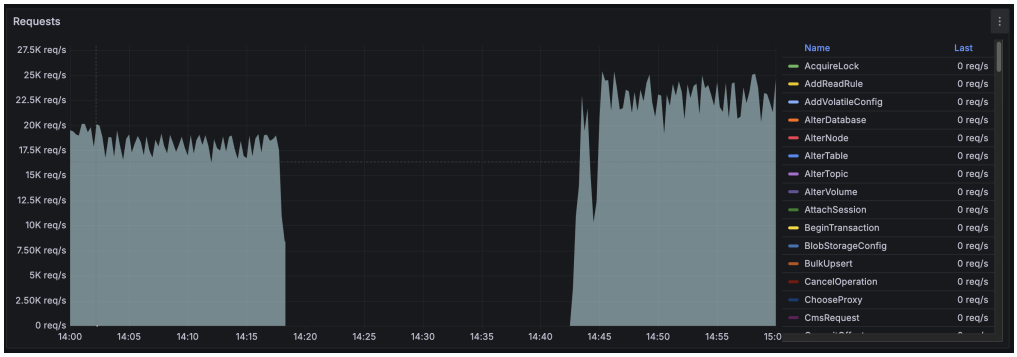
Результат

После увеличения количества ядер CPU в кластере YDB предупреждение о недостатке ресурсов CPU пропало:



А теперь давайте проверим производительность.

Количество обрабатываемых запросов увеличилось примерно до 14-15 тысяч в секунду.



Помогло ли добавление ядер CPU снизить задержки?



См. описание графика

На графике опять появилась `Группа 4.0` с задержками от 2 до 4 миллисекунд. `Группа 8.0` выросла в три раза. `Группа 16.0` изменилась незначительно. Практически пропала `Группа 32.0`.



См. описание графика

Процентили задержек на запись немного снизились, а также прекратились всплески p99.



См. описание графика

График задержек транзакций на чтение стал очень похож на график пишущих транзакций. Вернулась `Группа 4.0` с задержками от 2 до 4 миллисекунд, чья доля оказалась еще больше, чем при записи. `Группа 4.0` выросла в три раза. `Группа 16.0` изменилась незначительно. Практически пропала `Группа 32.0`.



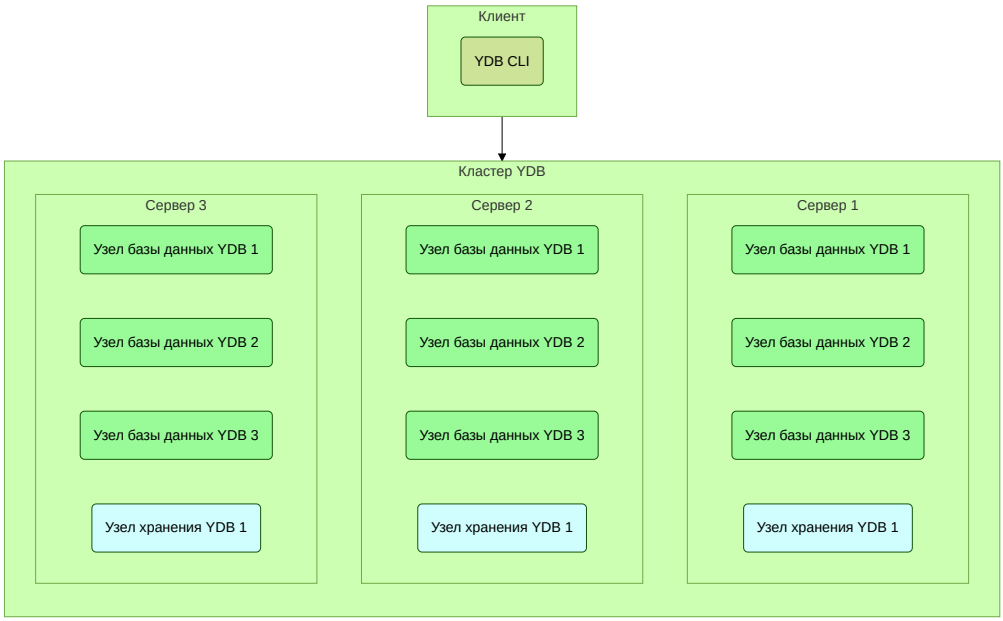
См. описание графика

Процентили задержек на чтение тоже снизились, а также практически исчезли всплески p99.

Тестовый стенд

Топология

Для этого примера мы использовали кластер YDB из трёх серверов на Ubuntu 22.04 LTS. На каждом сервере был запущен один узел хранения и три узла баз данных, обслуживающих одну и ту же базу данных.



Аппаратная конфигурация

Аппаратные ресурсы серверов (виртуальных машин) приведены ниже:

- Платформа: Intel Broadwell
- Гарантированный уровень производительности vCPU: 100%
- vCPU: 16
- Память: 32 GB
- Диски:
  - 3 × 93 GB SSD на каждом узле YDB
  - 20 GB HDD для операционной системы

Тест

Нагрузка на кластер YDB была запущена с помощью команды CLI `ydb workload`. Дополнительную информацию см. в статье Нагрузочное тестирование.

Чтобы воспроизвести нагрузку, выполните следующие шаги:

1. Проинициализируйте таблицы для нагрузочного тестирования:

   ```
   ydb workload kv init --min-partitions 1
   ydb yql -s 'ALTER TABLE `kv_test` SET (
       AUTO_PARTITIONING_MAX_PARTITIONS_COUNT = 2
   );'
   ```

   Мы намеренно ограничиваем максимальное количество партиций в таблице `kv_test` до 2 после инициализации.

2. Воспроизведите стандартную нагрузку на кластер YDB:

   ```
   ydb workload kv run mixed -s 600 -t 250 --rate 8000
   ```

   Мы запустили простую нагрузку, используя базу данных YDB как Key-Value хранилище. Точнее, мы использовали нагрузку `mixed`, которая одновременно пишет и читает данные, дополнительно проверяя что все записанные данные

успешно читаются.

Параметр `-t 250` используется для запуска нагрузочного тестирования в 250 потоков. Параметр `--rate 8000` используется для ограничение максимального количества запросов до 8000 запросов в секунду.

3. Создайте перегрузку на кластере YDB:

```
ydb workload kv run mixed -s 600 -t 400
```

Как только первый тест завершился, мы немедленно запустили тот же самый тест в 400 потоков без ограничения максимального количества запросов в секунду, чтобы создать перегрузку.

Смотрите также

- Диагностика проблем с производительностью
- Перегруженные таблетки data shard
- Строковые таблицы

Вопросы и ответы про YDB

- Общие
- SDK
- Ошибки
- YQL
- Аналитика
- Известные проблемы

Общие вопросы про YDB

Что такое YDB?

YDB — это распределенная отказоустойчивая Distributed SQL СУБД. YDB обеспечивает высокую доступность и масштабируемость, и, в то же время, строгую консистентность и поддержку ACID-транзакций. Для запросов используется диалект SQL (YQL).

YDB — это полностью управляемая база данных, для создания инстанса базы данных используется сервис управления базами данных YDB.

Какие возможности предоставляет YDB?

YDB обеспечивает высокую доступность и сохранность данных благодаря синхронной репликации в трех зонах доступности. Также YDB следит за равномерным распределением нагрузки на доступные аппаратные ресурсы. Это значит, что не нужно заказывать ресурсы, YDB автоматически выделяет и освобождает ресурсы, исходя из пользовательской нагрузки.

Какая модель консистентности используется в YDB?

Для чтения данных YDB использует модель строгой консистентности (непротиворечивости) данных.

Как проектировать первичный ключ?

Чтобы правильно спроектировать первичный ключ, стоит придерживаться следующих правил:

- Необходимо избегать ситуаций, когда основная часть нагрузки приходится на одну партицию таблицы. При равномерной нагрузке проще достигается высокая общая производительность.

  Как следствие этого правила, не стоит в качестве первичного ключа таблицы использовать монотонно возрастающую последовательность, например, timestamp.
- Чем меньше партиций таблиц будет задействовано во время исполнения запроса, тем быстрее он будет исполнен. Чтобы достичь большей производительности стоит руководствоваться правилом: один запрос — одна партиция.
- Необходимо избегать ситуаций, при которых какая-то малая часть БД нагружена существенно больше, чем остальные части БД.

Подробнее читайте в разделе Выбор первичного ключа.

Как равномерно распределить нагрузку по партициям таблицы?

Для равномерного распределения нагрузки по партициям таблиц и увеличения общей производительности БД, можно использовать следующие техники:

- Для избежания использования монотонно возрастающих значений первичного ключа можно:
  - изменить порядок следования его компонентов;
  - использовать хеш от значений ключевых колонок в качестве первичного ключа.
- Необходимо уменьшать количество партиций таблиц, задействованных в одном запросе.

Подробнее читайте в разделе Выбор первичного ключа.

Можно ли использовать NULL в ключевой колонке?

В YDB все колонки, включая ключевые, могут содержать значение `NULL`, однако использовать `NULL` в качестве значений в ключевых колонках не рекомендуется.

По стандарту языка SQL (ISO/IEC 9075) значение `NULL` нельзя сравнивать с другими значениями, вследствие чего использование лаконичных конструкций SQL с простыми операторами сравнения может приводить, например, к тому, что строки, содержащие значение NULL, могут быть пропущены при фильтрации.

Существует ли оптимальный размер строки базы данных?

Для достижения высокой производительности не рекомендуется записывать в БД строки размером более 8 МБ и ключевые колонки размером более 2 КБ.

Подробнее существующие ограничения описаны в разделе Ограничения базы данных.

Как работают вторичные индексы в YDB?

Вторичные индексы в YDB — глобальные и могут быть неуникальными.

Подробнее читайте в разделе Вторичные индексы.

Как осуществляется постраничный вывод?

Для организации постраничного вывода рекомендуется последовательно выбирать данные, отсортированные по первичному ключу, ограничивая количество строк ключевым словом `LIMIT`. Использование ключевого слова `OFFSET` для решения этой задачи крайне не желательно.

Подробнее читайте в разделе Постраничный вывод.

Как удалять устаревшие данные?

Для эффективного удаления устаревших данных рекомендуется использовать TTL.

Как происходит синхронизация данных между дата-центрами в геораспределенных кластерах

Таблетка-лидер записывает данные в распределенное сетевое хранилище, которое сохраняет копии в нескольких дата-центрах. YDB подтверждает пользователю запрос только после успешного сохранения нужного числа копий в нужном числе дата-центров.

SDK

Что делать, если SDK экстренно завершает работу при завершении работы приложения?

Не следует оборачивать компоненты SDK в синглтон, так как время их жизни не должно превышать время выполнения функции `main()`. Так как при уничтожении клиента происходит опустошение пулов сессий, требуется возможность перехода по сети. Но gPRC содержит глобальные статические переменные, которые к этому времени уже могут быть разрушены. Таким образом, работа gRPC невозможна. При необходимости сделать драйвер глобальным объектом, на драйвере нужно вызвать функцию `Stop(true)` перед выходом из `main()`.

Что делать, если при вызове `fork()` в дочернем процессе программа работает некорректно?

Вызов `fork()` в многопоточных приложениях является антипаттерном. Так как и SDK, и gRPC-библиотека являются многопоточными приложениями, стабильность их работы не гарантируется.

Что делать, если я получаю ошибку «Active sessions limit exceeded», хотя текущее количество активных сессий не превышает лимит?

Лимит действует на количество активных сессий. Активная сессия — сессия, переданная клиенту для использования в его коде. Сессия возвращается в пул в деструкторе. При этом, сама сессия является копируемым объектом. Возможно, вы сохранили в коде копию сессии.

Есть ли возможность делать запросы к разным БД из одного приложения?

Да, C++ SDK позволяет переопределять параметры БД и токен в момент создания клиента. Отдельные драйверы создавать не нужно.

Что делать, если отказала одна из ВМ и невозможно сделать запрос?

Чтобы выявить недоступность ВМ, нужно проставить клиентский таймаут. Все запросы содержат параметры для клиентского таймаута. Величина таймаута должна быть на порядок больше ожидаемого времени выполнения запроса.

Ошибки

Почему может возникать ошибка «Status: OVERLOADED Error: Pending previous query completion» в C++ SDK?

Q: При запуске двух запросов, пытаюсь получить ответ из фьючера от второго из них. Получаю: `Status: OVERLOADED Why: <main>: Error: Pending previous query completion`.

A: Сессия в SDK однопоточная. Если нужно параллельно использовать несколько запросов, нужно создавать несколько сессий.

Что делать, если я часто получаю ошибку «Transaction locks invalidated»?

Обычно при получении этой ошибки следует повторить выполнение транзакции, так как YDB использует оптимистичные блокировки. Если такая ошибка возникает часто, это означает, что в транзакции читается большое количество строк или, что много транзакций конкурируют за одни и те же «горячие» строки. Имеет смысл просмотреть запросы, выполняющиеся в транзакции и определить, не читают ли они лишние строки.

Почему возникает ошибка «Exceeded maximum allowed number of active transactions»?

В логике на клиентской стороне надо стараться держать как можно более короткие транзакции.

В рамках сессии разрешено не более 10 активных транзакций. При старте транзакции нужно использовать либо коммит-флаг для автокоммита, либо явный коммит/роллбек.

Что делать, если при запросе я получаю ошибку «Datashard: Reply size limit exceeded»?

Данная ошибка означает, что при выполнении запроса с одного из участвующих в нем даташардов была попытка вернуть более 50 МБ данных, что превышает допустимый лимит.

Рекомендации:

- Общая рекомендация — уменьшить объем данных, обрабатываемых в транзакции.
- Если выполняется операция `Join`, стоит убедиться, что она выполняется способом Index lookup Join.
- Если осуществляется простая выборка, стоит убедиться, что она делается по ключам, или добавить `LIMIT` в запрос.

Что делать, если при запросе я получаю ошибку «Datashard program size limit exceeded»?

Данная ошибка означает, что размер программы (в сумме со значениями параметров) для одного из даташардов превысил лимит в 50 МБ. Чаще всего это означает попытку записать в таблицы базы более 50 МБ данных в одной транзакции. В качестве записей учитываются все модифицирующие операции в транзакции, такие как `UPSERT`, `REPLACE`, `INSERT`, `UPDATE`.

Необходимо уменьшить суммарных объем записей в одной транзакции. В общем случае не рекомендуется объединять в единую транзакцию запросы, которые логически не требуют транзакционности. В случае добавления/обновления данных батчами, рекомендуется уменьшить размер одного батча до значений, не превышающих нескольких мегабайт.

YQL

Общие вопросы

Как выбрать из таблицы строчки по заданному списку ключей?

Выборка строчек таблицы по заданному списку значений первичного ключа (или префикса ключа) таблицы выполняется с помощью оператора `IN`:

```
DECLARE $keys AS List<UInt64>;

SELECT * FROM some_table
WHERE Key1 IN $keys;
```

В случае если выборка делается по составному ключу, параметр запроса должен иметь тип списка таплов:

```
DECLARE $keys AS List<Tuple<UInt64, String>>;

SELECT * FROM some_table
WHERE (Key1, Key2) IN $keys;
```

Для эффективного выполнения выборки важно, чтобы типы значений в параметрах совпадали с типами ключевых колонок в таблице.

Осуществляется ли поиск по индексу для условий, содержащих оператор LIKE?

Оператор `LIKE` может быть использован для поиска по индексу таблицы только в случае, если он задает префикс строки:

```
SELECT * FROM string_key_table
WHERE Key LIKE "some_prefix%";
```

Почему в результате запроса выводится только 1000 строк?

1000 строк — ограничение на размер одного результата для YQL запроса. В случае если результат запроса был обрезан, он будет помечен флагом `Truncated`. Чтобы получить большее количество строк из таблицы, можно воспользоваться постраничным выводом или операцией `ReadTable`.

Как экранировать кавычки JSON-строк при добавлении их в таблицу?

Рассмотрим пример с двумя возможными вариантами добавления JSON-строки в таблицу:

```
UPSERT INTO test_json(id, json_string)
VALUES
    (1, Json(@@[{"name":"Peter \"strong cat\" Kourbatov"}]@@)),
    (2, Json('[{"name":"Peter \\\"strong cat\\\" Kourbatov"}]'))
;
```

Для вставки значения в первой строке используется `raw string` и способ экранирования с помощью `\"`. Для вставки второй строки используется экранирование через `\\\"`.

Мы рекомендуем применять `raw string` и способ экранирования с помощью `\"`, так как он более нагляден.

Как обновить JSON в таблице?

Частичное обновление JSON-поля не поддерживается, поэтому необходимо сконструировать новое значение из частей старого и новых данных. Для этого используется `Yson` UDF в комбинации с функциями для работы со словарями.

В примере ниже к JSON-объекту из колонки `column` добавляется новое поле `new_field`:

```
UPDATE table
SET column = Yson::SerializeJson(
    Yson::From(
        SetUnion(
            Yson::ConvertTo(Yson::ParseJson(column), Dict<String, Yson>),
            {'new_field': Yson('"value"')},
            ($K, $v1, $v2) -> { RETURN COALESCE($v2, $v1); }
        )
    )
)
WHERE id = 1;
```

Как обновить только те значения, ключей которых нет в таблице?

Можно использовать операцию `LEFT JOIN`, чтобы пометить отсутствующие в таблице ключи, после чего обновить их значения:

```
DECLARE $values AS List<Struct<Key: UInt64, Value: String>>;

UPSERT INTO kv_table
SELECT v.Key AS Key, v.Value AS Value
FROM AS_TABLE($values) AS v
LEFT JOIN kv_table AS t
ON v.Key = t.Key
WHERE t.Key IS NULL;
```

Операции Join

Есть ли особенности в работе операции Join?

Операция `Join` в YDB выполняется одним из двух способов:

- Common Join;
- Index Lookup Join.

Common Join

Содержимое обеих таблиц (левая и правая части `Join`) отправляется на выполняющий запрос узел, где операция выполняется над данными целиком. Это универсальный способ выполнения операции `Join`, который применяется в случае, когда более оптимальные способы неприменимы. Для больших таблиц такой способ или работает медленно, или в общем случае не работает из-за превышения лимитов на пересылку данных.

Index lookup Join

Для строчек из левой части операции `Join` осуществляется поиск (lookup) соответствующих значений в правой части. Данный способ применяется, когда правая часть является таблицей и ключ операции `Join` является префиксом ее первичного ключа или ключа вторичного индекса. При использовании данного способа из правой таблицы делаются ограниченные выборки, вместо полного чтения, что позволяет использовать его при работе с большими таблицами.

> **Примечание**
>
> Для большинства OLTP запросов рекомендуется использовать Index Lookup Join с небольшим размером левой части. Такие операции читают мало данных и могут быть выполнены эффективно.

Как сделать Join с данными из параметров запроса?

Данные из параметров запроса можно использовать как константную таблицу. Для этого нужно использовать модификатор `AS_TABLE` с параметром, имеющим тип списка структур:

```
DECLARE $data AS List<Struct<Key1: UInt64, Key2: String>>;

SELECT * FROM AS_TABLE($data) AS d
INNER JOIN some_table AS t
ON t.Key1 = d.Key1 AND t.Key2 = d.Key2;
```

Явного ограничения на количество записей в константной таблице нет, но нужно иметь в виду стандартное ограничение на общий размер параметров запроса (50 МБ).

Как лучше реализовать запрос вида (key1, key2) IN ((v1, v2), (v3, v4), ...)?

Для выборки по набору ключей (в том числе составных) используется синтаксис `WHERE ... IN`:

```
DECLARE $key_pairs AS List<Tuple<Uint64, Uint64>>;

SELECT * FROM some_table
WHERE (Key1, Key2) IN $key_pairs;
```

Транзакции

Насколько эффективно выполнение нескольких запросов в транзакции?

При последовательном выполнении нескольких запросов, суммарный latency для транзакции может быть больше, чем при выполнении тех же операций в рамках одного запроса. В первую очередь это обусловлено дополнительными сетевыми задержками на выполнение каждого из запросов. Поэтому в случае, когда для транзакции не требуется интерактивность, рекомендуется формулировать все операции в одном YQL запросе.

Является ли отдельный запрос атомарным?

В общем случае YQL-запросы могут выполняться в несколько последовательных фаз. Например, запрос с операцией Join может быть выполнен в две фазы, читающие данные левой и правой таблицы соответственно. Этот аспект является важным при выполнении запроса в транзакции с низким уровнем изоляции (`online_read_only`), так как в этом случае данные между фазами выполнения могут быть изменены другими транзакциями.

Вопросы и ответы об аналитике в YDB

Можно ли использовать YDB для аналитических нагрузок (OLAP)?

Да, можно. Если это основной тип нагрузки для конкретной таблицы, то убедитесь, что она колоночная.

Как выбирать между строчными и колоночными таблицами?

Аналогично выбору между транзакционными (OLTP) и аналитическими (OLAP) системами управления базами данных, этот вопрос сводится к ряду компромиссов, которые необходимо учитывать:

- **Каков основной сценарий использования таблицы?** Для преимущественно транзакционных (OLTP) нагрузок используйте строчные таблицы. Для аналитических нагрузок (OLAP) используйте колоночные таблицы. Транзакционные нагрузки характеризуются высокой частотой запросов, каждый из которых затрагивает небольшое количество строк. Аналитические нагрузки связаны с обработкой больших объёмов данных для получения относительно небольших результатов запросов.
- **Как модифицируется таблица?** Как правило, строчные таблицы лучше работают, когда данные часто изменяются на месте, тогда как колоночные таблицы лучше подходят, когда данные в основном добавляются в виде новых строк. Таким образом, строчные таблицы обычно отражают текущее состояние набора данных, тогда как колоночные часто хранят историю каких-то неизменяемых событий.
- **Какие функциональные возможности нужны?** Хоть YDB и стремится к функциональному паритету между строчными и колоночными таблицами, могут быть текущие ограничения, которые следует учитывать. Для получения подробностей ознакомьтесь с документацией о конкретной функциональности, которую планируется использовать с данной таблицей.

В отличие от большинства других систем управления базами данных, YDB поддерживает как строчные, так и колоночные таблицы в одной и той же базе данных. Однако имейте в виду, что транзакционные и аналитические нагрузки имеют разные паттерны потребления ресурсов и могут влиять друг на друга при перегруженном кластере.

Известные проблемы

Не происходит объединение партиций таблицы при включённом автопартицировании

Для таблиц, созданных в ранних версиях YDB, механизм автоматического партицирования может не объединять партиции при достижении условий для объединения по нагрузке или по размеру. Это может привести к наличию избыточного количества партиций в таблице.

Проблема специфична для таблиц, соответствующих двум следующим критериям:

- Таблица была создана на кластере YDB до версии 22.2.
- На момент создания таблицы или впоследствии для неё не было задано значение минимального количества партиций.

Для устранения проблемы нужно явно задать для таблицы значение минимального количества партиций, например, с помощью параметра AUTO_PARTITIONING_MIN_PARTITIONS_COUNT:

```
ALTER TABLE `my_table` SET (AUTO_PARTITIONING_MIN_PARTITIONS_COUNT = 1);
```

Все вопросы на одной странице

Общие вопросы

Что такое YDB?

YDB — это распределенная отказоустойчивая Distributed SQL СУБД. YDB обеспечивает высокую доступность и масштабируемость, и, в то же время, строгую консистентность и поддержку ACID-транзакций. Для запросов используется диалект SQL (YQL).

YDB — это полностью управляемая база данных, для создания инстанса базы данных используется сервис управления базами данных YDB.

Какие возможности предоставляет YDB?

YDB обеспечивает высокую доступность и сохранность данных благодаря синхронной репликации в трех зонах доступности. Также YDB следит за равномерным распределением нагрузки на доступные аппаратные ресурсы. Это значит, что не нужно заказывать ресурсы, YDB автоматически выделяет и освобождает ресурсы, исходя из пользовательской нагрузки.

Какая модель консистентности используется в YDB?

Для чтения данных YDB использует модель строгой консистентности (непротиворечивости) данных.

Как проектировать первичный ключ?

Чтобы правильно спроектировать первичный ключ, стоит придерживаться следующих правил:

- Необходимо избегать ситуаций, когда основная часть нагрузки приходится на одну партицию таблицы. При равномерной нагрузке проще достигается высокая общая производительность.

  Как следствие этого правила, не стоит в качестве первичного ключа таблицы использовать монотонно возрастающую последовательность, например, timestamp.
- Чем меньше партиций таблиц будет задействовано во время исполнения запроса, тем быстрее он будет исполнен. Чтобы достичь большей производительности стоит руководствоваться правилом: один запрос — одна партиция.
- Необходимо избегать ситуаций, при которых какая-то малая часть БД нагружена существенно больше, чем остальные части БД.

Подробнее читайте в разделе Выбор первичного ключа.

Как равномерно распределить нагрузку по партициям таблицы?

Для равномерного распределения нагрузки по партициям таблиц и увеличения общей производительности БД, можно использовать следующие техники:

- Для избежания использования монотонно возрастающих значений первичного ключа можно:
  - изменить порядок следования его компонентов;
  - использовать хеш от значений ключевых колонок в качестве первичного ключа.
- Необходимо уменьшать количество партиций таблиц, задействованных в одном запросе.

Подробнее читайте в разделе Выбор первичного ключа.

Можно ли использовать NULL в ключевой колонке?

В YDB все колонки, включая ключевые, могут содержать значение `NULL`, однако использовать `NULL` в качестве значений в ключевых колонках не рекомендуется.

По стандарту языка SQL (ISO/IEC 9075) значение `NULL` нельзя сравнивать с другими значениями, вследствие чего использование лаконичных конструкций SQL с простыми операторами сравнения может приводить, например, к тому, что строки, содержащие значение NULL, могут быть пропущены при фильтрации.

Существует ли оптимальный размер строки базы данных?

Для достижения высокой производительности не рекомендуется записывать в БД строки размером более 8 МБ и ключевые колонки размером более 2 КБ.

Подробнее существующие ограничения описаны в разделе Ограничения базы данных.

Как работают вторичные индексы в YDB?

Вторичные индексы в YDB — глобальные и могут быть неуникальными.

Подробнее читайте в разделе Вторичные индексы.

Как осуществляется постраничный вывод?

Для организации постраничного вывода рекомендуется последовательно выбирать данные, отсортированные по первичному ключу, ограничивая количество строк ключевым словом `LIMIT`. Использование ключевого слова `OFFSET` для решения этой задачи крайне не желательно.

Подробнее читайте в разделе Постраничный вывод.

Как удалять устаревшие данные?

Для эффективного удаления устаревших данных рекомендуется использовать TTL.

Как происходит синхронизация данных между дата-центрами в геораспределенных кластерах

Таблетка-лидер записывает данные в распределенное сетевое хранилище, которое сохраняет копии в нескольких дата-центрах. YDB подтверждает пользователю запрос только после успешного сохранения нужного числа копий в нужном числе дата-центров.

SDK

Что делать, если SDK экстренно завершает работу при завершении работы приложения?

Не следует оборачивать компоненты SDK в синглтон, так как время их жизни не должно превышать время выполнения функции `main()`. Так как при уничтожении клиента происходит опустошение пулов сессий, требуется возможность перехода по сети. Но gPRC содержит глобальные статические переменные, которые к этому времени уже могут быть разрушены. Таким образом, работа gRPC невозможна. При необходимости сделать драйвер глобальным объектом, на драйвере нужно вызвать функцию `Stop(true)` перед выходом из `main()`.

Что делать, если при вызове `fork()` в дочернем процессе программа работает некорректно?

Вызов `fork()` в многопоточных приложениях является антипаттерном. Так как и SDK, и gRPC-библиотека являются многопоточными приложениями, стабильность их работы не гарантируется.

Что делать, если я получаю ошибку «Active sessions limit exceeded», хотя текущее количество активных сессий не превышает лимит?

Лимит действует на количество активных сессий. Активная сессия — сессия, переданная клиенту для использования в его коде. Сессия возвращается в пул в деструкторе. При этом, сама сессия является копируемым объектом. Возможно, вы сохранили в коде копию сессии.

Есть ли возможность делать запросы к разным БД из одного приложения?

Да, C++ SDK позволяет переопределять параметры БД и токен в момент создания клиента. Отдельные драйверы создавать не нужно.

Что делать, если отказала одна из ВМ и невозможно сделать запрос?

Чтобы выявить недоступность ВМ, нужно проставить клиентский таймаут. Все запросы содержат параметры для клиентского таймаута. Величина таймаута должна быть на порядок больше ожидаемого времени выполнения запроса.

Ошибки

Почему может возникать ошибка «Status: OVERLOADED Error: Pending previous query completion» в C++ SDK?

Q: При запуске двух запросов, пытаюсь получить ответ из фьючера от второго из них. Получаю: `Status: OVERLOADED Why: <main>: Error: Pending previous query completion`.

A: Сессия в SDK однопоточная. Если нужно параллельно использовать несколько запросов, нужно создавать несколько сессий.

Что делать, если я часто получаю ошибку «Transaction locks invalidated»?

Обычно при получении этой ошибки следует повторить выполнение транзакции, так как YDB использует оптимистичные блокировки. Если такая ошибка возникает часто, это означает, что в транзакции читается большое количество строк или, что много транзакций конкурируют за одни и те же «горячие» строки. Имеет смысл просмотреть запросы, выполняющиеся в транзакции и определить, не читают ли они лишние строки.

Почему возникает ошибка «Exceeded maximum allowed number of active transactions»?

В логике на клиентской стороне надо стараться держать как можно более короткие транзакции.

В рамках сессии разрешено не более 10 активных транзакций. При старте транзакции нужно использовать либо коммит-флаг для автокоммита, либо явный коммит/роллбек.

Что делать, если при запросе я получаю ошибку «Datashard: Reply size limit exceeded»?

Данная ошибка означает, что при выполнении запроса с одного из участвующих в нем даташардов была попытка вернуть более 50 МБ данных, что превышает допустимый лимит.

Рекомендации:

- Общая рекомендация — уменьшить объем данных, обрабатываемых в транзакции.
- Если выполняется операция `Join`, стоит убедиться, что она выполняется способом Index lookup Join.
- Если осуществляется простая выборка, стоит убедиться, что она делается по ключам, или добавить `LIMIT` в запрос.

Что делать, если при запросе я получаю ошибку «Datashard program size limit exceeded»?

Данная ошибка означает, что размер программы (в сумме со значениями параметров) для одного из даташардов превысил лимит в 50 МБ. Чаще всего это означает попытку записать в таблицы базы более 50 МБ данных в одной транзакции. В качестве записей учитываются все модифицирующие операции в транзакции, такие как `UPSERT`, `REPLACE`, `INSERT`, `UPDATE`.

Необходимо уменьшить суммарных объем записей в одной транзакции. В общем случае не рекомендуется объединять в единую транзакцию запросы, которые логически не требуют транзакционности. В случае добавления/обновления данных батчами, рекомендуется уменьшить размер одного батча до значений, не превышающих нескольких мегабайт.

YQL

Общие вопросы

Как выбрать из таблицы строчки по заданному списку ключей?

Выборка строчек таблицы по заданному списку значений первичного ключа (или префикса ключа) таблицы выполняется с помощью оператора `IN`:

```
DECLARE $keys AS List<UInt64>;

SELECT * FROM some_table
WHERE Key1 IN $keys;
```

В случае если выборка делается по составному ключу, параметр запроса должен иметь тип списка таплов:

```
DECLARE $keys AS List<Tuple<UInt64, String>>;

SELECT * FROM some_table
WHERE (Key1, Key2) IN $keys;
```

Для эффективного выполнения выборки важно, чтобы типы значений в параметрах совпадали с типами ключевых колонок в таблице.

Осуществляется ли поиск по индексу для условий, содержащих оператор LIKE?

Оператор `LIKE` может быть использован для поиска по индексу таблицы только в случае, если он задает префикс строки:

```
SELECT * FROM string_key_table
WHERE Key LIKE "some_prefix%";
```

Почему в результате запроса выводится только 1000 строк?

1000 строк — ограничение на размер одного результата для YQL запроса. В случае если результат запроса был обрезан, он будет помечен флагом `Truncated`. Чтобы получить большее количество строк из таблицы, можно воспользоваться постраничным выводом или операцией `ReadTable`.

Как экранировать кавычки JSON-строк при добавлении их в таблицу?

Рассмотрим пример с двумя возможными вариантами добавления JSON-строки в таблицу:

```
UPSERT INTO test_json(id, json_string)
VALUES
    (1, Json(@@[{"name":"Peter \"strong cat\" Kourbatov"}]@@)),
    (2, Json('[{"name":"Peter \\\"strong cat\\\" Kourbatov"}]'))
;
```

Для вставки значения в первой строке используется `raw string` и способ экранирования с помощью `\"`. Для вставки второй строки используется экранирование через `\\\"`.

Мы рекомендуем применять `raw string` и способ экранирования с помощью `\"`, так как он более нагляден.

Как обновить JSON в таблице?

Частичное обновление JSON-поля не поддерживается, поэтому необходимо сконструировать новое значение из частей старого и новых данных. Для этого используется `Yson` UDF в комбинации с функциями для работы со словарями.

В примере ниже к JSON-объекту из колонки `column` добавляется новое поле `new_field`:

```
UPDATE table
SET column = Yson::SerializeJson(
    Yson::From(
        SetUnion(
            Yson::ConvertTo(Yson::ParseJson(column), Dict<String, Yson>),
            {'new_field': Yson('"value"')},
            ($K, $v1, $v2) -> { RETURN COALESCE($v2, $v1); }
        )
    )
)
WHERE id = 1;
```

Как обновить только те значения, ключей которых нет в таблице?

Можно использовать операцию `LEFT JOIN`, чтобы пометить отсутствующие в таблице ключи, после чего обновить их значения:

```
DECLARE $values AS List<Struct<Key: UInt64, Value: String>>;

UPSERT INTO kv_table
SELECT v.Key AS Key, v.Value AS Value
FROM AS_TABLE($values) AS v
LEFT JOIN kv_table AS t
ON v.Key = t.Key
WHERE t.Key IS NULL;
```

Операции Join

Есть ли особенности в работе операции Join?

Операция `Join` в YDB выполняется одним из двух способов:

- Common Join;
- Index Lookup Join.

Common Join

Содержимое обеих таблиц (левая и правая части `Join`) отправляется на выполняющий запрос узел, где операция выполняется над данными целиком. Это универсальный способ выполнения операции `Join`, который применяется в случае, когда более оптимальные способы неприменимы. Для больших таблиц такой способ или работает медленно, или в общем случае не работает из-за превышения лимитов на пересылку данных.

Index lookup Join

Для строчек из левой части операции `Join` осуществляется поиск (lookup) соответствующих значений в правой части. Данный способ применяется, когда правая часть является таблицей и ключ операции `Join` является префиксом ее первичного ключа или ключа вторичного индекса. При использовании данного способа из правой таблицы делаются ограниченные выборки, вместо полного чтения, что позволяет использовать его при работе с большими таблицами.

> **Примечание**
>
> Для большинства OLTP запросов рекомендуется использовать Index Lookup Join с небольшим размером левой части. Такие операции читают мало данных и могут быть выполнены эффективно.

Как сделать Join с данными из параметров запроса?

Данные из параметров запроса можно использовать как константную таблицу. Для этого нужно использовать модификатор `AS_TABLE` с параметром, имеющим тип списка структур:

```
DECLARE $data AS List<Struct<Key1: UInt64, Key2: String>>;

SELECT * FROM AS_TABLE($data) AS d
INNER JOIN some_table AS t
ON t.Key1 = d.Key1 AND t.Key2 = d.Key2;
```

Явного ограничения на количество записей в константной таблице нет, но нужно иметь в виду стандартное ограничение на общий размер параметров запроса (50 МБ).

Как лучше реализовать запрос вида (key1, key2) IN ((v1, v2), (v3, v4), ...)?

Для выборки по набору ключей (в том числе составных) используется синтаксис `WHERE ... IN`:

```
DECLARE $key_pairs AS List<Tuple<Uint64, Uint64>>;

SELECT * FROM some_table
WHERE (Key1, Key2) IN $key_pairs;
```

Транзакции

Насколько эффективно выполнение нескольких запросов в транзакции?

При последовательном выполнении нескольких запросов, суммарный latency для транзакции может быть больше, чем при выполнении тех же операций в рамках одного запроса. В первую очередь это обусловлено дополнительными сетевыми задержками на выполнение каждого из запросов. Поэтому в случае, когда для транзакции не требуется интерактивность, рекомендуется формулировать все операции в одном YQL запросе.

Является ли отдельный запрос атомарным?

В общем случае YQL-запросы могут выполняться в несколько последовательных фаз. Например, запрос с операцией Join может быть выполнен в две фазы, читающие данные левой и правой таблицы соответственно. Этот аспект является важным при выполнении запроса в транзакции с низким уровнем изоляции (`online_read_only`), так как в этом случае данные между фазами выполнения могут быть изменены другими транзакциями.

Аналитика

Можно ли использовать YDB для аналитических нагрузок (OLAP)?

Да, можно. Если это основной тип нагрузки для конкретной таблицы, то убедитесь, что она колоночная.

Как выбирать между строчными и колоночными таблицами?

Аналогично выбору между транзакционными (OLTP) и аналитическими (OLAP) системами управления базами данных, этот вопрос сводится к ряду компромиссов, которые необходимо учитывать:

- **Каков основной сценарий использования таблицы?** Для преимущественно транзакционных (OLTP) нагрузок используйте строчные таблицы. Для аналитических нагрузок (OLAP) используйте колоночные таблицы. Транзакционные нагрузки характеризуются высокой частотой запросов, каждый из которых затрагивает небольшое количество строк. Аналитические нагрузки связаны с обработкой больших объёмов данных для получения относительно небольших результатов запросов.
- **Как модифицируется таблица?** Как правило, строчные таблицы лучше работают, когда данные часто изменяются на месте, тогда как колоночные таблицы лучше подходят, когда данные в основном добавляются в виде новых строк. Таким образом, строчные таблицы обычно отражают текущее состояние набора данных, тогда как колоночные часто хранят историю каких-то неизменяемых событий.
- **Какие функциональные возможности нужны?** Хоть YDB и стремится к функциональному паритету между строчными и колоночными таблицами, могут быть текущие ограничения, которые следует учитывать. Для получения подробностей ознакомьтесь с документацией о конкретной функциональности, которую планируется использовать с данной таблицей.

В отличие от большинства других систем управления базами данных, YDB поддерживает как строчные, так и колоночные таблицы в одной и той же базе данных. Однако имейте в виду, что транзакционные и аналитические нагрузки имеют разные паттерны потребления ресурсов и могут влиять друг на друга при перегруженном кластере.

Известные проблемы

Не происходит объединение партиций таблицы при включённом автопартицировании

Для таблиц, созданных в ранних версиях YDB, механизм автоматического партицирования может не объединять партиции при достижении условий для объединения по нагрузке или по размеру. Это может привести к наличию избыточного количества партиций в таблице.

Проблема специфична для таблиц, соответствующих двум следующим критериям:

- Таблица была создана на кластере YDB до версии 22.2.

- На момент создания таблицы или впоследствии для неё не было задано значение минимального количества партиций.

Для устранения проблемы нужно явно задать для таблицы значение минимального количества партиций, например, с помощью параметра AUTO_PARTITIONING_MIN_PARTITIONS_COUNT:

```
ALTER TABLE `my_table` SET (AUTO_PARTITIONING_MIN_PARTITIONS_COUNT = 1);
```

SDK

Что делать, если SDK экстренно завершает работу при завершении работы приложения?

Не следует оборачивать компоненты SDK в синглтон, так как время их жизни не должно превышать время выполнения функции `main()`. Так как при уничтожении клиента происходит опустошение пулов сессий, требуется возможность перехода по сети. Но gPRC содержит глобальные статические переменные, которые к этому времени уже могут быть разрушены. Таким образом, работа gRPC невозможна. При необходимости сделать драйвер глобальным объектом, на драйвере нужно вызвать функцию `Stop(true)` перед выходом из `main()`.

Что делать, если при вызове `fork()` в дочернем процессе программа работает некорректно?

Вызов `fork()` в многопоточных приложениях является антипаттерном. Так как и SDK, и gRPC-библиотека являются многопоточными приложениями, стабильность их работы не гарантируется.

Что делать, если я получаю ошибку «Active sessions limit exceeded», хотя текущее количество активных сессий не превышает лимит?

Лимит действует на количество активных сессий. Активная сессия — сессия, переданная клиенту для использования в его коде. Сессия возвращается в пул в деструкторе. При этом, сама сессия является копируемым объектом. Возможно, вы сохранили в коде копию сессии.

Есть ли возможность делать запросы к разным БД из одного приложения?

Да, C++ SDK позволяет переопределять параметры БД и токен в момент создания клиента. Отдельные драйверы создавать не нужно.

Что делать, если отказала одна из ВМ и невозможно сделать запрос?

Чтобы выявить недоступность ВМ, нужно проставить клиентский таймаут. Все запросы содержат параметры для клиентского таймаута. Величина таймаута должна быть на порядок больше ожидаемого времени выполнения запроса.

Загрузки YDB

В этом разделе представлены инструкции по загрузке различных сборок YDB и связанных инструментов:

- Сервер YDB (`ydbd`):
  - YDB Open-Source Database
  - Yandex Enterprise Database (Корпоративная СУБД Яндекса)
- Клиентская утилита командной строки (`ydb`)
- Утилита для управления дисковой подсистемой (`ydb-dstool`)
- Утилита для управления кластером (`ydbops`)
- Ansible-сценарии

Загрузка YDB Open-Source Database

YDB Open-Source Database (`ydbd`) — исполняемый файл для запуска узла кластера YDB. Распространяется под лицензией Apache 2.0.

См. также Загрузка Корпоративной СУБД Яндекса (Yandex Enterprise Database).

Linux

Версия	Дата выпуска	Скачать	Список изменений
v26.3			
v26.3.1.16-rc	18.09.26	Бинарный файл	См. список
v26.2			
v26.2.1.14	16.09.26	Бинарный файл	См. список
v26.1			
v26.1.1.22	27.07.26	Бинарный файл	См. список
v26.1.1.20	02.07.26	Бинарный файл	См. список
v25.4			
v25.4.1.15	24.04.26	Бинарный файл	См. список
v25.3			
v25.3.1.27	20.05.26	Бинарный файл	См. список
v25.2			
v25.2.1.26	12.05.26	Бинарный файл	См. список
v25.2.1.10-rc	21.09.25	Бинарный файл	См. список
v25.1			
v25.1.4.18	12.05.26	Бинарный файл	См. список
v25.1.2.7-rc	14.07.25	Бинарный файл	См. список
v24.4			
v24.4.4.12	03.06.25	Бинарный файл	См. список
v24.4.4.2	15.04.25	Бинарный файл	См. список
v24.3			
v24.3.15.5	06.02.25	Бинарный файл	См. список
v24.3.11.14	09.01.25	Бинарный файл	См. список
v24.3.11.13	24.12.24	Бинарный файл	См. список
v24.2			
v24.2.7	20.08.24	Бинарный файл	См. список
v24.1			
v24.1.18	31.07.24	Бинарный файл	См. список
v23.4			
v23.4.11	14.05.24	Бинарный файл	См. список
v23.3			
v23.3.17	14.12.23	Бинарный файл	
v23.3.13	12.10.23	Бинарный файл	См. список
v23.2			
v23.2.12	14.08.23	Бинарный файл	См. список

Docker

Версия	Дата выпуска	Docker `registry/image:tag`	Список изменений
v26.3			
v26.3.1.16-rc	18.09.26	`cr.yandex/crptqonuodf51kdj7a7d/ydb:26.3.1.16-rc`	См. список
v26.2			
v26.2.1.14	16.09.26	`cr.yandex/crptqonuodf51kdj7a7d/ydb:26.2.1.14`	См. список

v26.1			
v26.1.1.22	27.07.26	`cr.yandex/crptqonuodf51kdj7a7d/ydb:26.1.1.22`	См. список
v26.1.1.20	02.07.26	`cr.yandex/crptqonuodf51kdj7a7d/ydb:26.1.1.20`	См. список
v25.4			
v25.4.1.15	24.04.26	`cr.yandex/crptqonuodf51kdj7a7d/ydb:25.4.1.15`	См. список
v25.3			
v25.3.1.27	20.05.26	`cr.yandex/crptqonuodf51kdj7a7d/ydb:25.3.1.27`	См. список
v25.2			
v25.2.1.26	12.05.26	`cr.yandex/crptqonuodf51kdj7a7d/ydb:25.2.1.26`	См. список
v25.2.1.10-rc	21.09.25	`cr.yandex/crptqonuodf51kdj7a7d/ydb:25.2.1.10-rc`	См. список
v25.1			
v25.1.4.18	12.05.26	`cr.yandex/crptqonuodf51kdj7a7d/ydb:25.1.4.18`	См. список
v25.1.2.7-rc	14.07.25	`cr.yandex/crptqonuodf51kdj7a7d/ydb:25.1.2.7-rc`	См. список
v24.4			
v24.4.4.12	03.06.25	`cr.yandex/crptqonuodf51kdj7a7d/ydb:24.4.4.12`	См. список
v24.4.4.2	15.04.25	`cr.yandex/crptqonuodf51kdj7a7d/ydb:24.4.4.2`	См. список
v24.3			
v24.3.15.5	06.02.25	`cr.yandex/crptqonuodf51kdj7a7d/ydb:24.3.15.5`	См. список
v24.3.11.14	09.01.25	`cr.yandex/crptqonuodf51kdj7a7d/ydb:24.3.11.14`	См. список
v24.3.11.13	24.12.24	`cr.yandex/crptqonuodf51kdj7a7d/ydb:24.3.11.13`	См. список
v24.2			
v24.2.7	20.08.24	`cr.yandex/crptqonuodf51kdj7a7d/ydb:24.2.7`	См. список
v24.1			
v24.1.18	31.07.24	`cr.yandex/crptqonuodf51kdj7a7d/ydb:24.1.18`	См. список
v23.4			
v23.4.11	14.05.24	`cr.yandex/crptqonuodf51kdj7a7d/ydb:23.4.11`	См. список
v23.3			
v23.3.17	14.12.23	`cr.yandex/crptqonuodf51kdj7a7d/ydb:23.3.17`	
v23.3.13	12.10.23	`cr.yandex/crptqonuodf51kdj7a7d/ydb:23.3.13`	См. список
v23.2			
v23.2.12	14.08.23	`cr.yandex/crptqonuodf51kdj7a7d/ydb:23.2.12`	См. список

Исходный код

Версия	Дата выпуска	Ссылка	Список изменений
v26.3			
v26.3.1.16-rc	18.09.26	https://github.com/ydb-platform/ydb/tree/26.3.1.16-rc	См. список
v26.2			
v26.2.1.14	16.09.26	https://github.com/ydb-platform/ydb/tree/26.2.1.14	См. список
v26.1			
v26.1.1.22	27.07.26	https://github.com/ydb-platform/ydb/tree/26.1.1.22	См. список
v26.1.1.20	02.07.26	https://github.com/ydb-platform/ydb/tree/26.1.1.20	См. список
v25.4			
v25.4.1.15	24.04.26	https://github.com/ydb-platform/ydb/tree/25.4.1.15	См. список
v25.3			
v25.3.1.27	20.05.26	https://github.com/ydb-platform/ydb/tree/25.3.1.27	См. список
v25.2			
v25.2.1.26	12.05.26	https://github.com/ydb-platform/ydb/tree/25.2.1.26	См. список

v25.2.1.10-rc	21.09.25	https://github.com/ydb-platform/ydb/tree/25.2.1.10-rc	См. список
v25.1			
v25.1.4.18	12.05.26	https://github.com/ydb-platform/ydb/tree/25.1.4.18	См. список
v25.1.2.7-rc	14.07.25	https://github.com/ydb-platform/ydb/tree/25.1.2.7-rc	См. список
v24.4			
v24.4.4.12	03.06.25	https://github.com/ydb-platform/ydb/tree/24.4.4.12	См. список
v24.4.4.2	15.04.25	https://github.com/ydb-platform/ydb/tree/24.4.4.2	См. список
v24.3			
v24.3.15.5	06.02.25	https://github.com/ydb-platform/ydb/tree/24.3.15.5	См. список
v24.3.11.14	09.01.25	https://github.com/ydb-platform/ydb/tree/24.3.11.14	См. список
v24.3.11.13	24.12.24	https://github.com/ydb-platform/ydb/tree/24.3.11.13	См. список
v24.2			
v24.2.7	20.08.24	https://github.com/ydb-platform/ydb/tree/24.2.7	См. список
v24.1			
v24.1.18	31.07.24	https://github.com/ydb-platform/ydb/tree/24.1.18	См. список
v23.4			
v23.4.11	14.05.24	https://github.com/ydb-platform/ydb/tree/23.4.11	См. список
v23.3			
v23.3.17	14.12.23	https://github.com/ydb-platform/ydb/tree/23.3.17	
v23.3.13	12.10.23	https://github.com/ydb-platform/ydb/tree/23.3.13	См. список
v23.2			
v23.2.12	14.08.23	https://github.com/ydb-platform/ydb/tree/23.2.12	См. список

Загрузка Корпоративной СУБД Яндекса (Yandex Enterprise Database)

Корпоративная СУБД Яндекса — коммерческая СУБД на основе ядра YDB.

Информация об условиях использования Корпоративной СУБД Яндекса

Корпоративная СУБД Яндекса распространяется в соответствии с условиями лицензионного соглашения.

Бесплатное использование

Корпоративная СУБД Яндекса (Продукт) может использоваться бесплатно на условиях лицензионного соглашения в следующих целях:

- для оценки и изучения свойств Продукта;
- в рамках процессов разработки и тестирования программного обеспечения, использующего или встраивающего функции Продукта;
- для обучения администрированию Продукта;
- для обучения особенностям разработки программного обеспечения, взаимодействующего с Продуктом.

Коммерческое использование

Для коммерческого использования Корпоративной СУБД Яндекса необходимо приобретение лицензии, текст лицензионного соглашения. Цены на лицензии доступны по запросу.

Скачивание дистрибутивов

Дистрибутивы Корпоративной СУБД Яндекса доступны для скачивания по приведенным ниже ссылкам.

Linux

YDB Enterprise Server (`ydbd`) — исполняемый файл для запуска узла Корпоративной СУБД Яндекса.

Версия	Дата выпуска	Скачать	Контрольные суммы	Список изменений
v26.1				
v.26.1.1.ent.3	03.08.2026	Дистрибутив Отладочные символы	К дистрибутиву К отладочным символам	См. список
v25.4				
v.25.4.1.ent.2	17.06.2026	Дистрибутив Отладочные символы	К дистрибутиву К отладочным символам	См. список
v25.3				
v.25.3.1.ent.3	11.06.2026	Дистрибутив Отладочные символы	К дистрибутиву К отладочным символам	См. список
v25.2				
v.25.2.1.ent.13	11.06.2026	Дистрибутив Отладочные символы	К дистрибутиву К отладочным символам	См. список
v.25.2.1.ent.4	12.02.2026	Дистрибутив Отладочные символы	К дистрибутиву К отладочным символам	См. список
v25.1				
v.25.1.4.ent.8	12.02.2026	Дистрибутив Отладочные символы	К дистрибутиву К отладочным символам	См. список
v.25.1.4.ent.3	25.11.2025	Дистрибутив Отладочные символы	К дистрибутиву К отладочным символам	См. список
v24.4				
v.24.4.4.20	01.11.2025	Дистрибутив Отладочные символы	К дистрибутиву К отладочным символам	См. список
v.24.4.4.15	19.09.2025	Дистрибутив Отладочные символы	К дистрибутиву К отладочным символам	См. список
v.24.4.4.13	29.07.2025	Дистрибутив Отладочные символы	К дистрибутиву К отладочным символам	См. список
v24.3				
v.24.3.13.11	06.03.2024	Дистрибутив Отладочные символы	К дистрибутиву К отладочным символам	См. список
v.24.3.13.10	24.12.2024	Дистрибутив Отладочные символы	К дистрибутиву К отладочным символам	См. список
v24.2				
v.24.2.7.1	20.08.2024	Дистрибутив Отладочные символы	К дистрибутиву К отладочным символам	См. список
v24.1				
v.24.1.18.1	28.06.2024	Дистрибутив Отладочные символы	К дистрибутиву К отладочным символам	См. список

v23.4				
v.23.4.11.1	15.05.2024	Дистрибутив	Контрольные суммы	См. список
v23.3				
v.23.3.25.2	17.03.2024	Дистрибутив	Контрольные суммы	См. список

Docker

Версия	Дата выпуска	Скачать	Список изменений	
v26.1				
v.26.1.1.ent.3	03.08.2026	`cr.yandex/crptqonuodf51kdj7a7d/ydb-enterprise:26.1.1.ent.3`	См. список	
v25.4				
v.25.4.1.ent.2	17.06.2026	`cr.yandex/crptqonuodf51kdj7a7d/ydb-enterpise:25.4.1.ent.2`	См. список	
v25.3				
v.25.3.1.ent.3	11.06.2026	`cr.yandex/crptqonuodf51kdj7a7d/ydb-enterpise:25.3.1.ent.3`	См. список	
v25.2				
v.25.2.1.ent.13	11.06.2026	`cr.yandex/crptqonuodf51kdj7a7d/ydb-enterpise:25.2.1.ent.13`	См. список	
v.25.2.1.ent.4	12.02.2026	`cr.yandex/crptqonuodf51kdj7a7d/ydb-enterpise:25.2.1.ent.4`	См. список	
v25.1				
v.25.1.4.ent.8	12.02.2026	`cr.yandex/crptqonuodf51kdj7a7d/ydb-enterpise:25.1.4.ent.8`	См. список	
v.25.1.4.ent.3	25.11.2025	`cr.yandex/crptqonuodf51kdj7a7d/ydb-enterpise:25.1.4.ent.3`	См. список	
v24.4				
v.24.4.4.20	01.11.2025	`cr.yandex/crptqonuodf51kdj7a7d/ydb-enterpise:24.4.4.ent.20`	См. список	
v.24.4.4.15	19.09.2025	`cr.yandex/crptqonuodf51kdj7a7d/ydb-enterpise:24.4.4.ent.15`	См. список	
v.24.4.4.13	29.07.2025	`cr.yandex/crptqonuodf51kdj7a7d/ydb-enterpise:2025-08-08-73940688fea2aeb2e109f01a7827f2d7`	См. список	
v24.3				
v.24.3.13	05.12.2024	`cr.yandex/crptqonuodf51kdj7a7d/ydb:24.3.11.13`	См. список	
v24.2				
v.24.2.7	20.08.2024	`cr.yandex/crptqonuodf51kdj7a7d/ydb:24.2.7`	См. список	
v24.1				
v.24.1.18	31.07.2024	`cr.yandex/crptqonuodf51kdj7a7d/ydb:24.1.18`	См. список	
v23.4				
v.23.4.11	14.05.24	`cr.yandex/crptqonuodf51kdj7a7d/ydb:23.4.11`	См. список	
v23.3				
v.23.3.17	14.12.23	`cr.yandex/crptqonuodf51kdj7a7d/ydb:23.3.17`	См. список	
v.23.3.13	12.10.23	`cr.yandex/crptqonuodf51kdj7a7d/ydb:23.3.13`	См. список	

Сертифицированная версия

Дистрибутивы сертифицированных версий Корпоративной СУБД Яндекса не доступны для свободного скачивания. Доступ к ним предоставляется по запросу.

Сертифицированная версия Корпоративной СУБД Яндекса 25.1 имеет срок окончания технической поддержки 30.10.2030.

Загрузка YDB CLI

YDB CLI (`ydb`) — утилита командной строки для работы с базами данных YDB. Распространяется под лицензией Apache 2.0.

> **Совет**
>
> Рекомендуемый способ установки YDB CLI — с помощью скрипта установки. Скрипт автоматически скачает подходящий бинарный файл и добавит его в `PATH`.

Linux (amd64)

Версия	Дата выпуска	Скачать	Список изменений
v.2.33.0	23.06.2026	Бинарный файл	См. список
v.2.32.0	22.06.2026	Бинарный файл	См. список
v.2.31.0	20.04.2026	Бинарный файл	См. список
v.2.30.0	07.04.2026	Бинарный файл	См. список
v.2.29.0	11.02.2026	Бинарный файл	См. список
v.2.28.0	19.12.2025	Бинарный файл	См. список
v.2.27.0	30.10.2025	Бинарный файл	См. список
v.2.26.0	25.09.2025	Бинарный файл	См. список
v.2.25.0	01.09.2025	Бинарный файл	См. список
v.2.24.1	28.07.2025	Бинарный файл	См. список
v.2.24.0	23.07.2025	Бинарный файл	См. список
v.2.23.0	16.07.2025	Бинарный файл	См. список
v.2.22.1	17.06.2025	Бинарный файл	См. список
v.2.22.0	04.06.2025	Бинарный файл	См. список
v.2.21.0	22.05.2025	Бинарный файл	См. список
v.2.20.0	05.03.2025	Бинарный файл	См. список
v.2.19.0	05.02.2025	Бинарный файл	См. список
v.2.18.0	24.12.24	Бинарный файл	См. список
v.2.17.0	04.12.24	Бинарный файл	См. список
v.2.16.0	26.11.24	Бинарный файл	См. список
v.2.15.0	17.10.24	Бинарный файл	
v.2.14.0	03.10.24	Бинарный файл	
v.2.13.0	23.09.24	Бинарный файл	
v.2.12.0	19.09.24	Бинарный файл	
v.2.11.0	15.07.24	Бинарный файл	
v.2.10.0	24.06.24	Бинарный файл	См. список

Linux (arm64)

Версия	Дата выпуска	Скачать	Список изменений
v.2.33.0	23.06.2026	Бинарный файл	См. список
v.2.32.0	22.06.2026	Бинарный файл	См. список
v.2.31.0	20.04.2026	Бинарный файл	См. список
v.2.30.0	07.04.2026	Бинарный файл	См. список
v.2.29.0	11.02.2026	Бинарный файл	См. список
v.2.28.0	19.12.2025	Бинарный файл	См. список
v.2.27.0	30.10.2025	Бинарный файл	См. список
v.2.26.0	25.09.2025	Бинарный файл	См. список
v.2.25.0	01.09.2025	Бинарный файл	См. список
v.2.24.1	28.07.2025	Бинарный файл	См. список
v.2.24.0	23.07.2025	Бинарный файл	См. список

v.2.23.0	16.07.2025	Бинарный файл	См. список
v.2.22.1	17.06.2025	Бинарный файл	См. список
v.2.22.0	04.06.2025	Бинарный файл	См. список
v.2.21.0	22.05.2025	Бинарный файл	См. список
v.2.20.0	05.03.2025	Бинарный файл	См. список
v.2.19.0	05.02.2025	Бинарный файл	См. список
v.2.18.0	24.12.24	Бинарный файл	См. список
v.2.17.0	04.12.24	Бинарный файл	См. список
v.2.16.0	26.11.24	Бинарный файл	См. список
v.2.15.0	17.10.24	Бинарный файл	
v.2.14.0	03.10.24	Бинарный файл	

macOS (Intel)

Используйте бинарный файл amd64, если ваш Mac на базе процессора Intel.

Версия	Дата выпуска	Скачать	Список изменений
v.2.33.0	23.06.2026	Бинарный файл	См. список
v.2.32.0	22.06.2026	Бинарный файл	См. список
v.2.31.0	20.04.2026	Бинарный файл	См. список
v.2.30.0	07.04.2026	Бинарный файл	См. список
v.2.29.0	11.02.2026	Бинарный файл	См. список
v.2.28.0	19.12.2025	Бинарный файл	См. список
v.2.27.0	30.10.2025	Бинарный файл	См. список
v.2.26.0	25.09.2025	Бинарный файл	См. список
v.2.25.0	01.09.2025	Бинарный файл	См. список
v.2.24.1	28.07.2025	Бинарный файл	См. список
v.2.24.0	23.07.2025	Бинарный файл	См. список
v.2.23.0	16.07.2025	Бинарный файл	См. список
v.2.22.1	17.06.2025	Бинарный файл	См. список
v.2.22.0	04.06.2025	Бинарный файл	См. список
v.2.21.0	22.05.2025	Бинарный файл	См. список
v.2.20.0	05.03.2025	Бинарный файл	См. список
v.2.19.0	05.02.2025	Бинарный файл	См. список
v.2.18.0	24.12.24	Бинарный файл	См. список
v.2.17.0	04.12.24	Бинарный файл	См. список
v.2.16.0	26.11.24	Бинарный файл	См. список
v.2.15.0	17.10.24	Бинарный файл	
v.2.14.0	03.10.24	Бинарный файл	
v.2.13.0	23.09.24	Бинарный файл	
v.2.12.0	19.09.24	Бинарный файл	
v.2.11.0	15.07.24	Бинарный файл	
v.2.10.0	24.06.24	Бинарный файл	См. список

macOS (Apple Silicon)

Используйте бинарный файл arm64, если ваш Mac на базе процессора Apple Silicon (M1, M2 или более поздние).

Версия	Дата выпуска	Скачать	Список изменений
v.2.33.0	23.06.2026	Бинарный файл	См. список
v.2.32.0	22.06.2026	Бинарный файл	См. список
v.2.31.0	20.04.2026	Бинарный файл	См. список

v.2.30.0	07.04.2026	Бинарный файл	См. список
v.2.29.0	11.02.2026	Бинарный файл	См. список
v.2.28.0	19.12.2025	Бинарный файл	См. список
v.2.27.0	30.10.2025	Бинарный файл	См. список
v.2.26.0	25.09.2025	Бинарный файл	См. список
v.2.25.0	01.09.2025	Бинарный файл	См. список
v.2.24.1	28.07.2025	Бинарный файл	См. список
v.2.24.0	23.07.2025	Бинарный файл	См. список
v.2.23.0	16.07.2025	Бинарный файл	См. список
v.2.22.1	17.06.2025	Бинарный файл	См. список
v.2.22.0	04.06.2025	Бинарный файл	См. список
v.2.21.0	22.05.2025	Бинарный файл	См. список
v.2.20.0	05.03.2025	Бинарный файл	См. список
v.2.19.0	05.02.2025	Бинарный файл	См. список
v.2.18.0	24.12.24	Бинарный файл	См. список
v.2.17.0	04.12.24	Бинарный файл	См. список
v.2.16.0	26.11.24	Бинарный файл	См. список
v.2.15.0	17.10.24	Бинарный файл	
v.2.14.0	03.10.24	Бинарный файл	
v.2.13.0	23.09.24	Бинарный файл	
v.2.12.0	19.09.24	Бинарный файл	
v.2.11.0	15.07.24	Бинарный файл	
v.2.10.0	24.06.24	Бинарный файл	См. список

Windows

Версия	Дата выпуска	Скачать	Список изменений
v.2.33.0	23.06.2026	Бинарный файл	См. список
v.2.32.0	22.06.2026	Бинарный файл	См. список
v.2.31.0	20.04.2026	Бинарный файл	См. список
v.2.30.0	07.04.2026	Бинарный файл	См. список
v.2.29.0	11.02.2026	Бинарный файл	См. список
v.2.28.0	19.12.2025	Бинарный файл	См. список
v.2.27.0	30.10.2025	Бинарный файл	См. список
v.2.26.0	25.09.2025	Бинарный файл	См. список
v.2.25.0	01.09.2025	Бинарный файл	См. список
v.2.24.1	28.07.2025	Бинарный файл	См. список
v.2.24.0	23.07.2025	Бинарный файл	См. список
v.2.23.0	16.07.2025	Бинарный файл	См. список
v.2.22.1	17.06.2025	Бинарный файл	См. список
v.2.22.0	04.06.2025	Бинарный файл	См. список
v.2.21.0	22.05.2025	Бинарный файл	См. список
v.2.20.0	05.03.2025	Бинарный файл	См. список
v.2.19.0	05.02.2025	Бинарный файл	См. список
v.2.18.0	24.12.24	Бинарный файл	См. список
v.2.17.0	04.12.24	Бинарный файл	См. список
v.2.16.0	26.11.24	Бинарный файл	См. список
v.2.15.0	17.10.24	Бинарный файл	
v.2.14.0	03.10.24	Бинарный файл	

v.2.13.0	23.09.24	Бинарный файл	
v.2.12.0	19.09.24	Бинарный файл	
v.2.11.0	15.07.24	Бинарный файл	
v.2.10.0	24.06.24	Бинарный файл	См. список

Загрузка YDB DSTool

YDB DSTool (`ydb-dstool`) — утилита командной строки для управления дисковой подсистемой кластера YDB.

Для использования утилиты установите ydb-dstool.

Linux (amd64)

Версия	Дата выпуска	Скачать	Список изменений
v.0.0.14	18.07.2025	Бинарный файл	
v.0.0.13	20.11.2024	Бинарный файл	
v.0.0.12	25.10.2024	Бинарный файл	

Linux (arm64)

Версия	Дата выпуска	Скачать	Список изменений
v.0.0.14	18.07.2025	Бинарный файл	

macOS (Intel)

Используйте бинарный файл amd64, если ваш Mac на базе процессора Intel.

Версия	Дата выпуска	Скачать	Список изменений
v.0.0.14	18.07.2025	Бинарный файл	
v.0.0.13	20.11.2024	Бинарный файл	
v.0.0.12	25.10.2024	Бинарный файл	

macOS (Apple Silicon)

Используйте бинарный файл arm64, если ваш Mac на базе процессора Apple Silicon (M1, M2 или более поздние).

Версия	Дата выпуска	Скачать	Список изменений
v.0.0.14	18.07.2025	Бинарный файл	

Windows

Версия	Дата выпуска	Скачать	Список изменений
v.0.0.13	20.11.2024	Бинарный файл	
v.0.0.12	25.10.2024	Бинарный файл	

Загрузка YDB Ops

YDB Ops (`ydbops`) — утилита командной строки для управления кластером YDB.

Linux

Версия	Дата выпуска	Скачать	Список изменений
v0.0.25	03/12/2026	Бинарный файл	

macOS (Intel)

Версия	Дата выпуска	Скачать	Список изменений
v0.0.25	03/12/2026	Бинарный файл	

macOS (Apple Silicon)

Версия	Дата выпуска	Скачать	Список изменений
v0.0.25	03/12/2026	Бинарный файл	

Загрузка Ansible Playbooks для YDB

Набор автоматизированных сценариев для установки и сопровождения серверной части YDB Open-Source или Корпоративной СУБД Яндекса с помощью инструмента Ansible доступен для скачивания по ссылкам ниже. Исходный код сценариев опубликован на GitHub под лицензией Apache 2.0.

Версия	Дата выпуска	Скачать	Список изменений
v2.1.0	29.06.2026	ydb-ansible-2.1.0.tar.gz	См. список
v2.0.0	23.12.2025	ydb-ansible-2.0.0.tar.gz	См. список
v1.3.2	02.12.2025	ydb-ansible-1.3.2.tar.gz	
v0.15	13.12.2024	ydb-ansible-0.15.zip	Минимальный набор скриптов и ролей для запуска (устарело) — см. список

Список изменений Ansible Playbooks

Версия 2.1.0

Функциональность

- Добавлена поддержка Ansible 2.19 и более новых версий.
- Добавлены плейбуки для миграции кластеров между конфигурациями V1 и V2:
  - `migrate_to_v1`
  - `migrate_to_v2`
- Добавлена поддержка новых минорных версий дистрибутивов Linux.
- Отключён healthcheck для YDB 25.*.
- Добавлены новые плейбуки:
  - `generate_conf`
  - `run_dstool`
  - `run_ydbd`
- Добавлена поддержка раздельной установки статических и динамических узлов.

Улучшения

- Оптимизирована роль `ydbd_newdb`.
- Дашборды Grafana синхронизированы с репозиторием YDB.

Версия 2.0.0

Функциональность

- Добавлена поддержка установки кластера с конфигурацией YDB V2 (Distconf).
- Добавлена поддержка установки кластера в режиме 2DC.
- Добавлены новые плейбуки для CI:
  - `ydb_platform.ydb.run_ydbd`
  - `ydb_platform.ydb.run_dstool`
  - `ydb_platform.ydb.prepare_drives`

Улучшения

- Обновлена документация.
- Расширена валидация и внесены улучшения для упрощения процесса установки.
- Поддержка YDB 25+

Версия 1.3.2

25+ желательно не использовать.

Функциональность

- Коллекция переработана: вместо отдельных скриптов теперь используются Ansible-плейбуки.
- Добавлен единый плейбук для установки всего кластера YDB.
- Добавлена поддержка установки из файлов дистрибутивов YDB.
- Расширена поддержка современных операционных систем.

Улучшения

- Оптимизирован процесс установки кластера.
- Улучшена диагностика ошибок установки.

Версия 0.15

Устаревшая версия.
Коллекция отдельных скриптов и ролей. Была замещена плейбуками версий 1.x.x.

Видеозаписи

В этом разделе собраны видеозаписи с конференций и вебинаров о YDB:

- Видеозаписи 2026
- Видеозаписи 2025
- Видеозаписи 2024
- Видеозаписи 2023
- Видеозаписи 2022
- Видеозаписи 2021
- Видеозаписи 2020
- Видеозаписи 2019
- Видеозаписи 2018
- Видеозаписи 2017

Материалы разделены по категориям и размечены тегами:

Overview	– обзорные материалы, знакомящие с YDB и технологиями, которые применяются в ней.
Use cases	– кейсы использования YDB.
Practice	– практика использования YDB.
Database internals	– подробный разбор внутренней реализации YDB или её отдельных частей и механизмов.
Releases	– обзор новых возможностей, вышедших релизов YDB.
Testing	– кейсы тестирования производительности YDB и сравнения с другими одноклассовыми СУБД.
General	– материалы общего направления.

Видеозаписи 2026

Кирилл Курдюков — YDB и C#: история большой любви

Practice

В этом докладе представим себя в роли разработчиков, перед которыми стоит задача интеграции .NET-платформы с новой СУБД. Какие ключевые аспекты и проблемы необходимо учесть для успешной интеграции? Увидим, что интеграция YDB с .NET — это союз по любви, а не брак по расчету.

Доклад будет интересен разработчикам приложений на .NET и C#, а также инженерам, которые выбирают или внедряют клиентские SDK для работы с распределёнными СУБД.

КоРутинная практика: пишем свой TPC-C на корутинах

General

Евгений Иванов (Старший разработчик) рассказал о практическом применении корутин в C++ на примере реализации TPC-C — популярного бенчмарка для баз данных.

В докладе сравнили синхронный подход, асинхронную реализация на `future` и пулах потоков и вариант на корутинах. Разобрались, где выполняется код, как выбранная модель влияет на производительность и сложность программы и почему для TPC-C в YDB корутины оказались удачным компромиссом. Отдельно заглянули под капот C++-корутин: посмотрели, во что компилятор превращает `co_await`, и вручную написали continuation и state machine без использования языковых корутин.

Доклад будет интересен C++-разработчикам, которые работают с асинхронным кодом или хотят лучше понять устройство корутин.

Из доклада вы узнаете:

- почему тысячи потоков — плохой способ получить высокую конкурентность;
- чем отличаются синхронный код, `future` и корутины с точки зрения выполнения и производительности;
- что на самом деле происходит при `co_await`;

Слайды

Видеозаписи 2025

Транзакции YDB между топиками и таблицами на потоке порядка 3 Гб/с

Database internals

Доклад посвящён использованию транзакций YDB между топиками и таблицами на потоке порядка 3 Гб/с. В нём представлены способы решения задачи без использования данной функциональности, а также факторы, влияющие на производительность схемы при использовании функциональности. В частности, в докладе разобраны проблемы производительности, с которыми столкнулись в AppMetrica при миграции, и возможные способы их решения.

Доклад будет интересен разработчикам высоконагруженных систем, систем потоковой обработки данных, администраторам СУБД систем.

Демонстрация возможностей масштабирования и отказоустойчивости СУБД Яндекса

Database internals

Доклад Евгения Ярошa посвящен тестированию устойчивости базы данных Яндекс YDB к сбоям. В нём представлены различные сценарии отказов — от выключения сервера до отключения дата-центра. Затронуты особенности тестового стенда, на котором моделируется нагрузка из банковского процессинга. Кроме того, в докладе детально разобраны ключевые сценарии тестирования; особо отмечается, что YDB способна сохранять работоспособность даже при отключении трети кластера.

Доклад будет интересен системным архитекторам, администраторам СУБД систем и DevOps-инженерам.

Фичестор городских сервисов Яндекса/Яндекс GO

Practice

В своём докладе Павел Попов рассказывает про систему «Авалон» в Яндекс Go. «Авалон» - это хранилище признаков в сервисе Яндекс.GO с объемом данных 3 ТБ, 6,5 млрд ключей и 2048 токенов, сердцем которой является YDB. Доклад посвящен концепции, архитектуре и особенностям работы с данными системы «Авалон».

Доклад будет интересен backend-разработчикам, системным архитекторам и аналитикам данных.

Слайды

YDB: от SQL к распределённому SQL

Database internals

В мире, где объёмы данных растут с каждым днём, традиционные одноузловые базы данных уже не справляются с вызовами. Доклад Алексея Мясникова на Пых.конф'25 посвящён YDB — современной distributed SQL базе данных, которая воплощает тренды развития индустрии.

Вы узнаете из доклада:

- какие тренды происходят в индустрии обработки и хранения данных;
- ключевые особенности YDB, которые выгодно отличают её от других СУБД;
- принципы разработки SDK для работы с YDB;
- какой на сегодняшний день есть инструментарий для php-разработчиков, чтобы попробовать YDB;
- какие есть проблемы в связке PHP-YDB и как их можно решить.

Доклад будет интересен PHP-разработчикам прикладного программного обеспечения.

YouTube

VK

Слайды

От перегрузок к стабильности построение высоконагруженных систем с Axiom JDK, YDB и Jmix

Database internals

Вебинар, проведенный компанией «ФОРС Дистрибуция», был посвящен методам быстрой разработки высоконагруженных и масштабируемых бизнес-систем. В качестве основного технологического стека для решения этих задач были представлены фреймворк Jmix, среда исполнения Axiom JDK и распределённая отказоустойчивая СУБД YDB.

Материалы вебинара будут полезны ИТ-директорам, архитекторам, разработчикам и инженерам, которые занимаются проектированием, внедрением и поддержкой современных отечественных приложений, требующих высокой надёжности и производительности.

CI/CD как драйвер разработки

Use cases

Доклад Кирилла Сюзева посвящен эволюции CI/CD в сервисе SourceCraft, его архитектуре, выбору и анализу различных решений в области CI/CD (GitLab, GitHub, Arcadia CD), а также решению технических проблем. Одной из таких проблем является Real Time Logging, ключом к решению которой стало использование YDB как места для хранения логов и передаче их пользователю.

Доклад будет интересен DevOps-инженерам, разработчикам программного обеспечения.

Слайды

Эксплуатация распределенной системы на примере YDB

Database internals

Лекция, рассказанная Олегом Бондарем, в рамках Школы Анализа Данных, посвящена эксплуатации распределённой системы на примере кластеров YDB. В её рамках рассматриваются требования к инфраструктуре, инструменты, обеспечивающие удобство управления кластерами, а также подходы к резервному копированию и восстановлению данных.

Особое внимание уделяется методам обновления системы без прерывания обслуживания, управлению хостами и дисковыми ресурсами, мерам обеспечения безопасности — включая шифрование, управление правами доступа, аутентификацию и аудит — а также диагностике и мониторингу работоспособности и производительности кластеров.

Лекция будет полезна администраторам баз данных, инженерам баз данных и DevOps-инженерам.

YouTube

VK

Как YDB выполняет запросы

Database internals

Лекция, расказанная Виталием Гридневым в рамках Школы Анализа Данных, даёт вводное представление об архитектуре YDB: рассматриваются детерминистические транзакции и MVCC, объясняется жизненный цикл выполнения запроса и роль ключевых компонентов системы. На практических примерах показано, как обрабатываются запросы в кластере. Отдельно разбирается реализация вторичных индексов и особенности их добавления к существующим таблицам. В завершение освещаются подходы к тестированию YDB, включая методы верификации корректности и обеспечения надёжности.

Материал будет полезен backend-разработчикам, инженерам распределённых систем, SRE/DevOps-специалистам и QA-инженерам, работающим с высоконагруженными СУБД.

YouTube

VK

Петабайт в YDB over HDD в процессингах Яндекс.Метрики

Database internals

Антон Барабанов в своем докладе рассказывает о работе с данными в Яндекс.Метрике, включая процессинг данных с использованием YDB. Обсуждаются различные вопросы, возникающие при работе с базами данных — от сложностей с записью и чтением данных до неравномерной нагрузки на узлы кластера, а также вопросы оптимизации запросов, структуры данных и экономической выгоды использования SSD и HDD.

Эти темы будут интересны инженерам баз данных, разработчикам баз данных и администраторам баз данных.

Основы векторного и полнотекстового поиска

Database internals

В видео рассмотрены основы векторного и полнотекстового поиска, включая принципы работы векторной базы данных и алгоритмы поиска (точные и приближённые). Обсуждены методы оптимизации поиска — кластеризация, квантование, использование графовых алгоритмов и других техник. Также затронуты вопросы ранжирования, борьбы со спамом и реализации векторного поиска в базах данных.

Доклад будет полезен специалистам по анализу данных, ML-инженерам, разработчикам баз данных и backend-разработчикам.

Эволюция векторного поиска в YDB: от базовых методов к масштабируемому глобальному индексу

Database internals

YDB преодолела путь от простого и наивного векторного поиска к созданию продвинутого индекса.
Александр Зевайкин (Руководитель группы) на конференции Saint Highload 2025 рассказал о ключевых этапах развития, вызовах и решениях, которые позволили создать систему, способную эффективно обрабатывать миллиарды векторов и масштабироваться на тысячи узлов:

- как векторный поиск улучшает ответы больших языковых моделей;
- принципы работы векторного поиска и его применение в YDB;
- основные проблемы и сложности при разработке векторного индекса;
- особенности и преимущества глобального синхронного индекса;
- стратегии масштабирования и оптимизации производительности.

Этот доклад будет полезен всем, кто интересуется современными подходами к организации векторного поиска в больших данных.

Слайды

Чтение с реплик в распределённых системах: опыт YDB

Database internals

Александр Зевайкин (Руководитель группы) на конференции Я про бэкенд 2025 рассказал, как чтение с реплик в распределённых системах позволяет выполнять запросы к копиям данных с допустимым уровнем устаревания. И как это снижает задержки, повышает пропускную способность и устойчивость системы.

Также приведены архитектурные решения и практический опыт использования чтения с реплик в YDB.

YouTube

VK

Слайды

YDB — это просто! Администрирование YDB | СУБД Яндекса

General

На вебинаре представлены практические рекомендации по развёртыванию и администрированию универсальной распределённой СУБД YDB, разработанной Яндексом для высоконагруженных систем. Участникам показывается, как быстро развернуть YDB на собственном оборудовании, эффективно управлять крупными кластерами, а также выполнять обновление без простоев и потери доступности базы данных. В рамках вебинара разбираются основные сценарии административной работы с YDB, что помогает подготовиться к прохождению курса «Администратор YDB».

Вебинар будет полезен дата-инженерам, администраторам инфраструктуры и всем, кто хочет научиться развёртывать и поддерживать YDB.

Yandex

YouTube

YDB – архитектура, принципы работы СУБД и результаты тестирования

Вебинар, проведённый компанией «ФОРС Дистрибуция» — одним из крупнейших сервисных центров в России, специализирующимся на адаптации ИТ-инфраструктуры под новые реалии рынка, был посвящён внедрению, эксплуатации и развитию системы YDB. На вебинаре рассказали:

- об архитектурных особенностях YDB, её масштабируемости и отказоустойчивости;
- о современных подходах к мониторингу и диагностике состояния кластера YDB;
- о нюансах разработки приложений под YDB;
- о практиках и инструментах миграции данных в YDB;
- о методах тестирования горизонтальной масштабируемости YDB;
- о планах на будущие исследования и развитие системы.

Вебинар будет полезен ИТ-директорам, архитекторам, администраторам и инженерам, поддерживающим или внедряющим современные отечественные решения для хранения и обработки данных в условиях смены технологических платформ.

Подкаст Руцентра: Александр Зотов (СУБД Яндекса)

General

Данный выпуск подкаста посвящен обсуждению развития карьеры в IT-сфере и особенностям работы в области СУБД на примере опыта Александра Зотова, директора по развитию бизнеса СУБД Яндекса (YDB).

В этом выпуске обсуждаются особенности принятия управленческих решений в технологической отрасли и стратегический выбор между стабильностью и инновационным развитием. Особое внимание уделяется практическим аспектам работы с СУБД, роли искусственного интеллекта в обработке данных и текущим вызовам развития IT-индустрии.

Материал представляет собой экспертное мнение специалиста с практическим опытом работы в крупной технологической компании и будет полезен как начинающим специалистам, так и опытным профессионалам, интересующимся развитием карьеры в технологической сфере и особенностями работы с современными СУБД.

Стоимостный оптимизатор в YDB — как, зачем и почему?

Database internals

Данный доклад посвящен развитию стоимостного оптимизатора в распределенной СУБД YDB. Павел Велихов (Руководитель команды оптимизатора запросов) рассматривает эволюция системы от классической OLTP-архитектуры к гибридной HTAP-платформе с поддержкой колоночной аналитики. Подробно анализируется архитектура оптимизатора, включающая энумератор планов, механизм динамического программирования с алгоритмом DPhyp, систему оценки кардинальности и работу с гиперграфом запроса.

Особое внимание уделяется сравнению реализации с аналогичными решениями в других современных распределенных СУБД: CockroachDB, TiDB, Yugabyte и Trino. Представлены результаты бенчмарков, демонстрирующие эффективность оптимизатора, а также обсуждаются основные вызовы, возникшие в процессе разработки.

YDB Topics — корпоративный брокер сообщений от Яндекса

General

YDB Topics — совместимый с Apache Kafka корпоративный брокер сообщений, являющийся частью СУБД Яндекса YDB. На вебинаре Алексей Николаевский (Руководитель команды разработки стриминговых систем YDB) рассказал, как YDB Topics можно использовать для удобной разработки, быстрой миграции, безопасного администрирования и повышения отказоустойчивости микросервисной архитектуры.

Рассматриваются архитектурные решения, позволяющие совместить в YDB поддержку транзакционных, аналитических и потоковых нагрузок в рамках единой системы, а также преимущества такого подхода для корпоративных пользователей. Рассказывается, как благодаря поддержке распределённых ACID-транзакций между таблицами и топиками отпадает необходимость использования outbox-паттерна, что снижает сложность и затраты при интеграции микросервисов. Особое внимание уделяется гарантиям exactly-once при записи сообщений в ingest-сценариях, что позволяет эффективно бороться с дублями данных. Рассматриваются внутренние механизмы автоматического масштабирования YDB Topics, обеспечивающие быстрый ввод новых серверов и балансировку нагрузки без простоев. Также объясняется, почему для критичных бизнес-процессов важна полная доступность на запись (выше 99,99%), и каким образом YDB Topics этого достигает. В завершение приводятся возможности, важные для корпораций: совместимость с Apache Kafka, поддержка управления квотами и доступом, развитая ролевая модель, аудитные логи и сертификация.

Вебинар полезен разработчикам микросервисов, инженерам и архитекторам данных.

Как Яндекс разработал алгоритм для стоимостного оптимизатора в YDB

Database internals

Для поддержки эффективных аналитических запросов в СУБД Яндекса реализован механизм стоимостной оптимизации — алгоритм выбора оптимального плана выполнения запроса. При этом такая задача является NP-трудной — для её решения необходимо перебрать все варианты выполнения запроса, и время работы системы увеличивается экспоненциально.

На вебинаре рассматривается архитектура и реализация стоимостного оптимизатора в YDB, начиная от используемых теоретических основ до практического применения в производственных нагрузках. Подробно объясняется, как современные алгоритмы и методы динамического программирования, применённые в оптимизаторе запросов, позволяют системе эффективно обрабатывать сложные SQL-запросы с большим числом JOIN-операций. Отдельное внимание уделяется тому,

как реализованный оптимизатор обеспечивает высокую производительность СУБД, а также тому, за счёт каких технологических решений удаётся перебрать до 150 JOIN-операций за секунду.

Вебинар полезен тем, кто интересуется алгоритмами оптимизации, разработкой баз данных, динамическим программированием, — архитекторами и инженерам баз данных, разработчикам прикладного ПО и специалистам по системной интеграции.

YDB (СУБД Яндекса) для задач интеграции данных и бизнес-аналитики

Overview

На очередном вебинаре про аналитические возможности YDB детально рассматривается инструментарий для бизнес-аналитики и интеграции данных:

- федеративные запросы в СУБД и S3-совместимые хранилища данных;
- ETL и ELT-интеграции с Apache NiFi и Apache Airflow;
- обработка данных с помощью Apache Spark;
- анализ данных в Jupyter;
- инструменты визуализации: Grafana, DataLens, Apache Superset.

Вебинар будет полезен бизнес-аналитикам, архитекторам данных и специалистам, работающим с хранилищами данных.

Внутреннее устройство аналитической СУБД Яндекса — YDB DWH

Database internals

На вебинаре по глубокому погружению в архитектуру аналитических компонентов YDB (СУБД Яндекса) детально разобрали внутреннее устройство ключевых механизмов, отвечающих за эффективную обработку аналитических запросов:

- архитектура хранения данных в колоночных шардах YDB DWH и механизмы распределения данных в кластере;
- принципы работы и отличительные особенности стоимостного оптимизатора запросов YDB DWH;
- внутреннее устройство модуля управления нагрузкой (Workload manager) и его влияние на производительность базы данных.

Вебинар полезен разработчикам систем хранения данных, инженерам по оптимизации производительности и архитекторам распределённых систем, желающим глубже понять принципы работы современных СУБД.

Новые возможности YDB (СУБД Яндекса) для аналитических систем

General

На вебинаре рассказали о новой возможности YDB (СУБД Яндекса) — выполнению сложных аналитических запросов. Теперь СУБД Яндекса позволяет не только разрабатывать транзакционные приложения, но и строить аналитические системы в контуре заказчика — корпоративные хранилища данных. На мероприятии рассказали, как реализовали поддержку одновременно OLAP- и OLTP-нагрузок, добились производительности аналитических запросов в ресурсоёмких бизнес-критичных сценариях, а также поговорили про отличия YDB DWH от PostgreSQL, MongoDB, ClickHouse и Greenplum.

Вебинар будет полезен архитекторам и инженерам баз данных, SRE-инженерам, разработчикам прикладного ПО и специалистам по системной интеграции.

Секреты отладки производительности и ускорения YDB — распределенной СУБД на C++

Database internals

Евгений Иванов (Старший разработчик) рассказал, почему отладка производительности YDB и её оптимизация — это сложная, но увлекательная задача. Доклад посвящён практическому опыту поиска и устранения узких мест, методам тестирования производительности и общим подходам, полезным для разработчиков.

В конце доклада разобран практический пример использования самописного дискретно-событийного симулятора, раскрывающего красоту и пользу теории массового обслуживания в приложении к реальным сложным продакшн-системам.

Доклад будет интересен разработчикам высоконагруженных систем на C++, а также разработчикам на других языках программирования.

Из доклада вы узнаете:

- как значительно улучшить производительность без изменения кода — достаточно учитывать особенности NUMA;
- о «магии» компилятора, позволяющей ускорять код;
- как в YDB выстроена observability: метрики «на всё» и трассировка, пронизывающая систему;
- почему на флеймграфы можно смотреть бесконечно;
- и о том, что теория массового обслуживания не только полезна, но и по-настоящему увлекает.

Слайды

Видеозаписи 2024

Методы и подходы к тестированию распределённой базы данных

Testing

Виталий Гриднев (Ведущий разработчик) подробно рассказал о тестировании YDB. Доклад начинается с общего представления того, как организовано тестирование YDB, какие проблемы возникают при тестировании распределённых компонентов базы данных и почему в YDB необходимо проводить хаос-тестирование.

Затем Виталий углубляется в архитектуру тестового сценария для хаос-тестирования, рассматривая все этапы — от разработки генераторов и сценариев нагрузки до создания инструментов для внешнего воздействия на систему.

Вы узнаете, как с помощью характеристик safety и liveness можно выявлять дефекты в системе. Доклад будет полезен разработчикам в тестировании, интересующимся тестированием распределённых систем.

Слайды

Стоимостный оптимизатор в YDB — как, зачем и почему?

Database internals

YDB создавалась как OLTP-система для высоконагруженных проектов, но со временем в ней появились и аналитические запросы. Их сложно оптимизировать вручную, поэтому важную роль играет стоимостный оптимизатор.

Несколько лет назад команда YDB добавила поддержку колоночного хранения и сложной аналитики, что повысило требования к оптимизации. Теперь система может работать в режиме HTAP, где запросы автоматически распределяются между строковым и колоночным хранилищем.

В этом докладе Павел Велихов расскажет, как команда разработала стоимостный оптимизатор, какие задачи он решает и как YDB выглядит на фоне конкурентов в OLTP, OLAP и HTAP-сценариях.

Слайды

Доклад будет полезен разработчикам СУБД, аналитических платформ и высоконагруженных систем.

Транзакционная работа с топиками. Архитектура и сравнение решений в Apache Kafka и YDB

Database internals

Транзакции упрощают код и обеспечивают гарантии, такие как exactly-once доставка сообщений. В Apache Kafka транзакции позволяют достичь этих гарантий, но требуют сложной внутренней архитектуры. YDB изначально поддерживает транзакции, что влияет на подход к работе с топиками.

В докладе Алексея Николаевского рассматриваются:

- Понятия топика, транзакционной записи и чтения.
- Решение задачи решардирования с сохранением порядка и exactly-once обработки в Apache Kafka и YDB.
- Архитектуры транзакций в обеих системах.
- Сравнение производительности транзакций.

Слайды

Доклад будет полезен разработчикам распределенных и высоконагруженных систем, стремящимся к высоким гарантиям в обработке данных.

Как объединять данные из разных СУБД и делать это эффективно

Overview

Анализ данных, распределённых по различным системам хранения, таким как реляционные СУБД и CSV-файлы в S3, представляет собой сложную задачу. При небольшом объёме данных можно использовать скрипты для последовательного чтения и объединения информации в оперативной памяти. Однако с увеличением объёмов и сложности данных такой подход становится неэффективным.

В таких случаях на помощь приходят СУБД и движки обработки запросов с федеративными возможностями. Они позволяют использовать выразительный SQL для описания аналитических операций и эффективно обрабатывать большие объёмы данных из различных источников.

В своём докладе Виталий Исаев расскажет о принципах работы подобных систем и ключевых оптимизациях, обеспечивающих быструю и эффективную обработку данных.

Слайды

Доклад будет полезен разработчикам и аналитикам, работающим с данными, размещёнными в различных СУБД и хранилищах.

Гарантии доставки сообщений в YDB Topics

Database internals

Александр Зевайкин(Руководитель группы) продемонстрировал базовые проблемы ненадежной передачи данных и способы борьбы с ними: повторы и дедупликацию. На примере паттернов микропроцессорной архитектуры он проиллюстрировал гарантии доставки: at-most-once, at-least-once, exactly-once.

В завершение выступления Александр показал детали реализации гарантий доставки на примере двух брокеров очередей сообщений: Kafka и YDB Topics.

Слайды

Интеграция распределенных SQL-хранилищ с Java-фреймворками на примере YDB

Database internals

В докладе рассматриваются ключевые аспекты и основные вызовы, возникающие при интеграции популярных Java-фреймворков с распределённой СУБД на примере YDB. Освещаются проблемы, которые необходимо учесть для успешной интеграции. Докладчик раскрывает, как Java-фреймворки могут быть адаптированы и интегрированы с распределёнными SQL-хранилищами, и сравнивает различные решения в этой области.

Доклад будет полезен разработчикам, архитекторам и DevOps-инженерам, занимающимся внедрением или миграцией отечественных и распределённых СУБД в корпоративных Java-проектах.

Слайды

Гарантии доставки сообщений в YDB Topics

Database internals

Александр Зевайкин (Руководитель группы) рассмотрел основные проблемы ненадёжной передачи данных и методы борьбы с ними: повторы и дедупликацию. Александр на примере паттернов микропроцессорной архитектуры продемонстрировал гарантии доставки: at-most-once, at-least-once, exactly-once.

Александр с докладом «Гарантии доставки сообщений в YDB Topics» выступал на конференциях Saint HighLoad++ 2024 и ИТ-Пикник 2024:

Saint HighLoad++ 2024

ИТ-Пикник 2024

В заключение Александр на примере двух брокеров очередей сообщений — Kafka и YDB Topics — показал детали реализации гарантий доставки.

Слайды

Шардированный не значит распределенный: что важно знать, когда PostgreSQL мало

Testing

Евгений Иванов (Старший разработчик) и Олег Бондарь (CPO YDB) рассказали, чем отличаются распределённые СУБД от шардированных. Особое внимание уделили тому, почему решения, подобные Citus, не являются ACID в случае широких транзакций. В конце выступления на примере бенчмарка TPC-C показали, что в PostgreSQL вертикальное масштабирование ограничено ботлнеком в синхронной репликации, и сравнили производительность PostgreSQL и распределённых СУБД CockroachDB и YDB.

Доклад будет интересен как разработчикам приложений, которым требуется надёжная СУБД, так и людям, интересующимся распределёнными системами и базами данных.

Слайды

YDB — параллельная асинхронная репликация в распределенной базе

Database internals

Асинхронная репликация — хорошо известный метод создания независимой реплики базы данных, который дает возможность решить такие задачи, как обеспечение дополнительного уровня отказоустойчивости, георепликация и масштабирование рабочей нагрузки на чтение. Обычно все изменения исходной реплики пишутся в журнал изменений, который обрабатывается как входные данные для базы данных целевой реплики.

В YDB исходная и целевая реплики — это распределенные базы данных, которые справляются с высокой рабочей нагрузкой. В такой ситуации невозможно ограничиться единым журналом изменений, который можно было бы перенести в целевую базу данных. Олег Бондарь (CPO YDB) рассказал про то, как используются внутренние возможности YDB, чтобы обеспечить масштабируемость журнала изменений базы данных и гарантировать согласованность целевой базы данных.

Из доклада вы узнаете:

- Почему традиционный вариант хранения логов не подходит для распределенной БД?
- Что такое топики и как они помогают в хранении логов?
- С какими проблемами использования топиков для хранения логов столкнулась команда YDB и как их решала?
- Как обеспечить запись данных в целевой кластер при работе с распределенными транзакциями?

Слайды

YDB Topics: история взаимоотношений с Kafka

Database internals

Шина данных уровня Яндекса требует нестандартных решений. Александр Зевайкин (Руководитель группы) рассказал о том, как появился сервис YDB Topics и какие оптимизации были в нём применены. Также Александр описал архитектуру YDB Topics и объяснил, почему было принято решение отказаться от Kafka.

В завершение вы узнаете, как команда YDB реализовала Kafka API в режиме serverless.

Слайды

Видеозаписи 2023

Архитектура распределенного SQL-движка для аналитических запросов

Overview Database internals

Доклад посвящен архитектуре аналитического движка YDB, который позволяет выполнять сложные OLAP-запросы поверх транзакционной (OLTP) базы данных без влияния на основную нагрузку. Спикер Алексей Озерицкий детально разбирает, как это достигается.

Доклад будет интересен техническим специалистам интересующимся внутренним устройством распределенных систем и гибридных баз данных.

Слайды

Внедрение YDB CDC на примере Yandex Monitoring

Overview Database internals

Yandex Monitoring используется всеми сервисами Яндекса внутри, а также доступен в Яндекс Облаке для внешних пользователей. Мы обрабатываем 700 миллионов метрик на запись ежесекундно.

У каждого клиента есть конфигурация процесса загрузки метрик. Конфигурация — это Control-Plane-объекты, которые:

- хранятся как таблицы в базе данных;
- меняются на лету;
- связаны между собой.

Чем быстрее Monitoring получает обновления объектов, тем выше доступность для пользователей.

Для хранения мы используем YDB — распределённую open-source-базу данных Яндекса.
Команды YDB и Yandex Monitoring проработали сценарий загрузки изменений с помощью YDB CDC. В результате получили архитектуру, в которой изменения объектов доставляются в тысячи компонентов менее чем за 800 миллисекунд.

В докладе:

- Сложности до перехода на Change Data Capture;
- Подходы к доставке изменений, их особенности;
- Чем YDB CDC отличается от других решений;
- Рекомендации, как правильно готовить YDB CDC, с граблями и их решением;
- Какую модель данных выбрать, чтобы решить проблемы с конкурентными изменениями;
- Мониторинг и поддержка решения.

Доклад будет полезен программистам и архитекторам распределённых систем.

Слайды

Реализовать OLAP: как мы делали колоночное хранение в YDB

Overview Database internals

YDB – это платформа, которая умеет обрабатывать большой поток быстрых транзакций (OLTP, Online Transaction Processing). Команда, в которой работает Софья Новожилова (занимается развитием аналитического направления платформы YDB) захотела научить YDB обрабатывать другой тип запросов — аналитические (OLAP, Online Analytical Processing).

Из доклада вы узнаете ответы на вопросы:

- Достаточно ли просто поменять систему хранения, упаковать данные по колонкам, чтобы получить профит?
- Зачем это было нужно и какая польза от таких расширений системе в целом?

Доклад будет интересен разработчикам нагруженных систем и разработчикам платформ различного назначения.

Слайды

database/sql: плохой, хороший и злой. Опыт разработки драйвера для распределенной СУБД YDB

Database internals

В Golang есть пакет `database/sql`, который предоставляет универсальный интерфейс общения с базами данных. С течением времени `database/sql` сильно изменился и далеко ушёл от своего первоначального вида, он стал намного удобнее и функциональнее, но так было не всегда. Команда YDB работала с драйвером для YDB, начиная с версии Golang v1.11, и сталкивалась с различными трудностями в процессе эксплуатации в продакшнах пользователей.

В этом ретроспективном докладе Алексей Мясников (руководитель команды разработки клиентских библиотек YDB) расскажет о том, какие недочеты были в пакете `database/sql`, во что это выливалось при эксплуатации и как он становился все лучше от версии к версии Golang.

Доклад будет интересен разработчикам и пользователям драйверов в стандарте `database/sql`, а также пользователям распределенной базы данных YDB, разрабатывающим свой код на языке Golang.

Слайды

YDB-оптимизации производительности под ARM

Testing

Максим Кита (Разработчик) расскажет о том, с какими проблемами столкнулась команда YDB и как они их решали при оптимизации YDB под архитектуру ARM. В докладе детально рассмотрены основные проблемы оптимизаций высоконагруженных приложений под ARM, методы и инструменты, с помощью которых проводилось тестирование производительности и находились узкие места для оптимизации.

Доклад будет интересен разработчикам баз данных, разработчикам систем хранения данных и разработчикам высоконагруженных приложений.

Слайды

YDB Topic Service: как мы повышали производительность очереди сообщений

Database internals Testing

5 лет назад Яндекс перешел с Kafka на собственную разработку поверх YDB. С тех пор Yandex Topic Service сильно подрос по нагрузке и вышел в Open Source. В этом докладе Александр Зевайкин (Руководитель группы) рассказывает про ускорение YDB Topic Service и приводит сравнение с конкурентами.

Доклад будет интересен разработчикам, лидам разработки, техническим менеджерам. Всем, кто решал, решает или интересуется задачей масштабируемой поставки данных.

Слайды

Особенности шин данных для очень больших инсталляций на примере YDB Topics

Database internals

Шины передачи данных используются практически везде, но использование шин данных в очень больших инсталляциях на тысячи серверов накладывают особые требования для работы и приводят к отличиям в работе систем. Алексей Дмитриев (Технический менеджер) показывает на примере YDB Topics, в чем заключаются эти отличия, как они влияют на архитектуру и эксплуатацию.

Доклад будет интересен разработчикам и командам эксплуатации, особенно в компаниях больших размеров.

Слайды

Поиск по образцу на последовательностях строк в БД

Database internals

Задача поиска по образцу на последовательности строк БД может возникать в различных сферах деятельности. Например, в финансовой аналитике — поиск определённых паттернов изменения цены акций. Для реализации таких запросов к базам данных в стандарте `SQL:2016` была введена конструкция `MATCH_RECOGNIZE`. Евгений Зверев (Разработчик) рассказывает о реализации `MATCH_RECOGNIZE` в YDB: о том, как это работает под капотом, какие подходы и алгоритмы реализованы, с какими сложностями столкнулась команда.

Отдельная часть выступления посвящена отличиям в обработке аналитических запросов на табличках и обработке на потоках «живых» данных. Доклад будет интересен разработчикам БД, дата-аналитикам, а также всем, кто интересуется поиском по образцу на больших данных.

Слайды

Из pytest в Go. Тестовое окружение на фикстурах

Testing

Фикстуры позволяют писать очень лаконичные тесты и не отвлекаться на подготовку окружения. Тимофей Кулин (Разработчик) рассказал, как перенёс идеологию фикстур из pytest в Go и разработал библиотеку `fixenv`, которая сокращает код тестов и улучшает их читаемость.

Доклад будет интересен разработчикам на Go и тимлидам.

Слайды

Внутренняя платёжная система Яндекса: что под капотом?

Use cases

Доклад Антона Куранды «Внутренняя платёжная система Яндекса: что под капотом?» раскрывает архитектурные решения, применённые при создании высоконагруженной внутренней платёжной платформы Яндекса (Yandex PSP), рассчитанной на тысячи платежей в секунду и обработку сотен терабайт данных в год. Рассказывается об основных принципах: использование stateless-микросервисов на Golang, хранение состояний и событий в георасределённой СУБД YDB (в формате key-value), построение бизнес-логики на иерархических конечных автоматах с event sourcing, а также внедрение таск-шедулеров для планирования и асинхронной обработки задач. Подчёркивается, как данные подходы обеспечивают высокую отказоустойчивость (SLA ~99.997 %), ускорение проведения платежей в 6–10 раз и быстрый time-to-market для новых сервисов. Отмечается универсальность архитектуры — её можно применять в различных сферах: не только в финтехе, но и в e-commerce, автоматизации услуг, биллинге и логистике.

Этот доклад будет особенно полезен архитекторам, техническим руководителям, разработчикам высоконагруженных и транзакционных систем, а также специалистам, ищущим практические примеры построения крупных гибких платформ на современных инженерных принципах.

RuTube

YouTube

Слайды

YDB Java SDK: борьба за производительность

Overview

Александр Горшенин (Ведущий разработчик) даёт краткий обзор архитектуры YDB, описывает особенности распределенных систем, показывает необходимость сложной балансировки запросов на клиенте и сравнивает несколько вариантов решения проблем балансировки, и пула сессий. Еще Александр приводит теоретическое обоснование того, почему те или иные решения должны работать лучше, а также демонстрирует метрики, собранные с помощью тестового приложения, которые подтверждают или опровергают сделанные предположения.

Доклад будет интересен всем тем, кто работает с распределенными базами данными и сконцентрирован на вопросе создания высокопроизводительных приложений, использующих такие БД.

Слайды

Искусство бенчмаркинга распределённых баз данных на примере YDB

Testing

Евгений Иванов (Старший разработчик) рассказал, какие проблемы производительности нашли в популярных бенчмарках Yahoo! Cloud Serving Benchmark (YCSB) и Benchbase TPC-C. Как их исправили и как теперь используют эти бенчмарки, чтобы находить узкие места в YDB и делать оптимизацию. И насколько, согласно этим бенчмаркам, YDB опережает своих основных конкурентов CockroachDB и YugabyteDB, которые тоже являются распределёнными СУБД с открытым исходным кодом.

Доклад будет интересен как разработчикам приложений, которым требуется надёжная СУБД, так и людям, интересующимся распределёнными системами и базами данных.

Слайды

О релизе YDB v23.1

Releases

В вебинаре рассказывается о поддержке изначального сканирования в CDC, атомарной замене индексов, и аудитном логе:

- Устройство Change Data Capture (механизм отслеживания изменения данных в таблицах) и улучшения этой системы в новом релизе. Ильназ Низаметдинов (Руководитель группы).
- Новая функциональность YDB – аудитный лог, которая позволяет отслеживать ключевые действия и события в системе. Андрей Рыков (Старший разработчик).
- Улучшение в конфигурации акторной системы. Александр Крюков (Разработчик)
- Улучшения в форматах передачи данных между этапами выполнения запросов. Виталий Гриднев (Ведущий разработчик).
- Оптимизация производительности YDB – кеширование паттернов графов вычислений. Владислав Кузнецов (Старший разработчик).
- Атомарное переименование вторичных индексов. Даниил Чередник (Старший разработчик).
- Поддержка вторичных индексов в сканирующих запросах. Юлия Сидорина (Разработчик YDB).
- Улучшения в переносе предикатов на чтения из таблиц. Михаил Сурин.

Слайды

Используем YDB: Возможности встроенного веб-интерфейса и CLI

Practice

Антон Коваленко (Технический менеджер) рассказал и показал как запустить нагрузку на кластер с помощью YDB CLI, как найти причины проблем с производительностью запросов, и как с помощью встроенного веб-интерфейса оценить состояние кластера.

Вебинар будет полезен тем, кто работает с YDB, в том числе и разработчикам приложений.

Развертывание YDB в Kubernetes

Practice

Андрей Фомичев (Руководитель YDB), Михаил Бабич (Старший разработчик) и Максим Горбунов (Руководитель группы эксплуатации) рассказали про архитектуру системы и объяснили, как гарантируется надёжное хранение данных. Также спикеры продемонстрировали, как быстро развернуть собственный кластер в Managed Kubernetes® в Yandex Cloud и рассказали о практических рекомендациях по конфигурации узлов.

Вебинар будет полезен SRE-инженерам и разработчикам, которые занимаются развёртыванием и администрированием YDB с помощью Kubernetes® — как в публичных, так и в частных облаках.

Анализ потоковых данных с помощью Yandex Query - интерактивного сервиса виртуализации данных в YDB

Алексей Дмитриев (Технический менеджер) рассказал, что такое обработка потоковых данных, с какими задачами она помогает справляться и как её выполнять с помощью Yandex Query. В практической части вебинара показаны решения нескольких задач с помощью потоковой аналитики в сервисе Yandex Query.

Вебинар будет полезен разработчикам, особенно тем, кто ещё не пользовался потоковой обработкой и хочет разобраться, что это такое.

Видеозаписи 2022

Эволюция акторной системы

Database internals

Существует несколько подходов к созданию эффективных многопоточных приложений на C++. В YDB была выбрана модель акторов, и с нуля создана собственная акторная система. С тех пор прошло более 7 лет, и сегодня акторная система работает на десятках тысяч серверов. Чтобы пройти путь к созданию сложных модульных распределенных систем с помощью модели акторов команде YDB пришлось решить множество проблем.

В докладе Алексей Станкевичус (Руководитель групп разработки распределенного хранилища YDB) рассказал о некоторых из них:

- как совместить интерактивную нагрузку и фоновые задачи в одном приложении;
- как обеспечить гарантии latency и высокую утилизацию;
- как изолировать подсистемы и обойтись без резервирования CPU.

И, конечно, Алексей рассказал, почему выбрали именно модель акторов.

Доклад будет интересен разработчикам, интересующимся высоконагруженными системами обработки данных, асинхронным программированием, оптимизацией, планировщиками CPU.

Слайды

Считаем число просмотров видео для десятков миллионов пользователей в день

Use cases

Обычные продуктовые фичи могут стать нетривиальной задачей на больших масштабах. Иван Соколов (Руководитель службы видеохостинга (Внешний Дзен)) рассказал, как были спроектированы две системы подсчёта пользователей: общее число просмотров видео и количество текущих зрителей live-трансляций.

В докладе Иван рассказывает о решении следующих задач:

- близкое к realtime время обновления счётчиков;
- возможность горизонтального масштабирования;
- отказоустойчивость при выпадении части мощностей.

Доклад будет интересен разработчикам и аналитикам Video on Demand-сервисов.

Слайды

Миллион RPS в YDB: история одного переезда Метрики

Use cases

В Яндекс Метрике существует сборка визитов пользователей на сайте, для этого необходимо хранить историю всех событий и склеивать их друг с другом на лету. Для этого использовалась конвейерная распределенная система, со своим самописным локальным хранилищем и своей логикой репликации и шардирования. По мере роста нагрузки команда, в которой работает Александр Прудаев (Старший разработчик), уперлась в производительность отдельного шарда, при этом продолжать наращивать их количество без принципиальной перестройки архитектуры было крайне болезненно.

Александр в рамках доклада рассказал:

- Почему остановились на YDB, как переезжали, что сломали.
- Как научились работать с таблицей в 40ТБ и 1 миллионом запросов в секунду.
- Как тестировали и масштабировали.

Доклад рассчитан на разработчиков среднего уровня и выше.

Слайды

YDB: мультиверсионность в распределенной базе

Database internals

Андрей Фомичев (Руководитель YDB) рассказал про особенности выполнения распределенных транзакций, и что нужно делать, чтобы их поведение было привычным для пользователя, а также раскрыл особенности реализации `MVCC` в YDB:

- `MVCC` поверх LSM-деревьев.
- Как удалось сделать `MVCC` с консистентными снапшотами в распределенной базе данных.
- Почему выбрали глобальные, а не локальные таймстемпы.

Доклад подойдет всем, кто интересуется распределенными системами, базами данных.

Слайды

Q&A про Open Source YDB и планы на будущее

Overview

Андрей Фомичев (Руководитель YDB), Олег Бондарь (CPO YDB) и Сергей Пучин (Руководитель группы) выступили на Q&A-сессии, поделились планами на будущие и ответили на вопросы слушателей.

Также в докладе:

- Кратко о том, что такое YDB.
- История развития YDB: от первого коммита 10.01.2014 до выхода за пределы Yandex.
- Про выход в Open Source.

Выступление будет интересно тем, кто хочет глубже понять некоторые аспекты реализации и работы YDB.

Дедупликация 5 миллионов событий в секунду на YDB в АппМетрике

Use cases

Доклад Артема Исмагилова (Разработчик Яндекс.Метрики) раскрывает детали перехода АппМетрики на YDB. Из доклада Артема вы узнаете:

- Как была устроена прошлая версия сервиса: физические сервера с состоянием в оперативной памяти, сохраняемым на диск и самодельной репликацией.
- Подходы к реализации новой версии, которые рассматривали.
- Почему выбрали именно YDB для реализации новой версии сервиса.
- С какими трудностями столкнулись и как их преодолели: большой поток событий, необходимость транзакционной обработки событий, удаление старых данных из базы.
- Как уменьшили нагрузку на YDB в 10 раз, добавив фильтр блума в виде отдельной таблицы YDB.
- Что еще предстоит сделать.

Доклад будет интересен разработчикам высоконагруженных бэкендов.

Слайды

Просто о сложном: как работает драйвер распределенной базы данных YDB

Database internals

Драйвер распределенной базы данных существенно отличается от драйверов традиционных (нераспределенных) баз данных. Главная отличительная особенность распределенных баз данных - необходимость работать со множеством нод СУБД. Для равномерной нагрузки на ноды БД в YDB используется как клиентская, так и серверная балансировка. Для баз данных, работающих в режиме 24/7 и допускающих различные сценарии отказа, драйвер должен быть готов к ошибкам разного рода. Это влияет на то, каков должен быть драйвер распределенной базы данных.

В докладе Алексей Мясников (руководитель команды разработки клиентских библиотек YDB) рассказал про опыт разработки драйверов для распределенной БД на разных языках, про проблемы, с которыми сталкивались, и как их решали или митигировали, а также про вынесенные уроки и принятые решения. Доклад будет интересен разработчикам, интересующимся высоконагруженными системами.

Слайды

Интервью с Андреем Фомичевым

Overview

Андрей Фомичев (Руководитель YDB) рассказал о том, как он пришел в разработку баз данных, с чего начиналась YDB, и что предшествовало её появлению.

Open Source трибуна. YDB в Open Source — что это такое?

Overview

Яндекс понимает важность опенсорса, выпускает собственные открытые продукты и поддерживает сообщество. Олег Бондарь (CPO YDB) рассказал о выходе YDB в Open Source, о сообществе платформы, а также о балансе между потребностями внутренних и внешних пользователей.

Доклад будет интересен всем, кто интересуется распределенными базами данных.

Практические примеры использования Change Data Capture в YDB

Practice

Андрей Фомичев (Руководитель YDB), Максим Зиналь (Архитектор YDB), Ильназ Низаметдинов (Руководитель группы), Тимофей Кулин (Разработчик) объяснили, что такое Change Data Capture (CDC), и какие задачи с помощью него можно решить сейчас и в перспективе. Основной упор в вебинаре сделан на практику: продемонстрировано, как с помощью CDC поддерживать актуальное состояние кэша базы данных оперативной памяти приложения. Пример выполнен в виде программы на языке Go, в ней также показаны приёмы работы с YDB Go SDK.

В вебинаре:

- включение CDC для таблицы;
- прослушивание потока изменений;
- работа с CDC, с использованием командного интерпретатора YDB CLI.

Управляемый сервис YDB: настройка, применение, мониторинг

Practice

Применение YDB как управляемого сервиса позволяет разработчикам и архитекторам сосредоточиться на бизнес-задачах, а команда Yandex Cloud возьмёт на себя заботы о надёжности, производительности и ресурсной эффективности баз данных.

В вебинаре Максим Зиналь (Архитектор YDB) показал:

- YDB в целом и сервис YDB в Yandex Cloud.
- Разворачиваем сервисы YDB.
- Методы аутентификации пользователей, создание учётных записей.
- Обзор web-интерфейса консоли Yandex Cloud в части YDB.
- Работа с YDB через клиент командной строки.
- Пример приложения на Java.
- Приложение на Python (serverless).
- Стандартный генератор нагрузки YDB.
- Мониторинг кластера YDB под нагрузкой.
- Масштабирование кластера YDB.

Видеозаписи 2021

Как клиенты облака используют YDB Serverless

Use cases

Антон Коваленко (Технический менеджер) поделился клиентским опытом использования YDB. В докладе:

- Что такое YDB.
- YDB как Serverless-платформа.
- За что клиенты выбирают YDB.

Слайды

Распределенная трассировка с помощью Jaeger и YDB. Опыт Auto.ru и Yandex Cloud.

Use cases

Мы активно используем Jaeger как инструмент распределенной трассировки, и при росте нагрузки встал вопрос эффективности хранения и обработки данных. В докладе мы расскажем, как выбирали базу для хранения трейсов Jaeger и про дальнейший опыт эксплуатации Jaeger и YDB в Auto.ru и Yandex Cloud. Решение стало популярным внутри Яндекса, и мы выпустили Jaeger-драйвер для YDB в Open Source. Появление YDB Serverless дало пользователям возможность сэкономить, и мы хотим поделиться результатами тестов Jaeger с YDB Serverless.

Слайды

Мультиарендный подход Яндекса к построению инфраструктуры работы с данными

Database internals

Времена, когда для экземпляра базы данных выделялся отдельный компьютер, давно прошли. Сейчас повсюду управляемые решения, поднимающие необходимые процессы в виртуальных машинах. Для вычислений применяется еще более прогрессивный подход — «бессерверные вычисления», например AWS Lambda или Yandex Cloud Functions. И уж совсем на острие прогресса находятся бессерверные БД.

Андрей Фомичев (Руководитель YDB) рассказал о бессерверных решениях, которые еще до всеобщего хайпа стали популярны в Яндексе и по-прежнему используются для хранения и обработки данных.

Слайды

Большой таблеточный взрыв

Overview

Современная распределённая БД, как и сама Вселенная, состоит из мельчайших частиц. Из частиц, которые мы называем таблетками, строится петабайтная вселенная данных Yandex Database. Андрей Фомичев (Руководитель YDB) рассказал о том, какую задачу решают таблетки, как они обеспечивают согласованные изменения данных и отказоустойчивое хранение, а также как мечты помогают нам создавать новые технологии космического масштаба, не ограничиваясь стандартными решениями.

Миграция приложения с PostgreSQL на Yandex YDB Serverless. Производительность, стоимость, риски.

Use cases Practice

В исследовании, которое провела команда Yandex YDB, рассмотрены аспекты миграции E-commerce приложения, изначально разработанного для PostgreSQL, на Yandex YDB. Основная тема исследования — сравнение производительности и стоимости решений на PostgreSQL и YDB.

В докладе показано:

- Что нового в Cloud Functions. Глеб Борисов.
- Как клиенты облака используют Yandex Database Serverless. Антон Коваленко (Технический менеджер).
- Миграция приложения с PostgreSQL на Yandex Database Serverless. Производительность, стоимость, риски. Александр Смирнов (Технический эксперт).

Доклад будет полезен разработчикам, архитекторам, инженерам сопровождения.

Слайды

Практический пример использования serverless: интеграция сервисов Yandex Cloud с голосовыми технологиями компании Voximplant

Use cases

В докладе показано, как интегрировать сервисы Yandex Cloud с голосовыми технологиями компании Voximplant.

В докладе подробно описаны следующие шаги:

- Интеграция Voximplant Kit с Yandex SpeechKit — сервисом распознавания и синтеза речи.
- Настройка и автоматизация обработки диалога с помощью Voximplant Kit.
- CRM-система, построенная с помощью serverless API Gateway, Yandex Cloud Functions и YDB (serverless).
- Пример приложения на Go.

Видеозаписи 2020

Serverless: публичный запуск YDB

Overview

Сервис YDB вышел в публичный доступ в Яндекс Облаке. Теперь можно выбрать режим работы с выделенными серверами или бессерверный режим.

В докладе:

- Выход сервиса YDB в публичный доступ.
- Запуск нового продукта YDB Serverless.
- Новые возможности YDB.
- Ценообразование YDB и YDB Serverless.

Новости YDB — обзор новых возможностей

Overview

Антон Коваленко (Технический менеджер) сделал обзор новых возможностей БД: работа с JSON, системные представления и nodeJS SDK. Раньше пользователи Яндекс.Облака могли работать с Yandex Database только из виртуальных машин на платформе. Теперь это возможно через интернет.

Из вебинара вы узнаете про:

- Публикацию драйвера для системы сбора и анализа трейсов Jäger в Open Source.
- Расширение возможностей для работы с JSON из YQL.
- Системные представления для диагностики производительности запросов.
- Публикацию в Open Source JavaScript SDK.

Ссылки на материалы:

- Jaeger Plugin к YDB доступен на https://github
- Yandex Database Public Preview
- Python SDK
- Go SDK
- Java SDK
- JavaScript SDK

Презентация

Видеозаписи 2019

YDB: распределенные запросы в облаках

Database internals

Сергей Пучин (Руководитель группы) рассказал про применяемую модель транзакций и уровни изоляции, особенности SQL-диалекта Yandex Query Language (YQL), параметризацию и подготовку запросов, многошаговые транзакции и механизм оптимистичных блокировок. Также в докладе затронуты общие вопросы эффективного выполнения запросов в распределенных базах данных и рассмотрены основные факторы, влияющие на производительность запросов, и стандартные практики при работе с YDB.

В докладе разобраны сценарии использования YDB под высокими нагрузками внутри Яндекса:

- Турбо-страницы. YDB используется для хранения метаданных.
- Яндекс.Коллекции. YDB используется для хранения истории о рекомендациях пользователям.
- Яндекс Облако. YDB является основным хранилищем метаданных для системных сервисов и сервисов управления пользовательскими данными: Yandex Object Store, Yandex Message Queue.

Слайды

Распределенные транзакции в YDB

Database internals

В докладе рассмотрен алгоритм планирования транзакций, лежащий в основе транзакционной системы YDB. Сделан разбор сущностей, участвующих в транзакциях. На примере распространенной задачи рассмотрена реализация транзакций с применением двухфазного коммита и детерминистических транзакций.

Из доклада вы узнаете:

- Что такое таблетки и как они устроены.
- Как реализован ACID механизм.
- Какие есть уровни изоляции.
- Как работают распределенные транзакции.

Слайды

YDB — как мы обеспечиваем отказоустойчивость

Database internals

YDB – горизонтально масштабируемая геораспределенная отказоустойчивая СУБД, выдерживающая отказ дисков, серверов, стоек и дата-центров без нарушения консистентности. Для обеспечения отказоустойчивости применяется собственный алгоритм достижения распределенного консенсуса, а также ряд технических решений, которые детально рассмотрены в докладе.

Владислав Кузнецов (Старший разработчик) рассказал о том, как YDB хранит данные с избыточностью и как YDB решает задачу распределенного консенсуса.

Слайды

Распределенные транзакции в YDB

Database internals

В докладе рассмотрен алгоритм планирования транзакций, на котором основана транзакционная система YDB. Вы узнаете, какие сущности участвуют в транзакциях, кто устанавливает глобальный порядок транзакций и как достигается атомарность транзакций, надежность и изоляция высокого уровня.

На примере общей проблемы показана реализация транзакций с использованием двухфазного коммита и детерминированных транзакций.

Слайды

YDB — эффективная альтернатива традиционным noSQL-решениям

Overview

Андрей Фомичев (Руководитель YDB) рассказал, как и зачем была создана YDB, чем она отличается от других БД и для каких задач она лучше всего подходит.

В докладе подробно разобраны следующие свойства YDB:

- Автоматический split/merge шардов.
- Автоматическое восстановление после сбоев за время обнаружения отказа.
- Синхронная репликация данных, в том числе в геораспределенной конфигурации данных.
- Механизм serializable-транзакций между записями базы данных.

Слайды

YDB at Scale: опыт применения в высоконагруженных сервисах Яндекса

Use cases

Представители Auto.ru, Яндекс.Репетитора, Алисы и Condé Nast рассказали, почему они выбрали YDB и как эта СУБД помогает развивать их продукты.

Вы узнаете:

- Как YDB хранит гигабайты данных умных устройств.
- Почему разработчики Алисы выбрали YDB для хранения логов.
- Какие преимущества есть у YDB перед Cassandra и MongoDB.

Слайды

YDB — как выжать 10K IOPS из HDD и вставить в таблицу 50K записей на одном ядре

Use cases

Из доклада вы узнаете об успешном использовании YDB в качестве бэкэнда для распределенной трассировки Jaeger от представителей Auto.ru и Яндекс.Недвижимости, а также об архитектуре распределенного сетевого хранилища в YDB.

Слайды

YDB: Distributed SQL база данных Яндекса

Database internals

Сергей Пучин (Руководитель группы) рассказал об основных моментах, связанных с выполнением распределенных запросов в YDB:

- Модель транзакций и уровни изоляции.
- Особенности SQL-диалекта Yandex Query Language (YQL).
- Многошаговые транзакции и механизм оптимистичных блокировок.
- Эффективное выполнение запросов к распределенным БД в целом.
- Основные факторы, влияющие на производительность запросов.
- Стандартные практики работы с YDB, в том числе инструменты для разработчика.

Доклад будет интересен тем, кто хочет глубже погрузиться в процессы работы YDB и узнать как клиентские приложения взаимодействуют с YDB, и как работает система распределенных транзакций YDB.

Слайды

Опыт использования YDB на примере продуктов Яндекса

Use cases

Олег Бондарь (CPO YDB) поделился опытом использования YDB на примере продуктов Яндекса.

В вебинаре:

- Что такое YDB.
- Как выглядит YDB.
- Примеры использования YDB в Яндексе.
- Тестирование производительности (YCSB).

Вебинар будет интересен тем, кто хочет познакомиться с YDB.

Презентация

Видеозаписи 2018

Три доклада о YDB с about:cloud

Overview Use cases Database internals

1. YDB: платформа распределенных систем хранения данных, критичных к задержкам. Рассказ о YDB, как о платформе, на которой можно строить различные системы хранения и обработки данных. На платформе YDB построены:

- LogBroker (аналог Apache Kafka).
- Real-time Map Reduce (обработка потоков данных).
- Хранение временных рядов в системе мониторинга.

1. YDB: Distributed SQL база данных. Рассказ о YDB как базе данных. Из доклада вы узнаете:

- Что такое "Таблетка" и какими свойствами она обладает.
- Как работают распределенные транзакции в YDB и что такое детерминированные транзакции.
- Что такое YQL и чем он отличается от SQL.

1. YDB: сетевое блочное устройство. Рассказ о Network Block Store — сервисе виртуальных дисков, на которых работают все виртуальные машины Yandex Cloud. В докладе подробно разобрано:

- Устройство File System (Файл/директория, Операции с метаданными, Индекс блоков).
- Устройство Block Device (Хранение блоков данных, интерфейс взаимодействия).
- Работа с QEMU/KVM (Virtio-очереди, Виртуализация процессора, режим паравиртуализации).

Доклады будут полезны всем, кто хочет познакомиться с YDB.

Видеозаписи 2017

Как проверить систему, не запуская ее

Testing

Системы, которые мы разрабатываем, становятся сложнее с каждым днем. И кажется, что нет спасения от вездесущей сложности, которая проникает во все. Один из аспектов этой сложности — конфигурация. С одной стороны, конфигурация сильно влияет на стабильность и доступность системы, с другой — проверке ее корректности уделяется очень мало внимания. В своём докладе Андрей Сатарин рассказывает о том, как тестируются конфигурации и насколько это полезно.

Этот доклад будет интересен всем, кто хочет узнать простой способ увеличения стабильности и доступности системы в продакшене.

Слайды

Статьи

В этом разделе собраны статьи о YDB:

- Статьи 2026
- Статьи 2025
- Статьи 2024
- Статьи 2023
- Статьи 2021

Материалы разделены по категориям и размечены тегами:

Overview	– обзорные материалы, знакомящие с YDB и технологиями, которые применяются в ней.
Use cases	– кейсы использования YDB.
Practice	– практика использования YDB.
Database internals	– подробный разбор внутренней реализации YDB или её отдельных частей и механизмов.
Releases	– обзор новых возможностей, вышедших релизов YDB.
Testing	– кейсы тестирования производительности YDB и сравнения с другими одноклассовыми СУБД.
General	– материалы общего направления.

Статьи 2026

Миллиард записей и 8 Марта: как YDB спас праздник

Practice



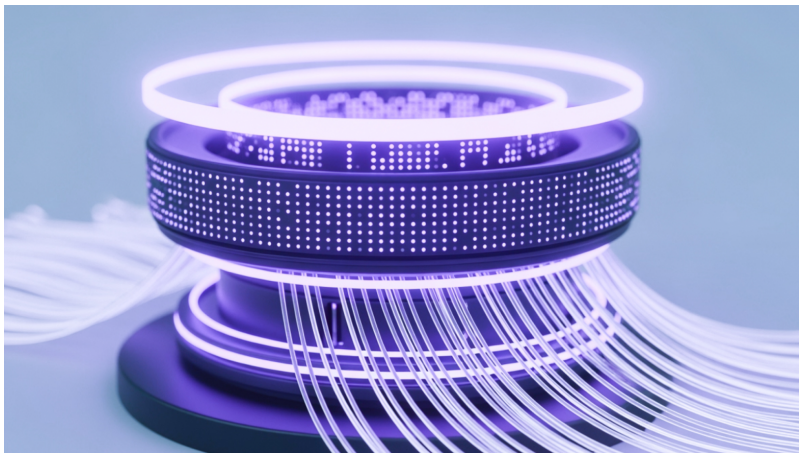
В статье рассматривается процесс переработки системы синхронизации складских остатков и цен в ритейле Яндекс Еды: от порядка миллиарда строк на старте проекта до нескольких миллиардов записей о ценах и стоках на проде, и переход от кеша на PostgreSQL с периодическими тяжёлыми JSON-выгрузками от партнёров к распределённому хранилищу на YDB с топиками брокера сообщений, командами PATCH/SAVE и protobuf в колонке данных. Автор показывает, почему полные JSON-снимки и перезапись больших фрагментов в базе перестали масштабироваться, и как их заменили на поток изменений и дельты: зачем вместо широких таблиц и JSON взяли protobuf, как батчировать сообщения для брокера, как устроено наложение патчей и какие сроки актуализации данных требуют витрина, поиск и операционные сценарии вроде заказов к праздникам.

Статья будет полезна для специалистов, которые проектируют или эксплуатируют высоконагруженные системы с большим объёмом данных, а также сталкиваются с миграцией с монолита и тяжёлых SQL/JSON-решений к распределённым хранилищам, таким как YDB. В материале показано сравнение подходов «до» и «после» и обоснование ключевых инженерных решений.

Статьи 2025

Опыт миграции DWH и сложности, которые при этом возникают

Practice

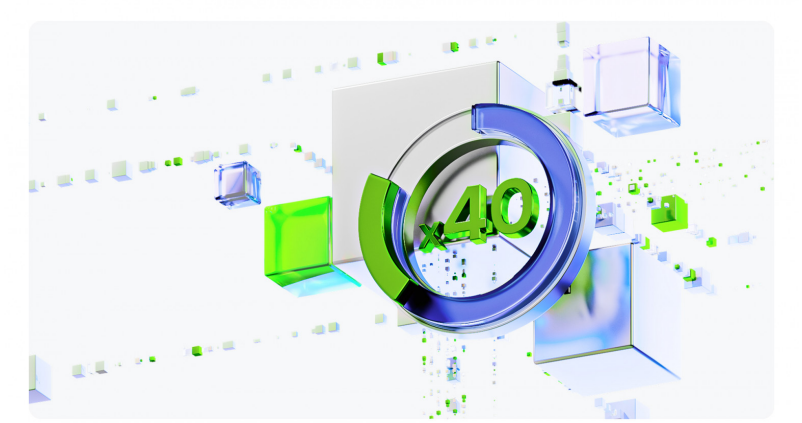


В статье рассматривается процесс миграции корпоративного хранилища данных (DWH) и одновременного построения на его основе двух высоконагруженных систем: real-time фиче-стора и системы управленческой отчетности. Особое внимание уделяется архитектурным решениям и сложностям, с которыми столкнулась команда: от реорганизации потоков данных для устранения расхождений до выполнения строгих требований к производительности новых сервисов.

Статья будет полезна для специалистов, которые сталкиваются с необходимостью миграции больших объемов данных, построения real-time аналитических платформ и выбора оптимальной архитектуры для высоконагруженных сервисов. В ней представлено сравнение архитектур «до» и «после» миграции, детальное описание проблем старого подхода и обоснование выбора новых инструментов в экосистеме Yandex Cloud для их решения.

Как мы ускорили заливку данных в YDB в 40 раз

Practice



В статье рассматривается реальный кейс по оптимизации производительности YDB, используемой в качестве хранилища для ML Online Feature Store. Автор уделяет особое внимание практическим методам ускорения массовой вставки данных и устранению ее влияния на скорость чтения.

Материал предназначен для специалистов, которые сталкиваются с необходимостью организации быстрой загрузки больших объемов данных в YDB в высоконагруженных системах, требующих одновременно низкой задержки на чтение. В статье представлены три ключевые оптимизации: снижение числа партиций, удаление избыточного глобального индекса и предварительная сортировка данных перед вставкой. Эти шаги позволили ускорить загрузку данных в 40 раз и стабилизировать latency чтения.

Запустили векторный поиск в YDB: рассказываем, как он работает

Database internals

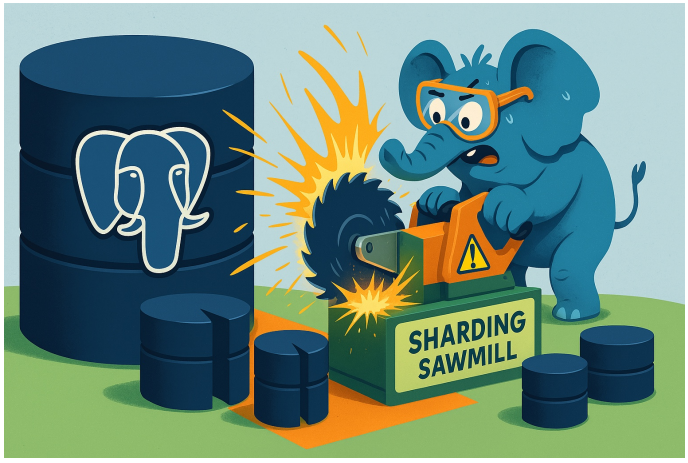


В статье рассказывается про две версии векторного поиска в YDB — точный и приближённый. Приближённый поиск может работать с миллиардами векторов, если использовать векторный индекс. Такая технология есть у небольшого количества технологических компаний в мире.

Материал будет полезен всем, кто интересуется современными подходами к организации векторного поиска в больших данных.

Шардированный не значит распределённый: что важно знать, когда PostgreSQL становится мало

Practice



В данной статье рассматривается организация распределенных транзакций, шардирование данных и использование двухфазного коммита. Особое внимание уделяется важному аспекту: двухфазный коммит обеспечивает лишь атомарность операций, но не гарантирует изоляцию транзакций, что может создавать определенные ожидания у разработчиков.

Материал предназначен для специалистов, которые сталкиваются с необходимостью масштабирования систем за пределы возможностей одиночной базы данных (например, PostgreSQL) и перехода к архитектуре с шардированием данных. В статье представлены практические рекомендации по выбору архитектурных решений и разъяснение ключевых концепций распределенных транзакций.

Релиз диалекта YDB для SQLAlchemy: как мы сделали интеграцию с Apache Superset

General

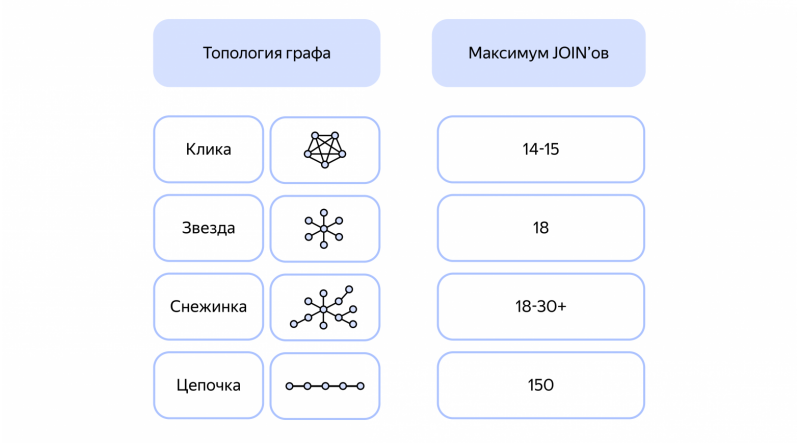


В данной статье описывается процесс разработки Python-клиента для взаимодействия с распределенной базой данных YDB. Учитывая богатые возможности Python в области работы с базами данных, включая стандартизированный интерфейс DBAPI (PEP-249) и множество ORM, команда разработчиков реализовала полноценную интеграцию YDB с аналитической платформой Apache Superset.

Статья подробно освещает технические аспекты интеграции, особенности взаимодействия с существующей экосистемой Python и решения возникающих при этом архитектурных задач.

Стоимостной оптимизатор: сердце гибридной базы данных YDB

Database internals

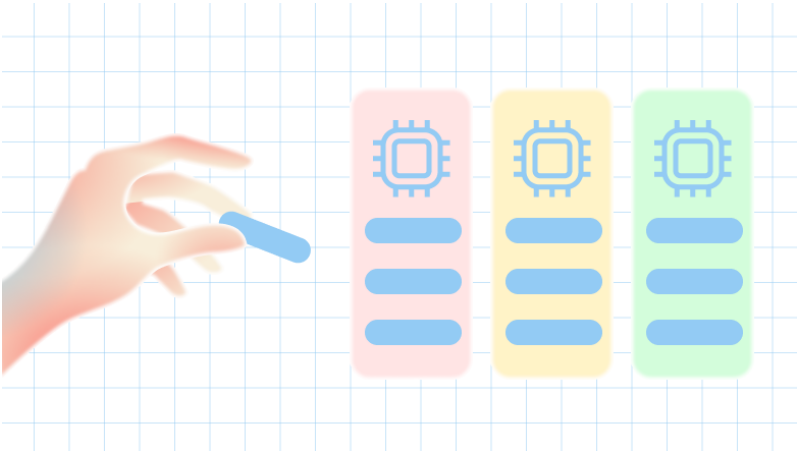


В статье рассматривается назначение, принципы работы и внутренняя реализация стоимостного (cost-based) оптимизатора запросов в гибридной базе данных YDB. Павел Велихов (Руководитель команды оптимизатора запросов) делится историей разработки собственной СУБД в Яндексе, объясняет отличия OLTP- и OLAP-нагрузок и обосновывает необходимость сложного оптимизатора для эффективной обработки аналитических запросов. Подробно описываются алгоритмы перебора и выбора оптимальных планов, такие как динамическое программирование и гиперграфовый алгоритм DPhyp, а также роль оценки кардинальности и функции стоимости. Материал иллюстрирует, как современные подходы позволяют YDB обрабатывать масштабные и сложные SQL-запросы, обеспечивая высокую производительность даже на больших объемах данных.

Статья предназначена для специалистов в области баз данных, разработчиков, девопсов, а также архитекторов IT-инфраструктуры.

Почему сложно разработать OLAP-базу данных, если у тебя уже есть OLTP

Database internals



В статье объясняются основные различия между OLTP- и OLAP-нагрузками и описываются архитектурные вызовы, возникающие при создании гибридной (HTAP) базы данных на примере YDB. Алексей Дмитриев (Технический менеджер) подробно рассматривает аспекты эффективного хранения, передачи и обработки данных для обоих типов нагрузки, а также различные подходы к оптимизации запросов и управлению вычислительными ресурсами. В материале приводятся примеры решений и компромиссов, к которым приходят разработчики в процессе построения HTAP-систем, а также даются уроки и наблюдения, приобретённые командой YDB в ходе развития продукта.

Статья будет полезной для инженеров и архитекторов, интересующихся современными подходами к проектированию гибридных баз данных.

Kafka API для работы с потоками данных Yandex Data Streams — топиками YDB

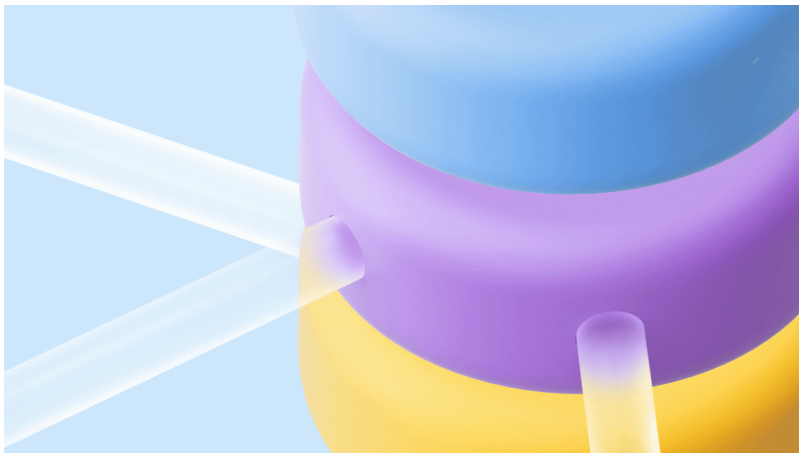
Practice

Статья представляет собой подробный туториал по использованию Kafka API для работы с потоками данных Yandex Data Streams (топиками YDB) в облачной инфраструктуре Yandex Cloud. Елена Калинина (Технический менеджер) даёт пошаговую инструкцию по созданию необходимых облачных ресурсов — базы данных, потоков, S3-бакета, сервисных аккаунтов и настройке доступа, а также приводит примеры настройки и использования Kafka CLI и Kafka Connect для записи, чтения и выгрузки сообщений из потока данных в облачное хранилище. Рассматриваются практические аспекты безопасности и даны рекомендации по хранению конфиденциальных данных. Материал позволяет быстро начать работу с потоками YDB через привычные Kafka-инструменты, облегчая миграцию с Apache Kafka и интеграцию с существующими бизнес-процессами.

Статья предназначена для DevOps-инженеров, специалистов по облачной инфраструктуре, разработчиков, а также всех, кто работает с потоковыми данными и интеграциями на основе Kafka.

Как я сделал PR на 14K строк в проект YDB будучи студентом

Database internals



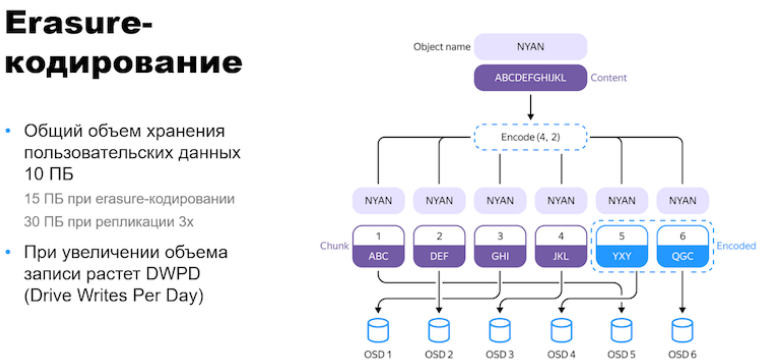
В статье Павел Орлов (студент) рассказывает о практическом опыте крупного вклада в проект YDB — переводе парсера языка запросов YQL с ANTLR3 на ANTLR4. Автор объясняет, почему задача была технически сложной, какие трудности возникли при миграции инструментов и шаблонов, а также как удалось аккуратно внедрить изменения, не нарушая существующую функциональность системы. Материал знакомит с принципами парсинга, особенностями работы ANTLR и процессом интеграции нового парсера. Также подчеркнута значимость грамотной работы с open source-проектами для профессионального роста.

Статья предназначена для разработчиков, специалистов по базам данных, студентов и начинающих инженеров, интересующихся open source и инструментами синтаксического анализа.

Статьи 2024

Шины данных для очень больших инсталляций, или Почему большим компаниям сложно с опенсорсом

Database internals



В статье рассматриваются особенности эксплуатации опенсорсных шин данных (на примере Apache Kafka) в очень больших производственных инсталляциях, характерных для крупных предприятий. Автор приводит подробное сравнение с YDB Topics, анализируя вопросы масштабируемости, отказоустойчивости, стоимости оборудования и операционного управления. Описаны архитектурные подходы к обеспечению высокой доступности, оптимизации TCO, автоматизации, балансировке нагрузки и резервированию ресурсов, а также выделены проблемы open source-решений при работе на масштабах enterprise. Материал будет полезен тем, кто проектирует инфраструктуру передачи данных в крупных компаниях и сравнивает модели open source и кастомных корпоративных платформ.

Статья предназначена для технических менеджеров, архитекторов корпоративных ИТ-систем, системных администраторов, DevOps-инженеров и специалистов по эксплуатации крупных распределённых систем.

Стоит ли бояться serializable-транзакций больше, чем труднонаходимых багов?

General

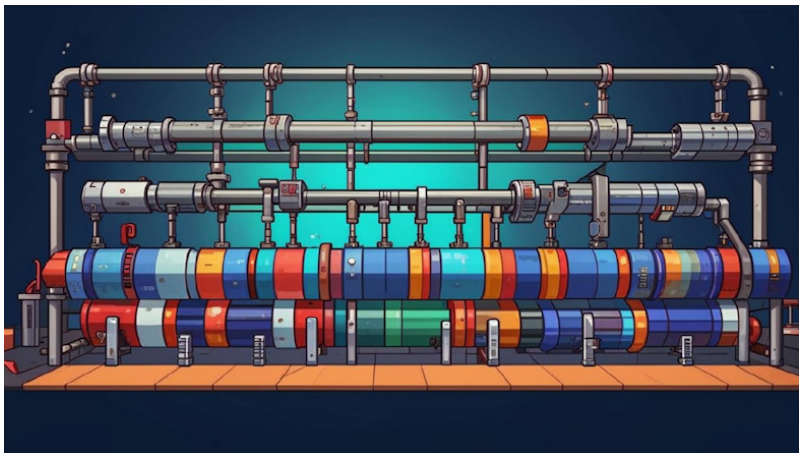


Статья рассматривает влияние разных уровней изоляции транзакций в базах данных на корректность приложений и безопасность, уделяя особое внимание режиму serializable. Автор анализирует реальные кейсы возникновения ошибок и уязвимостей, связанных со слабыми уровнями изоляции, и сравнивает их с мифом о значительных потерях производительности при использовании строгой изоляции. Делается вывод, что использование serializable-транзакций по умолчанию часто оправдано, поскольку профилактика concurrency-багов важнее незначительного снижения производительности, а современные СУБД демонстрируют устойчивый уровень производительности даже на строгих режимах.

Статья предназначена для специалистов по базам данных, разработчиков, архитекторов ПО и администраторов, использующих транзакционные СУБД.

Как мы повышали производительность очереди сообщений

Practice



В статье подробно описывается эволюция производительности очереди сообщений YDB Topics. Автор сравнивает архитектурные особенности YDB Topics с Apache Kafka и Pulsar, анализирует причины отказа от Kafka, а также рассказывает о технических инструментах нагрузочного тестирования и пути к оптимизации узких мест. Приведены конкретные результаты производительности YDB Topics после проведённых оптимизаций, а также рассказано о внедрении Kafka API для максимальной совместимости с существующим софтом и облачными сценариями использования. Статья иллюстрирует комплексный подход к обеспечению высокой производительности и надёжности потоковых очередей на масштабах интерпрайз-уровня.

Статья предназначена для инженеров по высоконагруженным системам, DevOps- и SRE-специалистов, архитекторов распределённых систем, а также тех, кто интересуется оптимизацией и эксплуатацией очередей сообщений.

Миграции схемы данных YDB с Flyway и распределенные блокировки

Practice



Статья посвящена поддержке и особенностям работы Flyway для управления миграциями схемы данных в YDB. Описывается процесс настройки Flyway с YDB как через Java/Gradle, так и через CLI, а также принцип работы основных команд: baseline, migrate, info, validate, repair и clean. Особое внимание уделено механизму распределённых блокировок для предотвращения конкурентного применения миграций — показано, как это реализовано в YDB по сравнению с другими СУБД.

Статья предназначена для разработчиков, инженеров данных и специалистов по эксплуатации СУБД, использующих YDB и инструменты автоматизации миграций.

database/sql: плохой, хороший, злой

Database internals



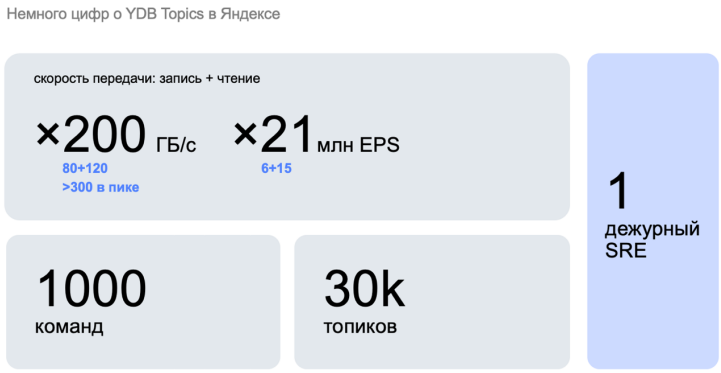
В статье проведён подробный разбор эволюции стандартной библиотеки Go для работы с базами данных (database/sql), особенностей её применения в распределённых системах, таких как YDB, и трудностей, возникающих при реализации и эксплуатации соответствующих драйверов. Рассматриваются архитектурные решения, внедрённые в YDB, а также специфические проблемы — управление соединениями, ретраи, контексты, параметры запросов, обработка ошибок,

совместимость с концепциями SQL и Go. Автор делится опытом создания и доработки database/sql-драйвера для YDB, описывает ретроспективу изменений в Go и связанные с этим инженерные вызовы, а также демонстрирует особенности интеграции и инструменты отладки. Материал полезен для всех, кто внедряет или сопровождает СУБД в экосистеме Go и интересуется лучшими практиками подключения к распределённым базам.

Статья предназначена для Go-разработчиков, инженеров данных, архитекторов распределённых систем и разработчиков драйверов СУБД.

Как Яндекс создал свою шину данных, чтобы передавать сотни гигабайт в секунду

Database internals



Статья рассказывает о причинах появления и архитектуре собственной шины данных Яндекса — YDB Topics — опенсорсного решения, способного передавать сотни гигабайт в секунду. На фоне проблем масштабирования и эксплуатации при использовании Apache Kafka, команда Яндекса разработала модульную шину поверх YDB с поддержкой высоких гарантий целостности, автоматического масштабирования и эффективного управления правами доступа. Подробно описаны архитектурные решения, отличия от Kafka и Pulsar, сценарии использования (федерация, кросс-ДЦ), а также особенности, такие как Erasure-кодирование для экономии места и масштабируемость с автоматизированным управлением.

Статья предназначена для инженеров по инфраструктуре, DevOps, архитекторов, а также всех, кто интересуется построением высоконагруженных систем передачи данных и open source-решениями в enterprise.

Поддержка диалекта YDB в инструменте миграции Liquibase

Practice

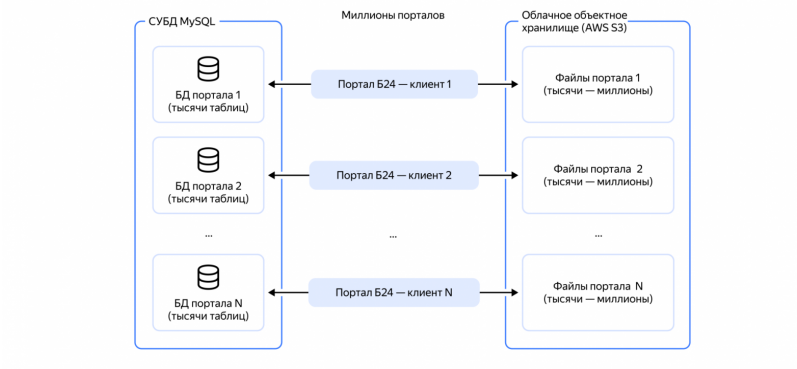


В статье показано, как использовать мощный инструмент миграций Liquibase для работы с базой данных YDB посредством YQL. Автор описывает нюансы интеграции, поддерживаемые SQL-конструкции, возможные ограничения и совместимость с Liquibase, а также приводит пошаговые примеры настройки, исполнения и эволюции схемы базы данных. Приводятся примеры создания и изменения таблиц, работы с индексами, загрузки данных, и применения нативных YQL-инструкций.

Статья предназначена для разработчиков, инженеров данных и DevOps-специалистов, использующих YDB и Liquibase.

Перенести проверенную схему бэкапа больших данных из S3 в Yandex Cloud: опыт Битрикс24

Use cases

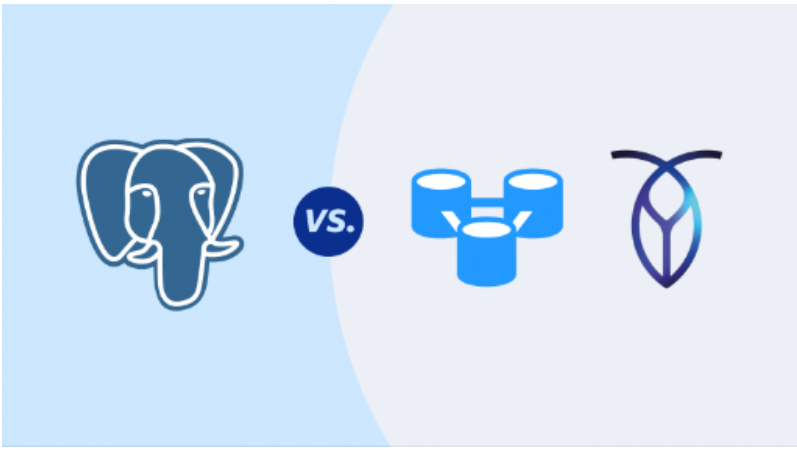


В статье на реальном кейсе компании Битрикс24 описан опыт миграции масштабируемой схемы резервного копирования и инкрементальной репликации файлов с AWS S3 в облако Yandex Cloud. Автор поэтапно разбирает эволюцию архитектуры, методы инкрементального и realtime-бэкапа, подходы к отложенному удалению файлов и реализации асинхронной обработки событий с использованием очередей, воркеров и serverless-технологий. Особое внимание уделено переходу на YDB с режимом совместимости с DynamoDB, мониторингу системы и автоматическому масштабированию ресурсов. Статья содержит практические советы по сохранению надёжности, минимизации ручного труда и оптимизации стоимости хранения больших объёмов данных при переходе между облачными провайдерами.

Статья предназначена для инженеров данных, архитекторов облачных решений, DevOps, технических руководителей и специалистов по эксплуатации масштабируемых систем хранения и резервного копирования.

Когда одного Postgres'а мало: сравнение производительности PostgreSQL и распределенных СУБД

Testing



В статье проведено практическое сравнение производительности PostgreSQL и распределённых СУБД YDB и CockroachDB на троих одинаковых физических серверах с использованием бенчмарка TPC-C при различных сценариях отказоустойчивости. Эксперимент подтверждает, что PostgreSQL крайне эффективна на одиночном сервере, но наличие синхронной репликации становится серьёзным ограничением по производительности и задержкам при увеличении числа нод. Распределённые СУБД показывают чуть ниже tpmC на такой же инфраструктуре, но демонстрируют лучшую отказоустойчивость, масштабируемость и предсказуемые задержки транзакций. В статье также приведён разбор инженерных "узких мест", подробные параметры тестов и практические рекомендации для выбора архитектуры.

Статья предназначена для специалистов по базам данных, архитекторов высоконагруженных систем, инженеров и администраторов, принимающих решения о выборе и СУБД, а также всех, кто интересуется сравнением PostgreSQL и распределённых СУБД на практике.

Обновление кешей сервисов в реальном времени с помощью YDB CDC на примере Yandex Monitoring

Database internals

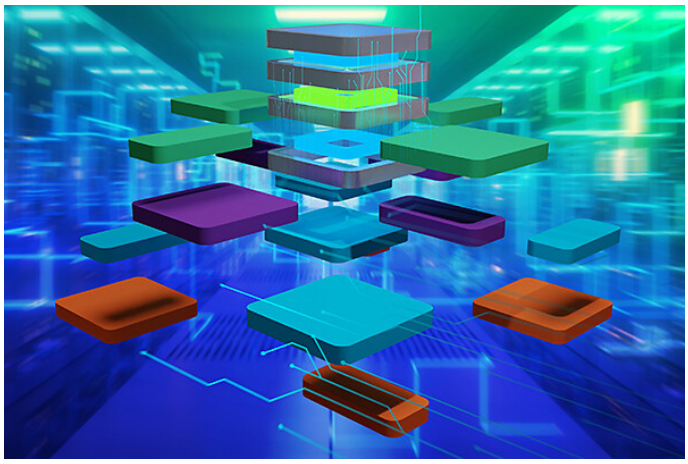


Летом 2023 года Егор Литвиненко (старший разработчик Yandex Observability Platform) рассказывал на Saint Highload в Санкт-Петербурге о пути внедрения YDB CDC для обновления данных в сервисах, чтобы решить проблему инвалидации кешей. В этой статье будет вся история внедрения с теорией, вопросами, ответами, ошибками, о которых он говорил на выступлении.

Статьи 2023

Открытые системы (электронный журнал). СУБД. Как открытый исходный код меняет современные СУБД

Overview



У YDB путь проектирования изначально начался с учетом масштабных надежных инсталляций, что позволило СУБД быстро занять одну из ключевых ролей во внутренней инфраструктуре «Яндекса» в транзакционных сценариях использования. Затем YDB доросла до публичного управляемого сервиса в Yandex Cloud, позже обзавелась бессерверным (serverless) режимом работы. Сегодня СУБД YDB переживает следующий этап развития — выход в Open Source и адаптация к новым реалиям.

Из статьи вы узнаете:

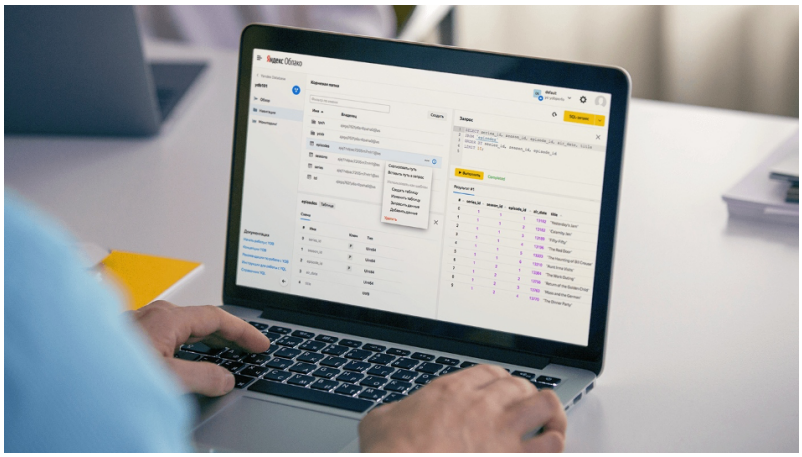
- Почему вообще YDB пошла в Open Source?
- Какими возможностями обладает YDB?
- Как развивался проект?
- В каких сценариях YDB показывает себя сопоставимо с конкурентами, а в каких их превосходит?

Ознакомиться со статьёй можно на сайте OSP.

Статьи 2021

Бессерверная альтернатива традиционным базам данных

Overview



Современная распределенная СУБД должна уметь поддерживать различные типы нагрузки, удовлетворяя запросы совершенно разных пользователей. СУБД YDB позволяет не только хранить петабайты данных, поддерживать обработку миллионов запросов в секунду, но и предоставляет режим бессерверных вычислений. Эта платформа дает возможность обслуживать проекты с различными типами нагрузки: ключ-значение, традиционные веб-приложения на основе реляционной базы, а также документоориентированные базы данных.

В статье изложена:

- История появления YDB.
- YDB с точки зрения пользователя.
- Архитектура YDB.
- Бессерверный режим в YDB.

Погружение в Serverless. Рождение YDB

Database internals



Продолжение беседы с разработчиками экосистемы сервисов Serverless. В начале нашего путешествия Глеб Борисов описал ситуацию с Yandex Cloud Function, затем Данил Ошеров погрузил нас в мир протокола S3 и сервиса Object Storage. В этой статье Андрей Фомичев(Руководитель YDB) поделился подробностями о NewSQL.

Мультиарендный подход Яндекса к построению инфраструктуры работы с данными

Overview

Времена, когда для экземпляра базы данных выделялся отдельный компьютер, давно прошли. Сейчас повсюду управляемые решения, поднимающие необходимые процессы в виртуальных машинах. Для вычислений применяется еще более прогрессивный подход — «бессерверные вычисления», например AWS Lambda или Yandex Cloud Functions. И уж совсем на острие прогресса находятся бессерверные БД.

В статье Андрей Фомичев(Руководитель YDB) рассказал о бессерверных решениях, которые еще до всеобщего хайпа стали популярны в Яндексе и по-прежнему используются для хранения и обработки данных.

Готовим с serverless. Голосовой сервис записи к врачу и регистрации в поликлинике

Use cases



Какой serverless-стек нужен, из чего состоит сценарий и как может быть устроена система CRM на стороне Yandex Cloud? Коммуникационная платформа Voximplant и Yandex Cloud подготовили рецепт голосового сервиса регистрации и записи на прием к врачу в поликлинику. Впрочем, им можно воспользоваться и для других похожих serverless-задач.

В статье:

- Устройство визуального конструктора Voximplant Kit.
- Разбор сценария входящего звонка.
- Устройство CRM-системы на стороне Yandex за API.
- Создание Yandex Database Serverless в Yandex Cloud.

Бессерверная альтернатива традиционным базам данных

Overview



Бессерверные вычисления стали следующим шагом к снижению «налога» на инфраструктуру, который вынуждены платить пользователи облачных сервисов, однако на рынке сегодня мало баз данных для этой экосистемы, и задача YDB — восполнить этот пробел.

В статье:

- История развития распределенных баз данных.
- О Yandex Database.
- Архитектура YDB.
- Бессерверный режим в YDB.
- YDB с точки зрения пользователя.

Подкасты

Подкасты разделены по категориям и размечены тегами:

Overview	– обзорные материалы, знакомящие с YDB и технологиями, которые применяются в ней.
Use cases	– кейсы использования YDB.
Practice	– практика использования YDB.
Database internals	– подробный разбор внутренней реализации YDB или её отдельных частей и механизмов.
Releases	– обзор новых возможностей, вышедших релизов YDB.
Testing	– кейсы тестирования производительности YDB и сравнения с другими одноклассовыми СУБД.
General	– материалы общего направления.

sysadmins №54. YDB

General

В подкасте linkmeup Антон Коваленко (Технический менеджер) представляет обзор современной распределенной СУБД.

Выпуск подкаста освещает предпосылки создания YDB, особенности архитектуры Distributed SQL, текущие кейсы использования системы и её позиционирование на рынке СУБД.

Особое внимание уделяется техническим аспектам реализации и неожиданным результатам разработки, включая создание распределенной файловой системы.

Материал предназначен для специалистов, интересующихся современными тенденциями в области управления данными и распределенных систем.

sysadmins №54 (26 февраля 2025). YDB

Список изменений YDB Server

Версия 26.3

Релиз кандидат 26.3.1.16

Дата выхода: 18.09.26

Функциональность

- Для колоночных таблиц доступны экспорт и импорт резервных копий, включая S3-совместимое хранилище.
- Колонки колоночных таблиц поддерживают словарное кодирование. Используйте `ENCODING(DICT)` для значений с низкой кардинальностью.
- Для колоночных таблиц включены локальные min_max-индексы. Они пропускают фрагменты данных вне диапазона запроса, сокращая объём чтения. Используйте `ADD INDEX ... LOCAL USING min_max` для применения к колонке.
- Добавлена декомиссия групп хранения посредством использования виртуальных групп. Данные перемещаются в виртуальные группы в фоновом режиме, приложения продолжают чтение и запись.
- Добавлена аутентификация через внешних поставщиков удостоверений OpenID Connect. YDB проверяет JWT-токены по JSON Web Key Set (JWKS) поставщика и периодически обновляет данные аутентификации.
- Kafka API поддерживает взаимную TLS-аутентификацию. Клиентский сертификат сопоставляется с идентификатором безопасности, SASL-аутентификация не требуется.
- Оптимизация движка колоночных таблиц: для колоночных таблиц используется обновлённая стратегия компактификации, которая эффективнее организует данные и новая стратегия слияния данных при чтении, которая ускоряет запросы на постоянно меняющихся данных
- Оптимизация подсистемы аутентификации/авторизации: по умолчанию включены пакетные запросы авторизации в AccessService, что снижает накладные расходы.
- Потоковым YQL-запросам стали доступны системные виртуальные атрибуты такие, как `__ydb_create_time`, `__ydb_write_time` и т.п., а также пользовательские атрибуты `__ydb_user_attributes`. Ссылка на функциональность.
- Оптимизация подсистемы Distributed Storage: полная синхронизация VDisk стала быстрее за счёт удаления обработанных данных SyncLog.
- В `DescribeTransfer` добавлены метрики и статистика для мониторинга и диагностики трансферов.
- Добавлен настраиваемый предел числа хранимых операций принудительной компактификации. Завершённые и отменённые операции могут удаляться автоматически после достижения предела.
- Записи Change Data Capture могут содержать идентификатор трейса OpenTelemetry запроса, создавшего изменение.
- При чтении топика с временной метки отфильтровываются сообщения с более ранним временем записи, в том числе из одного блоба с более новыми сообщениями.

Отключенная функциональность

Перечисленная ниже функциональность не включена по умолчанию.

- Для колоночных таблиц можно запускать принудительную компактификацию с помощью `ALTER TABLE ... COMPACT`.
- Между колоночными и строчными таблицами достигнут паритет по набору YQL типов (поддержаны Interval, Uuid, DyNumber).
- Добавлен гибридный поиск, объединяющий полнотекстовую релевантность и векторную близость в ранжированный результат.
- С топиками можно работать через Amazon SQS API, используя SQS-совместимые клиенты для чтения и записи сообщений.
- Добавлены JSON-индексы для ускорения запросов с `JSON_EXISTS` и `JSON_VALUE`.
- Полнотекстовые индексы поддерживают колонки фильтрации, позволяющие искать в логическом разделе таблицы.
- Полнотекстовые индексы можно создавать для таблиц с произвольными типами первичного ключа.

Версия 26.2

Версия 26.2.1.14

Дата выхода: 16 сентября 2026.

Функциональность

- Полнотекстовые индексы включены по умолчанию.
- Потоковые запросы поддерживают чтение из локальных топиков, запись в локальные топики, чтение локальных таблиц и несколько инструкций `INSERT` в одном запросе.
- В потоковых запросах доступны watermarks.
- Добавлены Bloom-индексы: Bloom и Bloom n-gram для колоночных таблиц и префиксные Bloom-индексы для строковых таблиц.
- Настройка сжатия колонок в колоночных таблицах доступна по умолчанию.
- Для построения индексов теперь можно настраивать уровень параллелизма.
- Для строковых таблиц в `ALTER TABLE` по умолчанию доступны инструкции `ALTER COLUMN SET DEFAULT` и `ALTER COLUMN DROP DEFAULT`.
- Для строковых таблиц по умолчанию доступна инструкция YQL `TRUNCATE TABLE`.
- Инструкция YQL `DISCARD SELECT` доступна по умолчанию.
- QueryService поддерживает выдачу результатов запросов в формате Apache Arrow; эта возможность включена по умолчанию.
- Для строковых таблиц добавлен запуск принудительной компакции с помощью `ALTER TABLE ... COMPACT`.
- Добавлена автоматическая балансировка хранилища между группами и фоновая проверка корректности размещения дисков.
- Ускорены операции разделения и слияния таблиц с большим числом партиций: SchemeShard обновляет только затронутые партиции вместо полного пересчёта списка.
- Добавлено аудитное логирование операций с топиками.
- Добавлен встроенный сбор минидампов на базе Google Breakpad для узлов Linux.
- Добавлена подкоманда `ydb-dstool pdisk populate` для воспроизведения нагрузки PDisk на другом устройстве.

Отключенная функциональность

Функциональность присутствует в ядре для возможности отката с будущего релиза 26-3, но не включена по умолчанию. Будет доступна по умолчанию в следующем major-релизе. Также может быть включена на некоторых управляемых сервисах YDB.

- Добавлена поддержка инкрементальных резервных копий, которые позволяют сохранять только изменения относительно предыдущей резервной копии в коллекции.
- Поддерживаются экспорт и импорт колоночных таблиц через S3-совместимые хранилища.
- Добавлены экспорт и импорт строковых таблиц через локальную файловую систему, в том числе через файловые системы, подключённые по NFS.
- Добавлено удержание snapshot для длительных аналитических запросов к колоночным таблицам, чтобы данные snapshot не удалялись до завершения запроса.
- QueryService может уведомлять SDK о завершении работы узла или сессии, чтобы клиент перестал направлять туда новые запросы.
- Для колоночных таблиц добавлены лимиты на количество и объём малых блобов на уровне базы данных. При превышении жёсткого лимита новые записи отклоняются.
- Выражения для watermarks могут вычисляться вне контекста отдельного сообщения.
- Добавлены min-max-индексы для колоночных таблиц.
- Добавлено словарное кодирование колонок в колоночных таблицах.
- Добавлено онлайн-построение уникальных вторичных индексов.
- Для транзакций между топиками и таблицами добавлена оптимизированная проверка конфликтов.

Исправления ошибок

- Исправлены некорректные результаты некоторых сканирующих запросов к колоночным таблицам.
- Исправлена обработка повреждённых Kafka-запросов, которая могла приводить к избыточному потреблению памяти или выходу за границы буфера.
- Исправлено зависание чтения из топика после перезапуска балансировщика чтения.
- Исправлены состояния гонки в серверной сессии чтения топика и в Topic SDK.
- Исправлены падение и зависание потоковых запросов при создании контрольных точек.
- Исправлены состояния гонки при отмене и планировании распределённых транзакций.
- Исправлены обработка слишком больших блоков при зашифрованном экспорте и ложная ошибка повреждения данных при зашифрованном восстановлении.
- Исправлено состояние гонки при сборе статистики командой `ydb workload topic`.
- Исправлено повторное освобождение памяти при завершении `DqHashCombine` с spilling.
- Исправлена потеря подтверждений `ReadSet`, которая могла препятствовать завершению транзакции с топиками.
- Исправлена обработка ответа `NODATA` в KeyValue API: вместо падения процесса возвращается ошибка `NOT_FOUND` или `INTERNAL_ERROR`.
- Исправлена IAM-аутентификация для внешних источников данных в Generic Provider и обработка возвращаемых им ошибок.
- Исправлена утечка памяти при загрузке метаданных внешних источников данных.
- Исправлена обработка трёхсоставных feature flags в YAML-конфигурации, которая могла приводить к потере следующих настроек.
- Исправлено копирование и экспорт таблиц со вторичными индексами после удаления внутренних таблиц индекса.
- Исправлены фильтрация экспортируемых объектов и операции со списками при экспорте в файловую систему.
- Исправлены ответы Metadata в Kafka API, которые могли содержать пустой список брокеров или несогласованный идентификатор контроллера и приводить к тайм-аутам Kafka AdminClient и Kafka Streams.
- Исправлена утечка записей о выполнении скриптов, создаваемых потоковыми запросами.
- Исправлена перезапись пользовательского `config.yaml`, смонтированного по пути по умолчанию, при первом развёртывании `local-ydb` в Docker.
- Исправлено падение Hive после перезапуска, когда блокировка таблетки и сохранённый лидер указывали на разные узлы.

Версия 26.1

Версия 26.1.1.22

Дата выхода: 27 июля 2026.

Исправления ошибок

- Исправлено аутентификация через Kafka API для локальных пользователей: при включенной настройке `DomainLoginOnly` пользователи не могли работать с тенантными базами.
- Исправлено падение (use-after-free) при обновлении векторного индекса из-за асинхронного уничтожения ReadActor.

Версия 26.1.1.20

Дата выхода: 02 июля 2026.

Функциональность

- Доступны YQL-запросы `SHOW CREATE TABLE` и `SHOW CREATE VIEW` для получения DDL-выражений, необходимых для воссоздания структуры таблицы или представления.
- В `ALTER TABLE` добавлена поддержка значений по умолчанию при `ADD COLUMN` (`DEFAULT`).
- Shuffle Elimination включён в продакшене: оптимизатор может устранять лишние перераспределения данных при соединениях.
- Реализовано резервное копирование и восстановление схемных объектов: асинхронных репликаций, внешних источников данных, внешних таблиц и трансферов.
- Кластер сохраняет работоспособность при недоступности CMS.
- Добавлена возможность регистрации динамических узлов по клиентским TLS-сертификатам.
- Для LDAP-аутентификации сервисного аккаунта поддерживается протокол SASL с механизмом EXTERNAL — см. `enable_sasl_external_bind`.
- В асинхронной репликации поддержано зеркалирование автопартиционированных топиков; см. также партиции топиков в CDC.

- Расширена диагностика TLI (Transaction Lock Invalidation): конфигурация `tli_config`, логирование и системные представления.
- При автопартиционировании по нагрузке учитывается CPU-нагрузка на лидера партиции и на все её реплики.
- Потоковые запросы поддерживают запись результатов в локальные таблицы.
- В потоках изменений (CDC) доступна выгрузка идентификаторов безопасности пользователя (`USER_SIDS`) — см. `ALTER TABLE CHANGEFEED`.
- Для внешних источников данных добавлен `AUTH_METHOD=IAM`.
- В CLI поддерживается аутентификация по файлу токена (`--token-file`).
- Оптимизирована работа транзакций между топиками и таблицами.

Исправления ошибок

- Исправлено `RETURNING` в потоковых `UPDATE` и интерактивных запросах.
- Исправлено `SET DEFAULT` и `DROP DEFAULT` в `ALTER TABLE`.
- Исправлено обработка срока действия токенов аутентификации в ticket parser.
- Исправлено выполнение запросов в Workload Manager после пересоздания tenant.
- Исправлено use-after-free в gRPC-сервисе.
- Исправлено инкрементальное восстановление при перезапусках SchemeShard и сбоях шардов.
- Исправлено отсутствие метаданных потокового запроса сразу после создания.
- Исправлено зависание async checkpointing при полном входе и пустом checkpoint.
- Исправлено взаимное влияние квот разных баз в Kesus quoter proxy.
- Исправлено зависания в PQ read session.
- Исправлено зависание scan executor на `SELECT … LIMIT` по пустым таблицам.
- Исправлено отложенный сброс TLI `LOCKS_BROKEN`.
- Исправлено переполнение таймаута потокового запроса.
- Исправлено потокобезопасность и разбор срока действия токена в IAM credentials provider.
- Исправлено ошибки в HTTP gateway.
- Исправлено переполнение при обработке пустого пароля.
- Исправлено сбор CDC для колонок `IsBuildInProgress` отключён.
- Исправлено segfault при обновлении.
- Исправлено Shuffle Elimination с прагмой `HashJoinMode`.
- Исправлено дублирование строк в scan query при проблемах доставки.
- Исправлено доступ к HTTP-эндпоинтам viewer: только viewer/admin SID; scope базы для database-only токенов.
- Исправлено OOM при загрузке trash на blob depot.
- Исправлено падение `TQueryBase` после отмены потокового запроса.
- Исправлено локальное чтение CDC из YQL.
- Исправлено быстрая удалённая отмена запросов в query service.

Версия 25.4

Версия 25.4.1.15

Дата выхода: 5 июня 2026.

Функциональность

- Доступны инструкции YQL `BATCH UPDATE` и `BATCH DELETE FROM` для массового обновления и удаления данных в таблицах.
- Механизм выполнения операций записи существенно изменился -- теперь запись выполняется в потоковом режиме без полной материализации данных на стороне Query Processor перед отправкой в даташарды, что повышает производительность при крупных операциях записи. Изменение распространяется на часть сценариев; в отдельных случаях (например, таблицы со вторичными индексами) по-прежнему используется прежний подход. Общие сведения о конвейере выполнения см. в разделе Выполнение запросов.
- Оптимизировано выполнение Lookup Join: используется потоковый режим без материализации одной из сторон соединения, что снижает пиковое потребление памяти, ускоряет выполнение на больших наборах данных и снимает прежние ограничения на размер сторон соединения. См. описание Index lookup Join и синтаксис оператора `JOIN`.
- Добавлена возможность выставлять права доступа для системных представлений кластера и баз данных.
- Для строковых таблиц доступна настройка режимов кэширования и новый режим `in_memory`, который позволяет предзагружать данные таблицы в оперативную память при условии наличия необходимых объемов оперативной памяти.
- Для читателей топиков добавлен параметр `availability-period`, позволяющий продлить хранение неподтверждённых сообщений сверх retention-period
- Доступны попартиционные метрики топиков и выгрузка в пользовательские шард-квоты для учёта и наблюдаемости.
- Ускорение запросов с `LIMIT` в колоночных таблицах за счёт раннего ограничения выборки на узлах хранения (для запросов без сортировки или с сортировкой по первичному ключу). Общий синтаксис `LIMIT` и `OFFSET` в YQL.
- Колоночные таблицы поддерживают тип `Bool` в схеме и запросах — см. примитивные типы YQL.
- Фильтруемый векторный индекс корректно находит вставленные в таблицу после создания индекса строки с новыми значениями колонок фильтров.
- Потоковая обработка и поставка данных теснее интегрированы в ядро: трансфер «топик → таблица», потоковые запросы стали доступны пользователям при включении 'EnableStreamingQueries'.
- Добавлена опция `overlap_clusters` для существенного улучшения качества векторного поиска за счёт помещения векторов в несколько кластеров индекса (настройки индекса) — см. векторные индексы.
- Значительно ускорен поиск по всем видам векторных индексов за счёт подсчёта расстояний локально на каждом даташарде до передачи по сети — см. VIEW (векторный индекс) и векторные индексы.
- Ускорен полный векторный поиск без ANN-индекса за счёт pushdown (векторный поиск, KNN UDF) — см. векторный поиск и модуль KNN.
- Полноценно поддержан механизм работы с секретами, хранимыми в базе данных: создание, изменение, удаление и использование — см. Секреты. Обратите внимание, что старый синтаксис объявлен устаревшим.
- Улучшено выполнение `UNION ALL`: теперь поддерживается параллельное выполнение, что повышает производительность аналитических запросов.

Исправления ошибок

- Исправлена уязвимость LDAP-аутентификации: зная логин и пароль любого LDAP-пользователя (в том числе не входящего в группу с доступом к YDB), можно было обойти проверку членства в группе и получить доступ к кластеру (инъекция в LDAP-фильтр поиска пользователя; добавлено экранирование спецсимволов по RFC 2254).

Версия 25.3

Версия 25.3.1.27

Дата выхода: 20 мая 2026.

Функциональность

- Добавлена поддержка конфигурации 2 ДЦ с синхронной записью данных (режим `Bridge`); доступно в YDB Enterprise.
- Улучшения топиков:
  - в Kafka API теперь можно создавать компактифицированные топики; YDB автоматически создаёт и удаляет внутреннего служебного потребителя, используемого для компактификации топика;
  - API топиков расширен: в выдачу `DescribeConsumer` добавлены новые параметры, а попартиционные метрики топиков можно отгружать в пользовательские квоты.
- Реализованы резервное копирование и восстановление конфигурации топика в S3 и из S3.
- Реализован экспорт представлений (`VIEW`) в S3 и из S3.

Исправления ошибок

- Исправлена уязвимость LDAP-аутентификации: зная логин и пароль любого LDAP-пользователя (в том числе не входящего в группу с доступом к YDB), можно было обойти проверку членства в группе и получить доступ к кластеру (инъекция в LDAP-фильтр поиска пользователя; добавлено экранирование спецсимволов по RFC 2254).
- Исправлена ошибка, приводившая к утечке сессии на серверной стороне.
- Исправлена ошибка, из-за которой в редких случаях чтения из таблицы могли заблокировать её удаление.
- Исправлено состояние гонки при обновлении soft-лимита CPU.
- Исправлено поведение, при котором `ALTER TABLE` мог завершаться с ошибкой для таблиц с векторным индексом.
- Исправлены неконсистентные результаты в некоторых read-write транзакциях — конфликтующие записи больше не затирают незакоммиченные изменения.
- Исправлено нарушение сериализуемости в read-write транзакциях после рестартов шарда.
- Исправлена ошибка управления памятью при коммите смещения в топиках со включённым автоматическим партиционированием.
- Добавлены проверки включённого шифрования в zero-copy передаче.
- Исправлена ошибка, приводящая к зависанию VDisk в local recovery после ошибки `ChunkRead`.
- Исключено появление фантомных VDisk из-за гонок между операциями создания и удаления группы.
- Улучшено определение состояния PDisk — теперь используется фактическое состояние из BSC, что повышает точность healthcheck.
- При завершении сессии через attach stream теперь отправляется уведомление.
- Сервис координации теперь корректно возвращает код `SCHEME_ERROR` для несуществующих ресурсов вместо ошибочно использовавшегося кода `INTERNAL_ERROR`.
- Исправлены ошибки работы с памятью и нарушения внутренней согласованности данных в Workload Manager и связанном с планировщиком коде.
- Исправлена проблема, из-за которой запросы информации о PDisk могли завершаться таймаутом, если целевой узел был отключён или недоступен.

Версия 25.2

Версия 25.2.1.26

Дата выхода: 12 мая 2026.

Исправления ошибок

- Исправлена уязвимость LDAP-аутентификации: зная логин и пароль любого LDAP-пользователя (в том числе не входящего в группу с доступом к YDB), можно было обойти проверку членства в группе и получить доступ к кластеру (инъекция в LDAP-фильтр поиска пользователя; добавлено экранирование спецсимволов по RFC 2254).
- Исправлена проблема, из-за которой удаление таблетки могло зависать
- Исправлена ошибка вызывающая ошибку, при изменении follower'а таблицы
- Исправлен ряд ошибок, связанных с changefeed:
  - Исправлена ошибка, из-за которой импорт таблицы с Utf8-ключом и включённым changefeed мог завершиться неудачно
  - Исправлена ошибка, когда импорт таблицы без потоков изменений мог завершаться сбоем из-за некорректного поиска файлов changefeed
- Исправлена ошибка, которая могла приводить к сбоям при UPSERT-операциях в колоночных таблицах
- Исправлена ошибка, вызывавшая сбой из-за обращения к уже освобождённой памяти
- Исправлена ошибка с дубликатами в уникальных вторичных индексах
- Исправлена ошибка неверного совпадения контрольных сумм при восстановлении сжатых бэкапов из S3
- Исправлена ошибка, из-за которой некоторые запросы бенчмарка TPC-H 1000 могли завершаться с ошибкой
- Исправлен ряд проблем, связанных с инициализацией кластера:
  - Исправлена ошибка, из-за которой инициализация кластера могла зависать при обязательной авторизации
  - Исправлена проблема из-за которой создание новых баз данных сразу после развёртывания кластера было невозможно в течение нескольких минут
- Исправлена ошибка, при которой мог возникать race condition и клиенты получали ошибку `Could not find correct token validator`, если использовались недавно выданные токены до обновления состояния `LoginProvider`
- Исправлена ошибка, при которой именованное выражение, содержащее другое именованное выражение, приводило к некорректному бэкапу `VIEW`

Релиз кандидат 25.2.1.10

Дата выхода: 21 сентября 2025.

Функциональность

- Аналитические возможности доступны по умолчанию: колоночные таблицы могут создаваться без включения специальных флагов, с использованием сжатия LZ4 и хеш-партиционирования. Поддерживаемые операции включают широкий набор DML (UPDATE, DELETE, UPSERT, INSERT INTO ... SELECT) и CREATE TABLE AS SELECT. Интеграция с dbt, Apache Airflow, Jupyter, Superset и федеративные запросы к S3 позволяют строить сквозные аналитические пайплайны в YDB.
- Стоимостной оптимизатор работает по умолчанию для запросов, использующих хотя бы одну колоночную таблицу, но может быть включён принудительно и для остальных запросов. Стоимостной оптимизатор улучшает производительность выполнения запросов, вычисляя оптимальный порядок и тип соединений на основе статистики таблиц; поддерживаемые hints позволяют тонко настраивать планы выполнения для сложных аналитических запросов.
- Реализован трансфер данных – асинхронный механизм переноса данных из топика в таблицу. Создание экземпляра трансфера, его изменение и удаление осуществляется с использованием YQL. Для быстрого старта воспользуйтесь инструкцией с примером.
- Добавлен спиллинг, механизм управления памятью, при котором промежуточные данные, возникающие в результате выполнения запросов и превышающие доступный объём оперативной памяти узла, временно выгружаются во внешнее хранилище. Спиллинг обеспечивает выполнение пользовательских запросов, которые требуют обработки больших объёмов данных, превышающих доступную память узла.
- Увеличено максимальное время на выполнение одного запроса с 30 минут до 2 часов.
- Добавлена поддержка Certificate Authority (CA) и Yandex Cloud Identity and Access Management (IAM) аутентификации в асинхронной репликации.
- Обязательно к настройке:
  - Аутентификация и авторизация узлов для регистрации узлов в кластере.
- Включены по умолчанию:
  - векторный индекс для приближённого векторного поиска;
  - поддержка в YDB Topics Kafka API клиентской балансировки читателей, компактифицированных топиков и транзакций;
  - поддержка автопартиционирования топиков в CDC для строковых таблиц;
  - поддержка автопартиционирования топиков для асинхронной репликации;
  - поддержка параметризованного типа Decimal;
  - поддержка типа DateTime64;
  - автоудаление временных директорий и таблиц при экспорте в S3;
  - поддержка потока изменений в операциях резервного копирования и восстановления;
  - возможность указания числа реплик для вторичного индекса;
  - системные представления с историей перегруженных партиций.

Исправления ошибок

- Исправлена ошибка в Workload Manager, из-за которой потребление CPU колоночными таблицами могло превышать установленные пределы.

Версия 25.1

Версия 25.1.4.18

Дата выхода: 12 мая 2026.

Функциональность

- Добавлена возможность использовать привычные инструменты потоковой обработки данных – Kafka Connect, Confluent Schema Registry, Kafka Streams, Apache Flink, AKH через Kafka API при работе с YDB Topics. Теперь YDB Topics Kafka API поддерживает:
  - клиентскую балансировку читателей – включается установкой флага `enable_kafka_native_balancing` в конфигурации кластера. Как работает балансировка читателей в Apache Kafka. Теперь балансировка читателей в Kafka API YDB Topics будет работать точно так же,
  - компактифицированные топики – включается установкой флага `enable_topic_compactification_by_key`,
  - транзакции – включается установкой флага `enable_kafka_transactions`.
- Добавлен новый протокол в Node Broker, устраняющий всплески сетевого трафика на больших кластерах (более 1000 серверов), связанного с рассылкой информации об узлах.

YDB UI

- Исправлена ошибка, из-за которой не все таблетки отображались на вкладке Tablets в разделе диагностики.
- Исправлена ошибка, из-за которой на вкладке Storage в разделе диагностики базы данных отображались не только узлы хранения.
- Исправлена ошибка сериализации, которая могла приводить к падению при открытии статистики выполнения запроса.
- Изменена логика перехода узлов в критическое состояние – заполненный на 75-99% CPU pool теперь вызывает предупреждение, а не критическое состояние.

Производительность

- Оптимизирована обработка пустых входов при выполнении JOIN-операций.

Исправления ошибок

- Исправлена уязвимость LDAP-аутентификации: зная логин и пароль любого LDAP-пользователя (в том числе не входящего в группу с доступом к YDB), можно было обойти проверку членства в группе и получить доступ к кластеру (инъекция в LDAP-фильтр поиска пользователя; добавлено экранирование спецсимволов по RFC 2254).
- Добавлена поддержка в асинхронной репликации нового типа записи об изменениях — `reset`-записи (в дополнение к `update`- и `erase`-записям).
- Исправлена ошибка, из-за которой экземпляр репликации с неуказанным параметром `COMMIT_INTERVAL` приводил к сбою процесса.
- Исправлены редкие ошибки при чтении из топика во время балансировки партиций.
- Исправлена ошибка, из-за которой при удалении dedicated-базы данных системные таблетки базы могли остаться неудалёнными.

- Исправлена ошибка, из-за которой таблетки могли зависать при недостатки памяти на узлах. Теперь таблетки будут автоматически запускаться, как только на каком-либо из узлов освободится достаточное количество ресурсов.
- Исправлена ошибка, из-за которой при записи сообщений Kafka сохранялось только первое сообщение из батча, а остальные сообщения игнорировались.

Релиз кандидат 25.1.2.7

Дата выхода: 14 июля 2025.

Функциональность

- Реализован векторный индекс для приближённого векторного поиска. Для векторного поиска опубликованы рецепты для YDB CLI и YQL, а также примеры работы на C++ и Python.
- Добавлена поддержка консистентной асинхронной репликации.
- Добавлен механизм конфигурации V2, упрощающий развёртывание новых кластеров YDB и дальнейшую работу с ними. Сравнение механизмов конфигурации V1 и V2.
- Добавлена поддержка параметризованного типа Decimal.
- Добавлена возможность не использовать оператор `DECLARE` для объявления типов параметров в запросах. Теперь типы параметров определяются автоматически на основе переданных значений.
- Реализована клиентская балансировка партиций при чтении по протоколу Kafka (как у самой Kafka). Раньше балансировка происходила на сервере. Включается установкой флага `enable_kafka_native_balancing` в конфигурации кластера.
- Добавлена поддержка автопартиционирования топиков в CDC для строковых таблиц. Включается установкой флага `enable_topic_autopartitioning_for_cdc` в конфигурации кластера.
- Добавлена возможность изменить время хранения данных в CDC-топике с использованием выражения `ALTER TOPIC`.
- Поддержан формат DEBEZIUM_JSON для потоков изменений (changefeed).
- Добавлена возможность создавать потоки изменений к индексным таблицам.
- Добавлена возможность указания числа реплик для вторичного индекса. Включается установкой флага `enable_access_to_index_impl_tables` в конфигурации кластера.
- В операциях резервного копирования и восстановления расширен состав поддерживаемых объектов. Включается установкой флагов, указанных в скобках:
  - поддержка потока изменений (флаги `enable_changefeeds_export` и `enable_changefeeds_import`);
  - поддержка представлений (`VIEW`) (флаг `enable_view_export`).
- Добавлено автоудаление временных директорий и таблиц при экспорте в S3. Включается установкой флага `enable_export_auto_dropping` в конфигурации кластера.
- Добавлена автоматическая проверка целостности резервных копий при импорте, предотвращающая восстановление из повреждённых резервных копий и защищающая от потери данных.
- Добавлена возможность создания представлений, использующих UDF в запросах.
- Добавлены системные представления с информацией о настройках прав доступа, истории перегруженных партиций - включается установкой флага `enable_followers_stats` в конфигурации кластера, истории партиций строковых таблиц со сломанными блокировками (TLI).
- Добавлены новые параметры в операторы CREATE USER и ALTER USER:
  - `HASH` — возможность задания пароля в зашифрованном виде;
  - `LOGIN` и `NOLOGIN` — разблокировка и блокировка пользователя.
- Повышена безопасность учётных записей:
  - Добавлена проверка сложности пароля пользователя;
  - Реализована автоматическая блокировка пользователя при исчерпании лимита попыток ввода пароля;
  - Добавлена возможность самостоятельной смены пароля пользователем.
- Реализована возможность переключения функциональных флагов во время работы сервера YDB. Флаги, для которых в proto-файле не указан параметр `(RequireRestart) = true`, будут применяться без рестарта кластера.
- Теперь самые старые (а не новые) блокировки меняются на полношардовые при превышении количества блокировок на шардах.
- Реализовано сохранение оптимистичных блокировок в памяти при плавном перезапуске даташардов, что должно уменьшить число ошибок ABORTED из-за потери блокировок при балансировке таблиц между узлами.
- Реализована отмена волатильных транзакций со статусом ABORTED при плавном перезапуске даташардов.
- Добавлена возможность удалить `NOT NULL`-ограничения на столбец в таблице с помощью запроса `ALTER TABLE ... ALTER COLUMN ... DROP NOT NULL`.
- Добавлено ограничение в 100 000 на число одновременных запросов на создание сессий в сервисе координации.
- Увеличено максимальное число столбцов в первичном ключе с 20 до 30.
- Улучшена диагностика и интроспекция ошибок, связанных с памятью (#10419, #11968).
- ***(Экспериментально)*** Добавлен экспериментальный режим работы с более строгими проверками прав доступа. Включается установкой следующих флагов:
  - `enable_strict_acl_check` – не позволять выдавать права несуществующим пользователям и удалять пользователей, если им выданы права;
  - `enable_strict_user_management` — включает строгие правила администрирования локальных пользователей (т.е. администрировать локальных пользователей может только администратор кластера или базы данных);
  - `enable_database_admin` — добавляет роль администратора базы данных.

Изменения с потерей обратной совместимости

- Если вы используете запросы, в которых происходит обращение к именованным выражениям как к таблицам с помощью AS_TABLE, обновите temporal over YDB на версию v1.23.0-ydb-compat перед обновление YDB на текущую версию, чтобы избежать ошибок в выполнении таких запросов.

YDB UI

- В редактор запросов добавлена поддержка частичной загрузки результатов — отображение начинается сразу при получении первого фрагмента с сервера без ожидания полного завершения запроса. Это позволяет быстрее получать результаты.
- Улучшена безопасность: элементы управления, которые недоступны пользователю, теперь не отображаются в интерфейсе. Пользователи не будут сталкиваться с ошибками "Доступ запрещен".
- Добавлен добавлен поиск по идентификатору таблетки на вкладку "Tablets".

- Добавлена подсказка по горячим клавишам, которая открывается по комбинации `⌘+K`.
- На страницу базы данных добавлена вкладка "Операции", которая позволяет просматривать список операций и отменять их.
- Обновлена панель мониторинга кластера, добавлена возможность ее свернуть.
- Реализована поддержка поиска с учетом регистра в инструменте иерархического отображения JSON.
- На верхнюю панель после выбора базы данных добавлены примеры кода для подключения в YDB SDK, что ускоряет процесс разработки.
- Исправлена сортировка строк на вкладке Запросы.
- Удалены лишние запросы подтверждения при закрытии страницы браузера в редакторе запросов — подтверждение запрашивается только когда это необходимо.

Производительность

- Добавлена поддержка свёртки констант в оптимизаторе запросов по умолчанию, что повышает производительность запросов за счёт вычисления константных выражений на этапе компиляции.
- Добавлен новый протокол гранулярного таймкаста, который позволит сократить время выполнения распределённых транзакций (замедление одного шарда не будет приводить к замедлению всех).
- Реализована функциональность сохранения состояния даташардов в памяти при перезапусках, что позволяет сохранить блокировки и повысить шансы на успешное выполнение транзакций. Это сокращает время выполнения длительных транзакций за счёт уменьшения числа повторных попыток.
- Реализована конвейерная обработка внутренних транзакций в Node Broker, что ускорило запуск динамических узлов в кластере YDB.
- Улучшена устойчивость Node Broker к повышенной нагрузке со стороны узлов кластера.
- Включены по умолчанию выгружаемые B-Tree-индексы вместо невыгружаемых SST-индексов, что позволяет сократить потребление памяти при хранении «холодных» данных.
- Оптимизировано потребление памяти узлами хранения.
- Сократили время запуска Hive до 30%.
- Оптимизирован процесс репликации в распределённом хранилище.
- Оптимизирован размер заголовка больших двоичных объектов в VDisk.
- Уменьшено потребление памяти за счёт очистки страниц аллокатора.

Исправления ошибок

- Исправлена ошибка в настройке Interconnect, приводящая к снижению производительности.
- Исправлена ошибка «Out of memory» при удалении очень больших таблиц за счёт регулирования числа одновременно обрабатывающих данную операцию таблеток.
- Исправлена ошибка, возникавшая при указании одного и того же узла базы данных несколько раз в конфигурации для системных таблеток.
- Устранена ошибка длительного (секунды) чтения данных при частых операциях перешардирования таблицы.
- Исправлена ошибка чтения из асинхронных реплик, приводившая к сбою.
- Исправлены редкие зависания при первоначальном сканировании CDC.
- Исправлена обработка незавершённых схемных транзакций в даташардах при перезапуске системы.
- Исправлена ошибка несогласованного чтения из топика при попытке явно подтвердить сообщение, прочитанное в рамках транзакции. Теперь пользователь при попытке подтвердить сообщение получит ошибку.
- Исправлена ошибка, из-за которой автопартиционирование некорректно работало при работе с топиком в транзакции.
- Исправлены зависания транзакций при работе с топиками во время перезапуска таблеток.
- Исправлена ошибка «Key is out of range» при импорте из S3-совместимого хранилища.
- Исправлено некорректное определение конца поля с метаданными в конфигурации кластера.
- Улучшено построение вторичных индексов: при возникновении некоторых ошибок система ретраит процесс, а не прерывает его.
- Исправлена ошибка выполнения выражения `RETURNING` в запросах `INSERT INTO` и `UPSERT INTO`.
- Исправлена проблема зависания операции «Drop Tablet» в PQ tablet, особенно во время задержек в работе Interconnect.
- Исправлена ошибка, возникавшая во время компакшна VDisk.
- Исправлена проблема, из-за которой длительные сессии чтения топика завершались с ошибками «too big inflight».
- Исправлено зависание при чтении топика, если хотя бы одна партиция не имела входящих данных, но читалась несколькими потребителями.
- Устранена редкая проблема перезагрузок PQ tablet.
- Устранена проблема, при которой после обновления версии кластера Hive запускались подписчики в датацентрах без работающих узлов баз данных.
- Исправлена ошибка `Failed to set up listener on port 9092 errno# 98 (Address already in use)`, возникавшая при обновлении версии.
- Исправлена ошибка, приводившая к segmentation fault при одновременном выполнении запроса к healthcheck и отключении узла кластера.
- Исправлен сбой в партиционировании строковой таблицы при выборе разделённого ключа из образцов доступа, содержащих смешанные операции с полным ключом и префиксом ключа (например, точное чтение или чтение диапазона).
- Исправлена ошибка, из-за которой автопартиционирование топиков не работало, когда параметр конфигурации `max_active_partition` задавался с помощью выражения `ALTER TOPIC`.
- Исправлена ошибка, из-за которой `ydb scheme describe` возвращал список столбцов не в том порядке, в котором они были заданы при создании таблицы.

Версия 24.4

Версия 24.4.4.12

Дата выхода: 3 июня 2025.

Производительность

- Ограничено количество одновременно обрабатываемых изменений конфигураций.
- Оптимизировано потребление памяти PQ-таблетками.

- Оптимизировано потребление CPU таблеткой Scheme shard, что уменьшило задержки ответов на запросы. Теперь лимит на число операций Scheme shard проверяется до выполнения операций разделения и слияния таблеток.

Исправления ошибок

- Исправлена редкая ошибка зависания клиентских приложений во время выполнения коммита транзакции, когда удаление партиции совершалось раньше обновления квоты на запись в топик.
- Исправлена ошибка в копировании таблиц с типом Decimal, которая приводила к сбою при откате на предыдущую версию.
- Исправлена ошибка, при которой коммит без подтверждения записи в топик приводил к блокировке текущей и следующих транзакций с топиками.
- Исправлены зависания транзакций при работе с топиками при перезагрузке или удалении таблетки.
- Исправлены проблемы с чтением сообщений больше 6Mb через Kafka API.
- Устранена утечка памяти во время записи в топик.
- Исправлены ошибки обработки nullable столбцов и столбцов с типом UUID в строковых таблицах.

Версия 24.4.4.2

Дата выхода: 15 апреля 2025.

Функциональность

- Включены по умолчанию:
  - поддержка представлений (VIEW);
  - режим автопартиционирования топиков;
  - транзакции с участием топиков и строковых таблиц;
  - волатильные распределённые транзакции.
- Добавлена возможность чтения и записи в топик с использованием Kafka API без аутентификации.

Производительность

- Включен по умолчанию автоматический выбор вторичного индекса при выполнении запроса.

Исправления ошибок

- Исправлена ошибка, которая приводила к существенному снижению скорости чтения с подписчиков таблетки.
- Исправлена ошибка, которая приводила к ожиданию подтверждения волатильной распределённой транзакции до следующего рестарта.
- Исправлена редкая ошибка, которая приводила к сбою при подключении подписчиков таблетки к лидеру с несогласованным состоянием журнала команд.
- Исправлена редкая ошибка, которая приводила к сбою при перезапуске удалённого datashard с неконсистентными изменениями.
- Исправлена ошибка, из-за которой мог нарушаться порядок обработки сообщений в топике.
- Исправлена редкая ошибка, из-за которой могло зависать чтение из топика.
- Исправлена проблема, из-за которой транзакция зависала при одновременном управлении топиком пользователем и перемещении PQ таблетки на другой узел.
- Исправлена проблема с утечкой значения счётчика для userInfo, которая могла приводить к ошибке чтения `too big in flight`.
- Исправлен сбой прокси-сервера из-за дублирования топиков в запросе.
- Исправлена редкая ошибка, из-за которой пользователь мог писать в топик в обход ограничений квоты аккаунта.
- Исправлена проблема, из-за которой после удаления топика система возвращала "OK", но его таблетки продолжали работать. Для удаления таких таблеток воспользуйтесь инструкцией из pull request.
- Исправлена редкая ошибка, из-за которой не восстанавливалась резервная копия большой таблицы с вторичным индексом.
- Исправлена проблема, которая приводила к ошибке при вставке данных с помощью `UPSERT` в строковые таблицы с значениями по умолчанию.
- Устранена ошибка, которая приводила к сбою при выполнении запросов к таблицам с вторичными индексами, возвращающих списки результатов с помощью выражения `RETURNING *`.

Версия 24.3

Версия 24.3.15.5

Дата выхода: 6 февраля 2025.

Функциональность

- Добавлена возможность регистрировать узел базы данных по сертификату. В Node Broker добавлен флаг `AuthorizeByCertificate` использования сертификата при регистрации.
- Добавлены приоритеты проверки аутентификационных тикетов с использованием стороннего IAM-провайдера, с самым высоким приоритетом обрабатываются запросы от новых пользователей. Тикеты в кеше обновляют свою информацию с приоритетом ниже.

Производительность

- Ускорено поднятие таблеток на больших кластерах: 210 мс → 125 мс (ssd), 260 мс → 165 мс (hdd).

Исправления ошибок

- Снято ограничение на запись в тип Uint8 значений больше 127.
- Исправлена ошибка, из-за которой при чтении из топика маленьких сообщений маленькими порциями значительно увеличивалась назгрузка на CPU. Это могло приводить к задержкам в чтении/записи в этот топик.
- Исправлена ошибка восстановления из резервной копии, сохраненной в хранилище S3 с Path-style адресацией.
- Исправлена ошибка восстановления из резервной копии, которая была создана в момент автоматического разделения таблицы.
- Исправлена ошибка в сериализации `Uuid` для CDC.
- Исправлена потенциальная поломка "замороженных" блокировок, к которой могли приводить массовые операции (например, удаление по TTL).

- Исправлена ошибка, из-за которой чтение на подписчиках таблетки могло приводить к сбоям во время автоматического разделения таблицы.
- Исправлена ошибка, при которой узел координации успешно регистрировал прокси-серверы несмотря на разрыв связи.
- Исправлена ошибка, возникающие при открытии в интерфейсе вкладки с информацией о группах распределенного хранилища.
- Исправлена ошибка, из-за которой Health Check не сообщал о проблемах в синхронизации времени.
- Исправлена редкая проблема, которая приводила к ошибкам при выполнении запроса на чтение.
- Исправлена редкая проблема, которая приводила к утечкам незакоммиченных изменений.
- Исправлены проблемы с согласованностью, связанные с кэшированием удаленных диапазонов.

Версия 24.3.11.14

Дата выхода: 9 января 2025.

- Поддержан рестарт без потери доступности кластера в минимальной отказоустойчивой конфигурации из трех узлов.
- Добавлены новый функции UDF Roaring bitmap: AndNotWithBinary, FromUint32List, RunOptimize

Версия 24.3.11.13

Дата выхода: 24 декабря 2024.

Функциональность

- Добавлена трассировка запросов – инструмент, позволяющий детально посмотреть путь следования запроса по распределенной системе.
- Добавлена поддержка асинхронной репликации, которая позволяет синхронизировать данные между базами YDB почти в реальном времени. Также она может быть использована для миграции данных между базами с минимальным простоем работающих с ними приложений.
- Добавлена поддержка представлений (VIEW), которая может быть включена администратором кластера с помощью настройки `enable_views` в динамической конфигурации.
- В федеративных запросах поддержаны новые внешние источники данных: MySQL, Microsoft SQL Server, Greenplum.
- Разработана документация по разворачиванию YDB с функциональностью федеративных запросов (в ручном режиме).
- Для Docker-контейнера с YDB добавлен параметр запуска `FQ_CONNECTOR_ENDPOINT`, позволяющий указать адрес коннектора ко внешним источникам данных. Добавлена возможность TLS-шифрования соединения с коннектором. Добавлена возможность вывода порта сервиса коннектора, локально работающего на том же хосте, что и динамический узел YDB.
- Добавлен режим автопартиционирования топиков, в котором топики могут разбивать партиции в зависимости от нагрузки с сохранением гарантий порядка чтения сообщений и exactly once записи. Режим может быть включен администратором кластера с помощью настроек `enable_topic_split_merge` и `enable_pqconfig_transactions_at_scheme_shard` в динамической конфигурации.
- Добавлены транзакции с участием топиков и строковых таблиц. Таким образом, можно транзакционно перекладывать данные из таблиц в топики и в обратном направлении, а также между топиками, чтобы данные не терялись и не дублировались. Транзакции могут быть включены администратором кластера с помощью настроек `enable_topic_service_tx` и `enable_pqconfig_transactions_at_scheme_shard` в динамической конфигурации.
- Добавлена поддержка CDC для синхронных вторичных индексов.
- Добавлена возможность изменить период хранения записей в CDC топиках.
- Добавлена поддержка автоинкремента для колонок, включенных в первичный ключ таблицы.
- Добавлена запись в аудитный лог событий логина пользователей в YDB, событий завершения сессии пользователя в пользовательском интерфейсе, а также запроса бэкапа и восстановления из бэкапа.
- Добавлено системное представление, позволяющее получить информацию о сессиях, установленных с базой данных, с помощью запроса.
- Добавлена поддержка константных значений по умолчанию для колонок строковых таблиц.
- Добавлена поддержка выражения `RETURNING` в запросах.
- Добавлена встроенная функция `version()`.
- Добавлены время запуска/завершения и автор в метаданные операций резервного копирования/восстановления из S3-совместимого хранилища.
- Добавлена поддержка резервного копирования/восстановления из S3-совместимого хранилища ACL для таблиц.
- Для запросов, читающих из S3, в план добавлены пути и метод декомпрессии.
- Добавлены новые настройки парсинга для `timestamp`, `datetime` при чтении данных из S3.
- Добавлена поддержка типа `Decimal` в ключах партиционирования.
- Улучшена диагностика проблем хранилища в HealthCheck.
- ***(Экспериментально)*** Добавлен стоимостной оптимизатор для сложных запросов, где участвуют колоночные таблицы. Оптимизатор рассматривает большое количество альтернативных планов выполнения и выбирает из них лучший на основе оценки стоимости каждого варианта. На текущий момент оптимизатор работает только с планами, где есть операции JOIN.
- ***(Экспериментально)*** Реализована начальная версия менеджера рабочей нагрузки, который позволяет создавать пулы ресурсов с ограничениями по процессору, памяти и количеству активных запросов. Реализованы классификаторы ресурсов для отнесения запросов к определенному пулу ресурсов.
- ***(Экспериментально)*** Реализован автоматический выбор индекса при выполнении запроса, который может быть включен администратором кластера с помощью настройки `index_auto_choose_mode` в `table_service_config` в динамической конфигурации.

YDB UI

- Поддержано создание и отображение экземпляра асинхронной репликации.
- Добавлено обозначение столбцов с автоинкрементом.
- Добавлена вкладка с информацией о таблетках.
- Добавлена вкладка с информацией о группах распределенного хранилища.
- Добавлена настройка для добавления трассировки ко всем запросам и отображение результатов трассировки запроса.
- На страницу PDisk добавлены атрибуты, информация о потреблении дискового пространства, а также кнопка, которая запускает декомиссию диска.
- Добавлена информация о выполняющихся запросах.

- Добавлена настройка лимита строк в выдаче для редактора запроса и отображение, если результаты запроса превысили лимит.
- Добавлено отображение перечня запросов с максимальным потреблением CPU за последний час.
- Добавлен поиск на страницах с историей запросов и списком сохраненных запросов.
- Добавлена возможность прервать исполнение запроса.
- Добавлена возможность сохранять запрос из редактора горячими клавишами.
- Разделено отображение дисков от дисков-доноров.
- Добавлена поддержка InterruptInheritance ACL и улучшено отображение действующих ACL.
- Добавлено отображение текущей версии пользовательского интерфейса.
- Добавлена с информацией о состоянии настроек включения экспериментальной функциональности.

Производительность

- Ускорено восстановление из бэкапа таблиц со вторичными индексами до 20% по нашим тестам.
- Оптимизирована пропускная способность Interconnect.
- Улучшена производительность CDC-топиков, содержащих тысячи партиций.
- Сделан ряд улучшений алгоритма балансировки таблеток Hive.

Исправления ошибок

- Исправлена ошибка, которая приводила в неработоспособное состояние базу с большим количеством таблиц или партиций при восстановлении из резервной копии. Теперь при превышении лимитов на размер базы, операция восстановления завершится ошибкой, база продолжит работать в штатном режиме.
- Реализован механизм, принудительно запускающий фоновый компакшн при обнаружении несоответствий между схемой данных и данными, хранящимися в DataShard. Это решает редко возникающую проблему задержки в изменении схемы данных.
- Устранено дублирование аутентификационных тикетов, которое приводило к повышенному числу запросов в провайдеры аутентификации.
- Исправлена ошибка нарушения инварианта при первоначальном сканировании CDC, приводившая к аварийному завершению серверного процесса ydbd.
- Запрещено изменение схемы таблиц резервного копирования.
- Исправлено зависание первоначального сканирования CDC при частых обновлениях таблицы.
- Исключены удаленные индексы из подсчета лимита на максимальное количество индексов.
- Исправлена ошибка в отображении времени, на которое запланировано выполнение набора транзакций (планируемый шаг).
- Исправлена проблема прерывания blue–green deployment в больших кластерах, возникающая из-за частого обновления списка узлов.
- Исправлена редко возникающая ошибка, которая приводила к нарушению порядка выполнения транзакций.
- Исправлена ошибка в EvWrite API, которая приводила к некорректному освобождению памяти.
- Исправлена проблема зависания волатильных транзакций после перезапуска.
- Исправлена ошибка в CDC, приводящая в некоторых случаях к повышенному потреблению CPU, вплоть до ядра на одну CDC-партицию.
- Устранена задержка чтения, возникающая во время и после разделения некоторых партиций.
- Исправлены ошибки при чтении данных из S3.
- Исправлен способ расчета aws signature при обращении к S3.
- Исправлены ложные срабатывания системы HealthCheck в момент бэкапа базы с большим количеством шардов.

Версия 24.2

Дата выхода: 20 августа 2024.

Функциональность

- Добавлена возможность задать приоритеты задачам обслуживания в системе управления кластером.
- Добавлена настройка стабильных имён для узлов кластера в рамках тенанта.
- Добавлено получение вложенных групп от LDAP-сервера, в LDAP-конфигурации улучшен парсинг хостов и добавлена настройка для отключения встроенной аутентификацию по логину и паролю.
- Добавлена возможность аутентификации динамических узлов по SSL-сертификату.
- Реализовано удаление неактивных узлов из Hive без его перезапуска.
- Улучшено управление inflight pings при перезапуске Hive в кластерах большого размера.
- Изменен порядок установления соединения с узлами при перезапуске Hive.

YDB UI

- Добавлена возможность задать TTL для сессии пользователя в конфигурационном файле.
- Добавлена сортировка по `CPUTime` в таблицу со списком запросов.
- Исправлена потеря точности при работе с `double`, `float`.
- Поддержано создание директорий из UI.
- Добавлена возможность задать интервал фонового обновления данных на всех страницах.
- Улучшено отображения ACL.
- Включено автодополнение в редакторе запросов по умолчанию.
- Добавлена поддержка View.

Исправления ошибок

- Добавлена проверка на размер локальной транзакции до ее коммита, чтобы исправить ошибки в работе схемных операции при выполнении экспорта/бекапа больших баз.
- Исправлена ошибка дублирования результатов SELECT-запроса при уменьшении квоты в DataShard.
- Исправлены ошибки, возникающие при изменении состояния координатора.
- Исправлены ошибки, возникающие в момент первичного сканирования CDC.
- Исправлено состояние гонки в асинхронной доставке изменений (асинхронные индексы, CDC).
- Исправлена редкая ошибка, из-за которой удаление по TTL приводило к аварийному завершению процесса.

- Исправлена ошибка отображения статуса PDisk в интерфейсе CMS.
- Исправлены ошибки, из-за которых мягкий перенос (drain) таблеток с узла мог зависать.
- Исправлена ошибка остановки interconnect proxy на узле, работающем без перезапусков, при добавлении другого узла в кластер.
- Исправлен учет свободной памяти в interconnect.
- Исправлены счетчики UnreplicatedPhantoms/UnreplicatedNonPhantoms в VDisk.
- Исправлена обработка пустых запросов сборки мусора на VDisk.
- Исправлено управление настройками TVDiskControls через CMS.
- Исправлена ошибка загрузки данных, созданных более новыми версиями VDisk.
- Исправлена ошибка выполнении запроса `REPLACE INTO` со значением по умолчанию.
- Исправлена ошибка исполнения запросов, в которых выполнялось несколько left join'ов к одной строковой таблице.
- Исправлена потеря точности для `float`, `double` типов при использовании CDC.

Версия 24.1

Дата выхода: 31 июля 2024.

Функциональность

- Реализована Knn UDF для точного поиска ближайших векторов.
- Разработан gRPC сервис QueryService, обеспечивающий возможность выполнения всех типов запросов (DML, DDL) и выборку неограниченных объёмов данных.
- Реализована интеграция с LDAP протоколом и возможность получения перечня групп из внешних LDAP-каталогов.

Встроенный UI

- Добавлен дашборд диагностики потребления ресурсов, который находится на вкладке с информацией о базе данных и позволяет определить текущее состояние потребление основных ресурсов: ядер процессора, оперативной памяти и места в сетевом распределенном хранилище.
- Добавлены графики для мониторинга основных показателей работы кластера YDB.

Производительность

- Оптимизированы таймауты сессий сервиса координации от сервера до клиента. Ранее таймаут составлял 5 секунд, что в худшем случае приводило к определению неработающего клиента (и освобождению удерживаемых им ресурсов) в течение 10 секунд. В новой версии время проверки зависит от времени ожидания сеанса, что обеспечивает более быстрое реагирование при смене лидера или захвате распределённых блокировок.
- Оптимизировано потребление CPU репликами SchemeShard, особенно при обработке быстрых обновлений для таблиц с большим количеством партиций.

Исправления ошибок

- Исправлена ошибка возможного переполнения очереди, Change Data Capture резервирует емкость очереди изменений при первоначальном сканировании.
- Исправлена потенциальная взаимоблокировка между получением записей CDC и их отправкой.
- Исправлена проблема потери очереди задач медиатора при переподключении медиатора, исправление позволяет обработать очередь задач медиатора при ресинхронизации.
- Исправлена редко возникающая ошибка, когда при включённых и используемых волатильных транзакциях возвращался успешный результат подтверждения транзакции до того, как она была успешно закоммичена. Волатильные транзакции по умолчанию выключены, находятся в разработке.
- Исправлена редко возникающая ошибка, приводившая к потере установленных блокировок и успешному подтверждению транзакций, которые должны были завершиться ошибкой Transaction Locks Invalidated.
- Исправлена редкая ошибка, приводящая к возможному нарушению гарантий целостности данных при конкурентной записи и чтении данных по определённому ключу.
- Исправлена проблема, из-за которой реплики для чтения переставали обрабатывать запросы.
- Исправлена редкая ошибка, которая могла привести к аварийному завершению процессов базы данных при наличии неподтверждённых транзакций над таблицей в момент её переименования.
- Исправлена ошибка в логике определения статуса статической группы, когда статическая группа не помечалась нерабочей, хотя должна была.
- Исправлена ошибка частичного коммита распределённой транзакции с незакоммиченными изменениями в случае некоторых гонок с рестартами.
- Исправлены аномалии с чтением устаревших данных, которые были обнаружены с помощью Jepsen.

Версия 23.4

Дата выхода: 14 мая 2024.

Производительность

- Исправлена проблема повышенного потребления вычислительных ресурсов актором топиков `PERSQUEUE_PARTITION_ACTOR`.
- Оптимизировано использование ресурсов репликами SchemeBoard. Наибольший эффект заметен при модификации метаданных таблиц с большим количеством партиций.

Исправления ошибок

- Исправлена ошибка возможной неполной фиксации накопленных изменений при использовании распределенных транзакций. Данная ошибка возникает при крайне редкой комбинации событий, включающей в себя перезапуск таблеток, обслуживающих вовлеченные в транзакцию партиции таблиц.
- Устранена гонка между процессами слияния таблиц и сборки мусора, из-за которой сборка мусора могла завершиться ошибкой нарушения инвариантов и, как следствие, аварийным завершением серверного процесса `ydbd`.
- Исправлена ошибка в Blob Storage, из-за которой информация о смене состава группы хранения могла не поступать своевременно на отдельные узлы кластера. В результате в редких случаях могли блокироваться операции чтения и записи данных, хранящихся в затронутой группе, и требовалось ручное вмешательство администратора.
- Исправлена ошибка в Blob Storage, из-за которой при корректной конфигурации могли не запускаться узлы хранения данных. Ошибка проявлялась для систем с явным образом включенной экспериментальной функцией "blob depot" (по

умолчанию эта функция выключена).

- Исправлена ошибка, возникавшая в некоторых ситуациях записи в топик с пустым `producer_id` при выключенной дедупликации. Она могла приводить к аварийному завершению серверного процесса `ydbd`.
- Исправлена проблема, приводящая к падению процесса `ydbd` из-за ошибочного состояния сессии записи в топик.
- Исправлена ошибка отображения метрики количества партиций в топике, ранее в ней отображалось некорректное значение.
- Устранены утечки памяти, которые проявлялись при копировании данных топиков между кластерами YDB. Они могли приводить к завершению серверных процессов `ydbd` из-за исчерпания доступной оперативной памяти.

Версия 23.3

Дата выхода: 12 октября 2023.

Функциональность

- Реализована видимость собственных изменений внутри транзакций. Ранее при попытке прочитать данные, уже модифицированные в текущей транзакцией, запрос завершался ошибкой. Это приводило к необходимости упорядочивать чтения и записи внутри транзакции. С появлением видимости собственных изменений эти ограничения снимаются, и запросы могут читать измененные в данной транзакции строчки.
- Добавлена поддержка колоночных таблиц. Колоночные таблицы хорошо подходят для работы с аналитическими запросами (Online Analytical Processing), так как при выполнении запроса считываются только те столбцы, которые непосредственно участвуют в запросе. Колоночные таблицы YDB позволяют создавать аналитические отчёты с производительностью, сопоставимой со специализированными аналитическими СУБД.
- Добавлена поддержка Kafka API для топиков. Теперь с YDB-топиками можно работать через Kafka-совместимый API, предназначенный для миграции существующих приложений. Обеспечена поддержка протокола Kafka версии 3.4.0.
- Добавлена возможность записи в топик без дедупликации. Такой вид записи хорошо подходит для случаев, когда порядок обработки сообщений не критичен. Запись без дедупликации работает быстрее и потребляет меньше ресурсов на сервере, но упорядочение и дедупликация сообщений на сервере не происходит.
- В YQL добавлены возможности создавать, изменять и удалять топики.
- Добавлена возможность назначать и отзывать права доступа с помощью команд YQL GRANT и REVOKE.
- Добавлена возможность логгировать DML-операции в аудитном логе.
- ***(Экспериментально)*** При записи сообщений в топик теперь можно передавать метаданные. Для включения этой функциональности добавьте `enable_topic_message_meta: true` в конфигурационный файл.
- ***(Экспериментально)*** Добавлена возможность чтения из топиков и запись в таблицу в рамках одной транзакции. Новая возможность упрощает сценарий переноса данных из топика в таблицу. Для её включения добавьте `enable_topic_service_tx: true` в конфигурационный файл.
- ***(Экспериментально)*** Добавлена поддержка совместимости с PostgreSQL. Новый механизм позволяет выполнять SQL запросы в PostgreSQL диалекте на инфраструктуре YDB с использованием сетевого протокола PostgreSQL. Можно использовать привычные инструменты работы с PostgreSQL, такие, как psql и драйверы (pq для Golang и psycopg2 для Python), а также разрабатывать запросы на привычном PostgreSQL синтаксисе с горизонтальной масштабируемостью и отказоустойчивостью YDB.
- ***(Экспериментально)*** Добавлена поддержка федеративных запросов. Она позволяет получать информацию из различных источников данных без их переноса в YDB. Поддерживается взаимодействие с ClickHouse, PostgreSQL, S3 через YQL-запросы без дублирования данных между системами.

Встроенный UI

- В настройках селектора типа запроса добавлена новая опция `PostgreSQL`, которая доступна при включении параметра `Enable additional query modes`. Также в истории запросов теперь учитывается синтаксис, используемый при выполнении запроса.
- Обновлен шаблон YQL-запроса для создания таблицы. Добавлено описание доступных параметров.
- Сортировка и фильтрация для таблиц Storage и Nodes вынесена на сервер. Необходимо включить параметр `Offload tables filters and sorting to backend` в разделе экспериментов, чтобы использовать данный функционал.
- В контекстное меню были добавлены кнопки для создания, изменения и удаления топиков.
- Добавлена сортировка по критичности для всех issues в дереве в `Healthcheck`.

Производительность

- Реализованы итераторные чтения. Новая функциональность позволяет разделить чтения и вычисления между собой. Итераторные чтения позволяют даташардам увеличить пропускную способность читающих запросов.
- Оптимизирована производительность записи в топики YDB.
- Улучшена балансировка таблеток при перегрузке нод.

Исправления ошибок

- Исправлена ошибка возможной блокировки читающими итераторами снепшотов, о которых не знают координаторы.
- Исправлена утечка памяти при закрытии соединения в kafka proxy.
- Исправлена ошибка, при которой снепшоты, взятые через читающие итераторы, могут не восстанавливаться на рестартах.
- Исправлен некорректный residual предикат для условия `IS NULL` на колонку.
- Исправлена срабатывающая проверка `VERIFY failed: SendResult(): requirement ChunksLimiter.Take(sendBytes) failed`.
- Исправлен `ALTER TABLE` по `TTL` для колоночных таблиц.
- Реализован `FeatureFlag`, который позволяет отключать/включать работу с `CS` и `DS`.
- Исправлено различие координаторного времени между 23-2 и 23-3 на 50мс.
- Исправлена ошибка, при которой ручка `storage` возвращала лишние группы, когда в запросе параметр `node_id` во `viewer backend`.
- Добавлен `usage` фильтр в `/storage` во `viewer backend`.
- Исправлена ошибка в Storage v2, при которой возвращалось некорректное число в `Degraded`.
- Исправлена отмена подписки от сессий в итераторных чтениях при рестарте таблетки.
- Исправлена ошибка, при которой во время роллинг-рестарта при походе через балансер моргает `healthcheck` алертами про storage.
- Обновлены метрики `cpu usage` в ydb.

- Исправлено игнорирование `NULL` при указании `NOT NULL` в схеме таблицы.
- Реализован вывод записей об операциях `DDL` в общий лог.
- Реализован запрет для команды `ydb table attribute add/drop` работать с любыми объектами, кроме таблиц.
- Отключён `CloseOnIdle` для `interconnect`.
- Исправлено задваивание скорости чтения в UI.
- Исправлена ошибка, при которой могли теряться данные на `block-4-2`.
- Добавлена проверка имени топика.
- Исправлен возможный `deadlock` в акторной системе.
- Исправлен тест `KqpScanArrowInChanels::AllTypesColumns`.
- Исправлен тест `KqpScan::SqlInParameter`.
- Исправлены проблемы параллелизма для OLAP-запросов.
- Исправлена вставка `ClickBench parquet`.
- Добавлен недостающий вызов `CheckChangesQueueOverflow` в общем `CheckDataTxReject`.
- Исправлена ошибка возврата пустого статуса при вызовах `ReadRows API`.
- Исправлен некорректны ретрай экспорта в финальной стадии.
- Исправлена проблема с бесконечной квотой на число записей в CDC-топике.
- Исправлена ошибка импорта колонки `string` и `parquet` в колонку `string` OLAP.
- Исправлено падение `KqpOlapTypes.Timestamp` под tsan.
- Исправлено падение во `viewer backend` при попытке выполнить запрос к базе из-за несовместимости версий.
- Исправлена ошибка, при которой `viewer` не возвращал ответ от `healthcheck` из-за таймаута.
- Исправлена ошибка, при которой в Pdisk'ах могло сохраняться некорректное значение `ExpectedSerial`.
- Исправлена ошибка, при которой ноды базы падают по `segfault` в S3 акторе.
- Исправлена гонка в `ThreadSanitizer: data race KqpService::ToDictCache-UseCache`.
- Исправлена гонка в `GetNextReadId`.
- Исправлено завышение результата `SELECT COUNT(*)` сразу после импорта.
- Исправлена ошибка, при которой `TEvScan` мог вернуть пустой набор данных в случае сплита даташарда.
- Добавлен отдельный issue/код ошибки в случае исчерпания доступного места.
- Исправлена ошибка `GRPC_LIBRARY Assertion failed`.
- Исправлена ошибка, при которой при чтении по вторичному индексу в сканирующих запросах получался пустой результат.
- Исправлена валидация `CommitOffset` в `TopicAPI`.
- Уменьшено потребление `shared cache` при приближении к OOM.
- Смержена логика планировщиков из `data executer` и `scan executer` в один класс.
- Добавлены ручки `discovery` и `proxy` в процесс выполнения `query` во `viewer backend`.
- Исправлена ошибка, при которой ручка `/cluster` возвращает название корневого домена типа `/ru` во `viewer backend`.
- Реализована схема бесшовного обновления табличек для `QueryService`.
- Исправлена ошибка, при которой `DELETE` возвращал данные и НЕ удалял их.
- Исправлена ошибка работы `DELETE ON` в `query service`.
- Исправлено неожиданное выключение батчинга в дефолтных настройках схемы.
- Исправлена срабатывающая проверка `VERIFY failed: MoveUserTable(): requirement move.ReMapIndexesSize() == newTableInfo->Indexes.size()`.
- Увеличен дефолтный таймаут grpc-сриминга.
- Исключены неиспользуемые сообщения и методы из `QueryService`.
- Добавлена сортировка по `Rack` в `/nodes` во `viewer backend`.
- Исправлена ошибка, при которой запрос с сортировкой возвращает ошибку при убывании.
- Исправлено взаимодействие `QP` с `NodeWhiteboard`.
- Удалена поддержка старых форматов параметров.
- Исправлена ошибка, при которой `DefineBox` не применялся для дисков, на которых есть статическая группа.
- Исправлена ошибка `SIGSEGV` в диннодах при импорте `CSV` через `YDB CLI`.
- Исправлена ошибка с падением при обработке `NGRpcService::TRefreshTokenImpl`.
- Реализован `gossip` протокол обмена информацией о ресурсах кластера.
- Исправлена ошибка `DeserializeValuePickleV1(): requirement data.GetTransportVersion() == (ui32) NDqProto::DATA_TRANSPORT_UV_PICKLE_1_0 failed`.
- Реализованы автоинкрементные колонки.
- Использовать статус `UNAVAILABLE` вместо `GENERIC_ERROR` при ошибке идентификации шарда.
- Добавлена поддержка `rope payload` в `TEvVGet`.
- Добавлено игнорирование устаревших событий.
- Исправлено падение write-сессий на невалидном имени топика.
- Исправлена ошибка `CheckExpected(): requirement newConstr failed, message: Rewrite error, missing Distinct((id)) constraint in node FlatMap`.
- Включён `safe heal` по умолчанию.

Версия 23.2

Дата выхода: 14 августа 2023.

Функциональность

- ***(Экспериментально)*** Реализована видимость собственных изменений. При включении этой функции вы можете читать измененные значения из текущей транзакции, которая еще не была закоммичена. Также эта функциональность позволяет выполнять несколько модифицирующих операций в одной транзакции над таблицей с вторичными индексами.

Для включения этой функциональности добавьте `enable_kqp_immediate_effects: true` в секцию `table_service_config` в конфигурационный файл.

- ***(Экспериментально)*** Реализованы итераторные чтения. Эта функциональность позволяет разделить чтения и вычисления между собой. Итераторные чтения позволяют даташардам увеличить пропускную способность читающих запросов. Для включения этой функциональности добавьте `enable_kqp_data_query_source_read: true` в секцию `table_service_config` в конфигурационный файл.

Встроенный UI

- Улучшена навигация:
  - Кнопки переключения между режимами диагностики и разработки вынесены на левую панель.
  - На всех страницах добавлены хлебные крошки.
  - На странице базы данных информация о группах хранения и узлах базы перенесена во вкладки.
- История и сохраненные запросы перенесены во вкладки над редактором запросов.
- На вкладках Info для объектов схемы настройки выведены в терминах конструкции `CREATE` или `ALTER`.
- Поддержано отображение колоночных таблиц в дереве схемы.

Производительность

- Для сканирующих запросов реализована возможность эффективного поиска отдельных строк с использованием первичного ключа или вторичных индексов, что позволяет во многих случаях значительно улучшить производительность. Как и в обычных запросах, для использования вторичного индекса необходимо явно указать его имя в тексте запроса с использованием ключевого слова `VIEW`.
- ***(Экспериментально)*** Добавлена возможность управлять системными таблетками базы (SchemeShard, Coordinators, Mediators, SysViewProcessor) её собственному Hive'y, вместо корневого Hive'a, и делать это сразу в момент создания новой базы. Без этого флага системные таблетки новой базы создаются в корневом Hive'е, что может негативно сказаться на его загруженности. Включение этого флага делает базы полностью изолированными по нагрузке, что может быть особенно актуально для инсталляций, состоящих из ста и более узлов. Для включения этой функциональности добавьте `enable_alter_database_create_hive_first: true` в секцию `feature_flags` в конфигурационный файл.

Исправления ошибок

- Исправлена ошибка в автоконфигурации акторной системы, в результате чего вся нагрузка ложится на системный пул.
- Исправлена ошибка, приводящая к полному сканированию при поиске по префиксу первичного ключа через `LIKE`.
- Исправлены ошибки при взаимодействии с репликами даташардов.
- Исправлены ошибки при работе с памятью в колоночных таблицах.
- Исправлена ошибки при обработке условий для immediate-транзакций.
- Исправлена ошибка в работе итераторных чтений на репликах даташардов.
- Исправлена ошибка, приводящая к лавинообразной переустановке сессий доставки данных до асинхронных индексов
- Исправлены ошибки в оптимизаторе в сканирующих запросах
- Исправлена ошибка некорректного расчёта потребления хранилища hive'ом после расширения базы
- Исправлена ошибка зависания операций от несуществующих итераторов
- Исправлены ошибки при чтении диапазона на `NOT NULL` колонке
- Исправлена ошибка зависания репликации VDisk'ов
- Исправлена ошибка в работе опции `run_interval` в TTL

Версия 23.1

Дата выхода 5 мая 2023. Для обновления до версии 23.1 перейдите в раздел Загрузки.

Функциональность

- Добавлено первоначальное сканирование таблицы при создании потока изменений CDC. Теперь можно выгрузить все данные, которые существуют на момент создания потока.
- Добавлена возможность атомарной замены индекса. Теперь можно атомарно и прозрачно для приложения подменить один индекс другим заранее созданным индексом. Замена выполняется без простоя.
- Добавлен аудитный лог — поток событий, который содержит информацию обо всех операциях над объектами YDB.

Производительность

- Улучшены форматы передачи данных между стадиями исполнения запроса, что ускорило SELECT на запросах с параметрами на 10%, на операциях записи — до 30%.
- Добавлено автоматическое конфигурирование пулов акторной системы в зависимости от их нагруженности. Это повышает производительность за счет более эффективного совместного использования ресурсов ЦПУ.
- Оптимизирована логика применения предикатов — выполнение ограничений с использованием OR и IN с параметрами автоматически переносится на сторону DataShard.
- (Экспериментально) Для сканирующих запросов реализована возможность эффективного поиска отдельных строк с использованием первичного ключа или вторичных индексов, что позволяет во многих случаях значительно улучшить производительность. Как и в обычных запросах, для использования вторичного индекса необходимо явно указать его имя в тексте запроса с использованием ключевого слова `VIEW`.
- Реализовано кеширование графа вычисления при выполнении запросов, что уменьшает потребление ЦПУ при его построении.

Исправления ошибок

- Исправлен ряд ошибок в реализации распределенного хранилища данных. Мы настоятельно рекомендуем всем пользователям обновиться на актуальную версию.
- Исправлена ошибка построения индекса на not null колонках.
- Исправлен подсчет статистики при включенном MVCC.
- Исправлены ошибки с бэкапами.
- Исправлена гонка во время сплита и удаления таблицы с CDC.

Версия 22.5

Дата выхода 7 марта 2023. Для обновления до версии **22.5** перейдите в раздел Загрузки.

Что нового

- Добавлены параметры конфигурации потока изменения для передачи дополнительной информации об изменениях в топик.
- Добавлена поддержка переименования для таблиц с включенным TTL.
- Добавлено управление временем хранения записей для потока изменений.

Исправления ошибок и улучшения

- Исправлена ошибка при вставке 0 строк операцией BulkUpsert.
- Исправлена ошибка при импорте колонок типа Date/DateTime из CSV.
- Исправлена ошибка импорта данных из CSV с разрывом строки.
- Исправлена ошибка импорта данных из CSV с пустыми значениями.
- Улучшена производительность Query Processing (WorkerActor заменен на SessionActor).
- Компактификация DataShard теперь запускается сразу после операций split или merge.

Версия 22.4

Дата выхода 12 октября 2022. Для обновления до версии **22.4** перейдите в раздел Загрузки.

Что нового

- YDB Topics и Change Data Capture (CDC):
  - Представлен новый Topic API. Топик YDB — это сущность для хранения неструктурированных сообщений и доставки их различным подписчикам.
  - Поддержка нового Topic API добавлена в YDB CLI и SDK. Topic API предоставляет методы потоковой записи и чтения сообщений, а также управления топиками.
  - Добавлена возможность захвата изменений данных таблицы с отправкой сообщений об изменениях в топик.
- SDK:
  - Добавлена возможность взаимодействовать с топиками в YDB SDK.
  - Добавлена официальная поддержка драйвера database/sql для работы с YDB в Golang.
- Embedded UI:
  - Поток изменений CDC и вторичные индексы теперь отображаются в иерархии схемы базы данных как отдельные объекты.
  - Улучшена визуализация графического представления query explain планов.
  - Проблемные группы хранения теперь более заметны.
  - Различные улучшения на основе UX-исследований.
- Query Processing:
  - Добавлен Query Processor 2.0 — новая подсистема выполнения OLTP-запросов со значительными улучшениями относительно предыдущей версии.
  - Улучшение производительности записи составило до 60%, чтения до 10%.
  - Добавлена возможность включения ограничения NOT NULL для первичных ключей в YDB во время создания таблиц.
  - Включена поддержка переименования вторичного индекса в режиме онлайн без остановки сервиса.
  - Улучшено представление query explain, которое теперь включает графы для физических операторов.
- Core:
  - Для read-only транзакций добавлена поддержка консистентного снапшота, который не конфликтует с пишущими транзакциями.
  - Добавлена поддержка BulkUpsert для таблиц с асинхронными вторичными индексами.
  - Добавлена поддержка TTL для таблиц с асинхронными вторичными индексами.
  - Добавлена поддержка сжатия при экспорте данных в S3.
  - Добавлен audit log для DDL statements.
  - Поддержана аутентификация со статическими учетными данными.
  - Добавлены системные представления для диагностики производительности запросов.

Список изменений сервера Корпоративной СУБД Яндекса

Версия 26.1

Версия 26.1.1.ent.3

Дата выхода: 3 августа 2026.

Версия включает в себя все доработки, содержащиеся в сборке YDB 26.1.1.22, см. описание изменений. Кроме того, версия включает в себя перечисленные ниже доработки, специфичные для Корпоративной СУБД.

Доработки, специфичные для Корпоративной СУБД

Перечисленные ниже изменения доступны в Корпоративной СУБД Яндекса дополнительно к соответствующей сборке YDB:

- Добавлена оптимизация, позволяющая фильтровать строки по колонкам из индекса до запроса к основной таблице, что позволяет сократить количество обращений к основной таблице при выполнении некоторых типов запросов.
- Реализован набор исправлений в индексном доступе (StreamIndexLookup), устраняющий возможность проявления редких ситуаций «зависания» выполняемых запросов и сокращающий потребление оперативной памяти при выполнении запросов.
- Некорректные представления теперь могут быть восстановлены из резервной копии. Это позволяет восстанавливать резервные копии, созданные из БД, содержащей такие представления, без дополнительных действий со стороны администратора.
- Добавлена поддержка взаимной аутентификации по сертификатам (mTLS) в Kafka API.
- В системные представления `.sys/top_queries_*` и `.sys/query_sessions` добавлена колонка `TraceId` с идентификатором трассировки запроса.

Версия 25.4

Версия 25.4.1.ent.2

Дата выхода: 17 июня 2026.

Версия включает в себя все доработки, содержащиеся в сборке YDB 25.4.1.15, см. описание изменений. Кроме того, включены все дополнительные исправления, перечисленные ниже для версии 25.2.1.ent.13.

Версия 25.3

Версия 25.3.1.ent.3

Дата выхода: 11 июня 2026.

Версия включает в себя все доработки, содержащиеся в сборке YDB 25.3.1.27, см. описание изменений. Кроме того, включены все дополнительные исправления, перечисленные ниже для версии 25.2.1.ent.13.

Версия 25.2

Версия 25.2.1.ent.13

Дата выхода: 11 июня 2026.

Версия включает в себя все доработки, содержащиеся в сборке YDB 25.2.1.26, см. описание изменений. Кроме того, версия включает в себя ряд дополнительных доработок, портированных из актуальной версии 26.1.

Дополнительные исправления

Перечисленные ниже изменения были перенесены из версии 26.1 в поддерживаемые стабильные версии Корпоративной СУБД Яндекса:

- Исправлена ошибка, нарушавшая заданный в запросе порядок сортировки при обращении к системным таблицам.
- Исправлена ошибка в логике проверки целостности внутреннего состояния, которая в редких случаях могла приводить к единичному (не массовому) перезапуску узлов хранения.
- Добавлена оптимизация, позволяющая фильтровать строки по колонкам из индекса до запроса к основной таблице, что позволяет сократить количество обращений к основной таблице при выполнении некоторых типов запросов.
- Реализован набор исправлений в индексном доступе (StreamIndexLookup), устраняющий возможность проявления редких ситуаций «зависания» выполняемых запросов и сокращающий потребление оперативной памяти при выполнении запросов.
- Добавлена оптимизация, сокращающая потребление памяти при обработке запросов с операцией TopSort (`SELECT ... ORDER BY x LIMIT n`).
- Добавлена поддержка материализации индексов при резервном копировании и восстановлении.
- Сообщения об ошибках TLI (Transaction Locks Invalidated) во всех случаях снабжаются идентификатором либо путём затронутой таблицы.
- В статистики запроса, предоставляемые через системные таблицы `.sys/query_metrics_*`, добавлены метрики блокировок.
- Некорректные представления теперь могут быть восстановлены из резервной копии. Это позволяет восстанавливать резервные копии, созданные из БД, содержащей такие представления, без дополнительных действий со стороны администратора.

Версия 25.2.1.ent.4

Дата выхода: 12 февраля 2026.

Функциональность

- Аналитические возможности доступны по умолчанию: колоночные таблицы могут создаваться без включения специальных флагов, с использованием сжатия LZ4 и хеш-партиционирования. Поддерживаемые операции включают широкий набор DML (UPDATE, DELETE, UPSERT, INSERT INTO ... SELECT) и CREATE TABLE AS SELECT. Интеграция с dbt, Apache Airflow, Jupyter, Superset и федеративные запросы к S3 позволяют строить сквозные аналитические пайплайны в YDB.
- Стоимостной оптимизатор работает по умолчанию для запросов, использующих хотя бы одну колоночную таблицу, но может быть включён принудительно и для остальных запросов. Стоимостной оптимизатор улучшает производительность

выполнения запросов, вычисляя оптимальный порядок и тип соединений на основе статистики по таблицам; поддерживаемые hints позволяют тонко настраивать планы выполнения для сложных аналитических запросов.

- Реализован трансфер данных – асинхронный механизм переноса данных из топика в таблицу. Создание экземпляра трансфера, его изменение и удаление осуществляется с использованием YQL. Для быстрого старта воспользуйтесь инструкцией с примером.
- Добавлен спиллинг, механизм управления памятью, при котором промежуточные данные, возникающие в результате выполнения запросов и превышающие доступный объём оперативной памяти узла, временно выгружаются во внешнее хранилище. Спиллинг обеспечивает выполнение пользовательских запросов, которые требуют обработки больших объёмов данных, превышающих доступную память узла.
- Увеличено максимальное время на выполнение одного запроса с 30 минут до 2 часов.
- Добавлена поддержка Certificate Authority (CA) и Yandex Cloud Identity and Access Management (IAM) аутентификации в асинхронной репликации.
- Включены по умолчанию:
  - векторный индекс для приближённого векторного поиска;
  - поддержка в YDB Topics Kafka API клиентской балансировки читателей, компактифицированных топиков и транзакций;
  - поддержка автопартиционирования топиков в CDC для строковых таблиц;
  - поддержка автопартиционирования топиков для асинхронной репликации;
  - поддержка параметризованного типа Decimal;
  - поддержка типа DateTime64;
  - автоудаление временных директорий и таблиц при экспорте в S3;
  - поддержка потока изменений в операциях резервного копирования и восстановления;
  - возможность указания числа реплик для вторичного индекса;
  - системные представления с историей перегруженных партиций.

Исправления ошибок

- Исправлена ошибка в Workload Manager, из-за которой потребление CPU колоночными таблицами могло превышать установленные пределы.
- Исправлена проблема, из-за которой удаление таблетки могло зависать
- Исправлена ошибка вызывающая ошибку, при изменении follower'а таблицы
- Исправлен ряд ошибок, связанных с changefeed:
  - Исправлена ошибка, из-за которой импорт таблицы с Utf8-ключом и включённым changefeed мог завершиться неудачно
  - Исправлена ошибка, когда импорт таблицы без потоков изменений мог завершаться сбоем из-за некорректного поиска файлов changefeed
- Исправлена ошибка, которая могла приводить к сбоям при UPSERT-операциях в колоночных таблицах
- Исправлена ошибка, вызывавшая сбой из-за обращения к уже освобождённой памяти
- Исправлена ошибка с дубликатами в уникальных вторичных индексах
- Исправлена ошибка несовпадения контрольных сумм при восстановлении сжатых бэкапов из S3
- Исправлена ошибка, из-за которой некоторые запросы бенчмарка TPC-H 1000 могли завершаться с ошибкой
- Исправлен ряд проблем, связанных с инициализацией кластера:
  - Исправлена ошибка, из-за которой инициализация кластера могла зависать при обязательной авторизации
  - Исправлена проблема, из-за которой создание новых баз данных сразу после развёртывания кластера было невозможно в течение нескольких минут
- Исправлена ошибка, при которой мог возникать race condition и клиенты получали ошибку `Could not find correct token validator`, если использовались недавно выданные токены до обновления состояния `LoginProvider`

Версия 25.1

Версия 25.1.4.ent.8

Дата выхода: 12 февраля 2026.

Исправления ошибок

- Исправлена ошибка, при которой именованное выражение, содержащее другое именованное выражение, приводило к некорректному бэкапу `VIEW`.
- Исправлена ошибка, из-за которой сортировка по убыванию не работала корректно в запросах к системным представлениям.

Версия 25.1.4.ent.3

Дата выхода: 25 ноября 2025.

Функциональность

- Реализован векторный индекс для приближённого векторного поиска. Для векторного поиска опубликованы рецепты для YDB CLI и YQL, а также примеры работы на C++ и Python.
- Добавлена поддержка консистентной асинхронной репликации.
- Добавлен механизм конфигурации V2, упрощающий развёртывание новых кластеров YDB и дальнейшую работу с ними. Сравнение механизмов конфигурации V1 и V2.
- Добавлена поддержка параметризованного типа Decimal.
- Реализована клиентская балансировка партиций при чтении по протоколу Kafka (как у самой Kafka). Раньше балансировка происходила на сервере. Включается установкой флага `enable_kafka_native_balancing` в конфигурации кластера.
- Добавлена поддержка автопартиционирования топиков в CDC для строковых таблиц. Включается установкой флага `enable_topic_autopartitioning_for_cdc` в конфигурации кластера.
- Добавлена возможность изменить время хранения данных в CDC-топике с использованием выражения `ALTER TOPIC`.
- Поддержан формат DEBEZIUM_JSON для потоков изменений (changefeed).
- Добавлена возможность создавать потоки изменений к индексным таблицам.
- Добавлена возможность указания числа реплик для вторичного индекса. Включается установкой флага `enable_access_to_index_impl_tables` в конфигурации кластера.

- В операциях резервного копирования и восстановления поддержаны потоки изменений. Чтобы воспользоваться функциональностью, необходимо установить флаги `enable_changefeeds_export` и `enable_changefeeds_import` в секции `feature_flags` конфигурации базы данных или кластера.
- Добавлено автоудаление временных директорий и таблиц при экспорте в S3. Включается установкой флага `enable_export_auto_dropping` в конфигурации кластера.
- Добавлена автоматическая проверка целостности резервных копий при импорте, предотвращающая восстановление из повреждённых резервных копий и защищающая от потери данных.
- Добавлена возможность создания представлений, использующих UDF в запросах.
- Добавлены системные представления с информацией о настройках прав доступа, истории перегруженных партиций - включается установкой флага `enable_followers_stats` в конфигурации кластера, истории партиций строковых таблиц со сломанными блокировками (TLI).
- Добавлены новые параметры в операторы CREATE USER и ALTER USER:
  - `HASH` — возможность задания пароля в зашифрованном виде;
  - `LOGIN` и `NOLOGIN` — разблокировка и блокировка пользователя.
- Повышена безопасность учётных записей:
  - Добавлена проверка сложности пароля пользователя;
  - Реализована автоматическая блокировка пользователя при исчерпании лимита попыток ввода пароля;
  - Добавлена возможность самостоятельной смены пароля пользователем.
- Реализована возможность переключения функциональных флагов во время работы сервера YDB. Флаги, для которых в proto-файле не указан параметр `(RequireRestart) = true`, будут применяться без рестарта кластера.
- Теперь самые старые (а не новые) блокировки меняются на полношардовые при превышении количества блокировок на шардах.
- Реализовано сохранение оптимистичных блокировок в памяти при плавном перезапуске даташардов, что должно уменьшить число ошибок ABORTED из-за потери блокировок при балансировке таблиц между узлами.
- Реализована отмена волатильных транзакций со статусом ABORTED при плавном перезапуске даташардов.
- Добавлена возможность удалить `NOT NULL`-ограничения на столбец в таблице с помощью запроса `ALTER TABLE ... ALTER COLUMN ... DROP NOT NULL`.
- Добавлено ограничение в 100 000 на число одновременных запросов на создание сессий в сервисе координации.
- Увеличено максимальное число столбцов в первичном ключе с 20 до 30.
- Улучшена диагностика и интроспекция ошибок, связанных с памятью (#10419, #11968).
- ***(Экспериментально)*** Добавлен экспериментальный режим работы с более строгими проверками прав доступа. Включается установкой следующих флагов:
  - `enable_strict_acl_check` – не позволять выдавать права несуществующим пользователям и удалять пользователей, если им выданы права;
  - `enable_strict_user_management` — включает строгие правила администрирования локальных пользователей (т.е. администрировать локальных пользователей может только администратор кластера или базы данных);
  - `enable_database_admin` — добавляет роль администратора базы данных.
- Добавлена возможность использовать привычные инструменты потоковой обработки данных – Kafka Connect, Confluent Schema Registry, Kafka Streams, Apache Flink, AKH через Kafka API при работе с YDB Topics. Теперь YDB Topics Kafka API поддерживает:
  - клиентскую балансировку читателей – включается установкой флага `enable_kafka_native_balancing` в конфигурации кластера. Как работает балансировка читателей в Apache Kafka. Теперь балансировка читателей в Kafka API YDB Topics будет работать точно так же,
  - компактифицированные топики – включается установкой флага `enable_topic_compactification_by_key`,
  - транзакции – включается установкой флага `enable_kafka_transactions`.
- Добавлен новый протокол в Node Broker, устраняющий всплески сетевого трафика на больших кластерах (более 1000 серверов), связанного с рассылкой информации об узлах.

Изменения с потерей обратной совместимости

- Если вы используете запросы, в которых происходит обращение к именованным выражениям как к таблицам с помощью AS_TABLE, обновите temporal over YDB на версию v1.23.0-ydb-compat перед обновление YDB на текущую версию, чтобы избежать ошибок в выполнении таких запросов.

YDB UI

- В редактор запросов добавлена поддержка частичной загрузки результатов — отображение начинается сразу при получении первого фрагмента с сервера без ожидания полного завершения запроса. Это позволяет быстрее получать результаты.
- Улучшена безопасность: элементы управления, которые недоступны пользователю, теперь не отображаются в интерфейсе. Пользователи не будут сталкиваться с ошибками "Доступ запрещен".
- Добавлен добавлен поиск по идентификатору таблетки на вкладку "Tablets".
- Добавлена подсказка по горячим клавишам, которая открывается по комбинации `⌘+K`.
- На страницу базы данных добавлена вкладка "Операции", которая позволяет просматривать список операций и отменять их.
- Обновлена панель мониторинга кластера, добавлена возможность ее свернуть.
- Реализована поддержка поиска с учетом регистра в инструменте иерархического отображения JSON.
- На верхнюю панель после выбора базы данных добавлены примеры кода для подключения в YDB SDK, что ускоряет процесс разработки.
- Исправлена сортировка строк на вкладке Запросы.
- Удалены лишние запросы подтверждения при закрытии страницы браузера в редакторе запросов — подтверждение запрашивается только когда это необходимо.
- Исправлена ошибка, из-за которой не все таблетки отображались на вкладке Tablets в разделе диагностики.
- Исправлена ошибка, из-за которой на вкладке Storage в разделе диагностики базы данных отображались не только узлы хранения.
- Исправлена ошибка сериализации, которая могла приводить к падению при открытии статистики выполнения запроса.
- Изменена логика перехода узлов в критическое состояние – заполненный на 75-99% CPU pool теперь вызывает предупреждение, а не критическое состояние.

Производительность

- Добавлена поддержка свёртки констант в оптимизаторе запросов по умолчанию, что повышает производительность запросов за счёт вычисления константных выражений на этапе компиляции.
- Добавлен новый протокол гранулярного таймкаста, который позволит сократить время выполнения распределённых транзакций (замедление одного шарда не будет приводить к замедлению всех).
- Реализована функциональность сохранения состояния даташардов в памяти при перезапусках, что позволяет сохранить блокировки и повысить шансы на успешное выполнение транзакций. Это сокращает время выполнения длительных транзакций за счёт уменьшения числа повторных попыток.
- Реализована конвейерная обработка внутренних транзакций в Node Broker, что ускорило запуск динамических узлов в кластере YDB.
- Улучшена устойчивость Node Broker к повышенной нагрузке со стороны узлов кластера.
- Включены по умолчанию выгружаемые B-Tree-индексы вместо невыгружаемых SST-индексов, что позволяет сократить потребление памяти при хранении «холодных» данных.
- Оптимизировано потребление памяти узлами хранения.
- Сократили время запуска Hive до 30%.
- Оптимизирован процесс репликации в распределённом хранилище.
- Оптимизирован размер заголовка больших двоичных объектов в VDisk.
- Уменьшено потребление памяти за счёт очистки страниц аллокатора.
- Оптимизирована обработка пустых входов при выполнении JOIN-операций.

Исправления ошибок

- Исправлена ошибка в настройке Interconnect, приводящая к снижению производительности.
- Исправлена ошибка «Out of memory» при удалении очень больших таблиц за счёт регулирования числа одновременно обрабатывающих данную операцию таблеток.
- Исправлена ошибка, возникавшая при указании одного и того же узла базы данных несколько раз в конфигурации для системных таблеток.
- Устранена ошибка длительного (секунды) чтения данных при частых операциях перешардирования таблицы.
- Исправлена ошибка чтения из асинхронных реплик, приводившая к сбою.
- Исправлены редкие зависания при первоначальном сканировании CDC.
- Исправлена обработка незавершённых схемных транзакций в даташардах при перезапуске системы.
- Исправлена ошибка несогласованного чтения из топика при попытке явно подтвердить сообщение, прочитанное в рамках транзакции. Теперь пользователь при попытке подтвердить сообщение получит ошибку.
- Исправлена ошибка, из-за которой автопартиционирование некорректно работало при работе с топиком в транзакции.
- Исправлены зависания транзакций при работе с топиками во время перезапуска таблеток.
- Исправлена ошибка «Key is out of range» при импорте из S3-совместимого хранилища.
- Исправлено некорректное определение конца поля с метаданными в конфигурации кластера.
- Улучшено построение вторичных индексов: при возникновении некоторых ошибок система ретраит процесс, а не прерывает его.
- Исправлена ошибка выполнения выражения `RETURNING` в запросах `INSERT INTO` и `UPSERT INTO`.
- Исправлена проблема зависания операции «Drop Tablet» в PQ tablet, особенно во время задержек в работе Interconnect.
- Исправлена ошибка, возникавшая во время компакшна VDisk.
- Исправлена проблема, из-за которой длительные сессии чтения топика завершались с ошибками «too big inflight».
- Исправлено зависание при чтении топика, если хотя бы одна партиция не имела входящих данных, но читалась несколькими потребителями.
- Устранена редкая проблема перезагрузок PQ tablet.
- Устранена проблема, при которой после обновления версии кластера Hive запускались подписчики в датацентрах без работающих узлов баз данных.
- Исправлена ошибка `Failed to set up listener on port 9092 errno# 98 (Address already in use)`, возникавшая при обновлении версии.
- Исправлена ошибка, приводившая к segmentation fault при одновременном выполнении запроса к healthcheck и отключении узла кластера.
- Исправлен сбой в партиционировании строковой таблицы при выборе разделённого ключа из образцов доступа, содержащих смешанные операции с полным ключом и префиксом ключа (например, точное чтение или чтение диапазона).
- Исправлена ошибка, из-за которой тип индекса ошибочно определялся как `GLOBAL SYNC`, хотя в запросе явно указывался `UNIQUE`.
- Исправлена ошибка, из-за которой автопартиционирование топиков не работало, когда параметр конфигурации `max_active_partition` задавался с помощью выражения `ALTER TOPIC`.
- Исправлена ошибка, из-за которой `ydb scheme describe` возвращал список столбцов не в том порядке, в котором они были заданы при создании таблицы.
- Добавлена поддержка в асинхронной репликации нового типа записи об изменениях — `reset`-записи (в дополнение к `update`- и `erase`-записям).
- Исправлена ошибка, из-за которой экземпляр репликации с неуказанным параметром `COMMIT_INTERVAL` приводил к сбою процесса.
- Исправлены редкие ошибки при чтении из топика во время балансировки партиций.
- Исправлена ошибка, из-за которой при удалении dedicated-базы данных системные таблетки базы могли остаться неудалёнными.
- Исправлена ошибка, из-за которой таблетки могли зависать при недостатки памяти на узлах. Теперь таблетки будут автоматически запускаться, как только на каком-либо из узлов освободится достаточное количество ресурсов.
- Исправлена ошибка, из-за которой при записи сообщений Kafka сохранялось только первое сообщение из батча, а остальные сообщения игнорировались.

Версия 24.4

Версия 24.4.4.20

Дата выхода: 1 ноября 2025.

Функциональность

- В операциях резервного копирования и восстановления поддержаны представления (VIEW). Чтобы воспользоваться функциональностью, необходимо установить флаг `enable_view_export` в секции `feature_flags` конфигурации базы данных или кластера.
- В текст ошибок Transaction locks invalidated в случае невозможности идентифицировать таблицу (Unknown table) добавляются дополнительные идентификаторы: идентификатор пути к объекту (`PathId`) и идентификатор таблетки (`TabletId`).

Версия 24.4.4.15

Дата выхода: 19 сентября 2025.

Производительность

- Столбцы, по которым осуществляется сортировка результатов выполнения запроса, учитываются оптимизатором при автоматическом выборе вторичного индекса. Функциональность работает только для запросов к единственной таблице, без присоединения других таблиц.

Исправления ошибок

- При получении ошибки `OperationAborted` в ответе от S3 операция экспорта не завершается ошибкой, а повторяет попытку записи в S3.

Версия 24.4.4.13

Дата выхода: 29 июля 2025.

Функциональность

- Поддержан рестарт без потери доступности кластера в минимальной отказоустойчивой конфигурации из трех узлов.
- Добавлены новые функции UDF Roaring bitmap: AndNotWithBinary, FromUint32List, RunOptimize
- Добавлена возможность регистрировать узел базы данных по сертификату. В Node Broker добавлен флаг `AuthorizeByCertificate` использования сертификата при регистрации.
- Добавлены приоритеты проверки аутентификационных тикетов с использованием стороннего IAM-провайдера, с самым высоким приоритетом обрабатываются запросы от новых пользователей. Тикеты в кеше обновляют свою информацию с приоритетом ниже.
- Добавлена возможность чтения и записи в топик с использованием Kafka API без аутентификации.
- Включены по умолчанию:
  - поддержка представлений (VIEW);
  - режим автопартиционирования топиков;
  - транзакции с участием топиков и строковых таблиц;
  - волатильные распределённые транзакции.

Производительность

- Ускорено поднятие таблеток на больших кластерах: 210 мс → 125 мс (ssd), 260 мс → 165 мс (hdd).
- Ограничено количество одновременно обрабатываемых изменений конфигураций.
- Оптимизировано потребление памяти PQ-таблетками.
- Оптимизировано потребление CPU таблеткой Scheme shard, что уменьшило задержки ответов на запросы. Теперь лимит на число операций Scheme shard проверяется до выполнения операций разделения и слияния таблеток.
- Включен по умолчанию автоматический выбор вторичного индекса при выполнении запроса.

Исправления ошибок

- Исправлена ошибка, из-за которой при чтении из топика маленьких сообщений маленькими порциями значительно увеличивалась назгрузка на CPU. Это могло приводить к задержкам в чтении/записи в этот топик.
- Исправлена ошибка восстановления из резервной копии, которая была создана в момент автоматического разделения таблицы.
- Исправлена ошибка в сериализации `Uuid` для CDC.
- Исправлена ошибка, из-за которой чтение на подписчиках таблетки могло приводить к сбоям во время автоматического разделения таблицы.
- Исправлена ошибка, при которой узел координации успешно регистрировал прокси-серверы несмотря на разрыв связи.
- Исправлена ошибка, возникающие при открытии в интерфейсе вкладки с информацией о группах распределенного хранилища.
- Исправлена ошибка, из-за которой Health Check не сообщал о проблемах в синхронизации времени.
- Исправлена редкая ошибка зависания клиентских приложений во время выполнения коммита транзакции, когда удаление партиции совершалось раньше обновления квоты на запись в топик.
- Исправлена ошибка в копировании таблиц с типом Decimal, которая приводила к сбою при откате на предыдущую версию.
- Исправлена ошибка, при которой коммит без подтверждения записи в топик приводил к блокировке текущей и следующих транзакций с топиками.
- Исправлены зависания транзакций при работе с топиками при перезагрузке или удалении таблетки.
- Исправлены проблемы с чтением сообщений больше 6Mb через Kafka API.
- Устранена утечка памяти во время записи в топик.
- Исправлены ошибки обработки nullable столбцов и столбцов с типом UUID в строковых таблицах.
- Исправлена ошибка, которая приводила к существенному снижению скорости чтения с подписчиков таблетки.
- Исправлена ошибка, которая приводила к ожиданию подтверждения волатильной распределённой транзакции до следующего рестарта.
- Исправлена редкая ошибка, которая приводила к сбою при подключении подписчиков таблетки к лидеру с несогласованным состоянием журнала команд.
- Исправлена редкая ошибка, которая приводила к сбою при перезапуске удалённого datashard с неконсистентными изменениями.
- Исправлена ошибка, из-за которой мог нарушаться порядок обработки сообщений в топике.
- Исправлена редкая ошибка, из-за которой могло зависать чтение из топика.

- Исправлена проблема, из-за которой транзакция зависала при одновременном управлении топиком пользователем и перемещении PQ таблетки на другой узел.
- Исправлена проблема с утечкой значения счётчика для userInfo, которая могла приводить к ошибке чтения `too big in flight`.
- Исправлен сбой прокси-сервера из-за дублирования топиков в запросе.
- Исправлена редкая ошибка, из-за которой пользователь мог писать в топик в обход ограничений квоты аккаунта.
- Исправлена проблема, из-за которой после удаления топика система возвращала "OK", но его таблетки продолжали работать. Для удаления таких таблеток воспользуйтесь инструкцией из pull request.
- Исправлена редкая ошибка, из-за которой не восстанавливалась резервная копия большой таблицы с вторичным индексом.
- Исправлена проблема, которая приводила к ошибке при вставке данных с помощью `UPSERT` в строковые таблицы с значениями по умолчанию.
- Устранена ошибка, которая приводила к сбою при выполнении запросов к таблицам с вторичными индексами, возвращающих списки результатов с помощью выражения `RETURNING *`.

Версия 24.3

Версия 24.3.13.11

Дата выхода: 6 марта 2025.

Исправления ошибок

- Исправлена редкая проблема, которая приводила к утечкам незакоммиченных изменений.
- Исправлены проблемы с согласованностью, связанные с кэшированием удаленных диапазонов.
- Исправлена проблема длительного кеширования негативных ответов на запросы аутентификации от LDAP-совместимого каталога пользователей и групп.

Версия 24.3.13.10

Дата выхода: 24 декабря 2024.

Функциональность

- Добавлена трассировка запросов – инструмент, позволяющий детально посмотреть путь следования запроса по распределенной системе.
- Добавлена поддержка асинхронной репликации, которая позволяет синхронизировать данные между базами YDB почти в реальном времени. Также она может быть использована для миграции данных между базами с минимальным простоем работающих с ними приложений.
- Добавлена поддержка представлений (VIEW), которая может быть включена администратором кластера с помощью настройки `enable_views` в динамической конфигурации.
- В федеративных запросах поддержаны новые внешние источники данных: MySQL, Microsoft SQL Server, Greenplum.
- Разработана документация по разворачиванию YDB с функциональностью федеративных запросов (в ручном режиме).
- Для Docker-контейнера с YDB добавлен параметр запуска `FQ_CONNECTOR_ENDPOINT`, позволяющий указать адрес коннектора ко внешним источникам данных. Добавлена возможность TLS-шифрования соединения с коннектором. Добавлена возможность вывода порта сервиса коннектора, локально работающего на том же хосте, что и динамический узел YDB.
- Добавлен режим автопартиционирования топиков, в котором топики могут разбивать партиции в зависимости от нагрузки с сохранением гарантий порядка чтения сообщений и exactly once записи. Режим может быть включен администратором кластера с помощью настроек `enable_topic_split_merge` и `enable_pqconfig_transactions_at_scheme_shard` в динамической конфигурации.
- Добавлены транзакции с участием топиков и строковых таблиц. Таким образом, можно транзакционно перекладывать данные из таблиц в топики и в обратном направлении, а также между топиками, чтобы данные не терялись и не дублировались. Транзакции могут быть включены администратором кластера с помощью настроек `enable_topic_service_tx` и `enable_pqconfig_transactions_at_scheme_shard` в динамической конфигурации.
- Добавлена поддержка CDC для синхронных вторичных индексов.
- Добавлена возможность изменить период хранения записей в CDC топиках.
- Добавлена поддержка автоинкремента для колонок, включенных в первичный ключ таблицы.
- Добавлена запись в аудитный лог событий логина пользователей в YDB, событий завершения сессии пользователя в пользовательском интерфейсе, а также запроса бэкапа и восстановления из бэкапа.
- Добавлено системное представление, позволяющее получить информацию о сессиях, установленных с базой данных, с помощью запроса.
- Добавлена поддержка константных значений по умолчанию для колонок строковых таблиц.
- Добавлена поддержка выражения `RETURNING` в запросах.
- Добавлена встроенная функция `version()`.
- Добавлены время запуска/завершения и автор в метаданные операций резервного копирования/восстановления из S3-совместимого хранилища.
- Добавлена поддержка резервного копирования/восстановления из S3-совместимого хранилища ACL для таблиц.
- Для запросов, читающих из S3, в план добавлены пути и метод декомпрессии.
- Добавлены новые настройки парсинга для `timestamp`, `datetime` при чтении данных из S3.
- Добавлена поддержка типа `Decimal` в ключах партиционирования.
- Улучшена диагностика проблем хранилища в HealthCheck.
- ***(Экспериментально)*** Добавлен стоимостной оптимизатор для сложных запросов, где участвуют колоночные таблицы. Оптимизатор рассматривает большое количество альтернативных планов выполнения и выбирает из них лучший на основе оценки стоимости каждого варианта. На текущий момент оптимизатор работает только с планами, где есть операции JOIN.
- ***(Экспериментально)*** Реализована начальная версия менеджера рабочей нагрузки, который позволяет создавать пулы ресурсов с ограничениями по процессору, памяти и количеству активных запросов. Реализованы классификаторы ресурсов для отнесения запросов к определенному пулу ресурсов.
- ***(Экспериментально)*** Реализован автоматический выбор индекса при выполнении запроса, который может быть включен администратором кластера с помощью настройки `index_auto_choose_mode` в `table_service_config` в динамической конфигурации.

YDB UI

- Поддержано создание и отображение экземпляра асинхронной репликации.
- Добавлено обозначение столбцов с автоинкрементом.
- Добавлена вкладка с информацией о таблетках.
- Добавлена вкладка с информацией о группах распределенного хранилища.
- Добавлена настройка для добавления трассировки ко всем запросам и отображение результатов трассировки запроса.
- На страницу PDisk добавлены атрибуты, информация о потреблении дискового пространства, а также кнопка, которая запускает декомиссию диска.
- Добавлена информация о выполняющихся запросах.
- Добавлена настройка лимита строк в выдаче для редактора запроса и отображение, если результаты запроса превысили лимит.
- Добавлено отображение перечня запросов с максимальным потреблением CPU за последний час.
- Добавлен поиск на страницах с историей запросов и списком сохраненных запросов.
- Добавлена возможность прервать исполнение запроса.
- Добавлена возможность сохранять запрос из редактора горячими клавишами.
- Разделено отображение дисков от дисков-доноров.
- Добавлена поддержка InterruptInheritance ACL и улучшено отображение действующих ACL.
- Добавлено отображение текущей версии пользовательского интерфейса.
- Добавлена с информацией о состоянии настроек включения экспериментальной функциональности.

Производительность

- Ускорено восстановление из бэкапа таблиц со вторичными индексами до 20% по нашим тестам.
- Оптимизирована пропускная способность Interconnect.
- Улучшена производительность CDC-топиков, содержащих тысячи партиций.
- Сделан ряд улучшений алгоритма балансировки таблеток Hive.

Исправления ошибок

- Исправлена ошибка, которая приводила в неработоспособное состояние базу с большим количеством таблиц или партиций при восстановлении из резервной копии. Теперь при превышении лимитов на размер базы, операция восстановления завершится ошибкой, база продолжит работать в штатном режиме.
- Реализован механизм, принудительно запускающий фоновый компакшн при обнаружении несоответствий между схемой данных и данными, хранящимися в DataShard. Это решает редко возникающую проблему задержки в изменении схемы данных.
- Устранено дублирование аутентификационных тикетов, которое приводило к повышенному числу запросов в провайдеры аутентификации.
- Исправлена ошибка нарушения инварианта при первоначальном сканировании CDC, приводившая к аварийному завершению серверного процесса ydbd.
- Запрещено изменение схемы таблиц резервного копирования.
- Исправлено зависание первоначального сканирования CDC при частых обновлениях таблицы.
- Исключены удаленные индексы из подсчета лимита на максимальное количество индексов.
- Исправлена ошибка в отображении времени, на которое запланировано выполнение набора транзакций (планируемый шаг).
- Исправлена проблема прерывания blue–green deployment в больших кластерах, возникающая из-за частого обновления списка узлов.
- Исправлена редко возникающая ошибка, которая приводила к нарушению порядка выполнения транзакций.
- Исправлена ошибка в EvWrite API, которая приводила к некорректному освобождению памяти.
- Исправлена проблема зависания волатильных транзакций после перезапуска.
- Исправлена ошибка в CDC, приводящая в некоторых случаях к повышенному потреблению CPU, вплоть до ядра на одну CDC-партицию.
- Устранена задержка чтения, возникающая во время и после разделения некоторых партиций.
- Исправлены ошибки при чтении данных из S3.
- Исправлен способ расчета aws signature при обращении к S3.
- Исправлены ложные срабатывания системы HealthCheck в момент бэкапа базы с большим количеством шардов.
- Снято ограничение на запись в тип Uint8 значений больше 127.
- Исправлена ошибка восстановления из резервной копии, сохраненной в хранилище S3 с Path-style адресацией.
- Исправлена потенциальная поломка "замороженных" блокировок, к которой могли приводить массовые операции (например, удаление по TTL).
- Исправлена редкая проблема, которая приводила к ошибкам при выполнении запроса на чтение.

Версия 24.2

Версия 24.2.7.1

Дата выхода: 20 августа 2024.

Функциональность

- Добавлена возможность задать приоритеты задачам обслуживания в системе управления кластером.
- Добавлена настройка стабильных имён для узлов кластера в рамках тенанта.
- Добавлено получение вложенных групп от LDAP-сервера, в LDAP-конфигурации улучшен парсинг хостов и добавлена настройка для отключения встроенной аутентификацию по логину и паролю.
- Добавлена возможность аутентификации динамических узлов по SSL-сертификату.
- Реализовано удаление неактивных узлов из Hive без его перезапуска.
- Улучшено управление inflight pings при перезапуске Hive в кластерах большого размера.
- Изменен порядок установления соединения с узлами при перезапуске Hive.

YDB UI

- Добавлена возможность задать TTL для сессии пользователя в конфигурационном файле.

- Добавлена сортировка по `CPUTime` в таблицу со списком запросов.
- Исправлена потеря точности при работе с `double`, `float`.
- Поддержано создание директорий из UI.
- Добавлена возможность задать интервал фонового обновления данных на всех страницах.
- Улучшено отображения ACL.
- Включено автодополнение в редакторе запросов по умолчанию.
- Добавлена поддержка View.

Исправления ошибок

- Добавлена проверка на размер локальной транзакции до ее коммита, чтобы исправить ошибки в работе схемных операции при выполнении экспорта/бекапа больших баз.
- Исправлена ошибка дублирования результатов SELECT-запроса при уменьшении квоты в DataShard.
- Исправлены ошибки, возникающие при изменении состояния координатора.
- Исправлены ошибки, возникающие в момент первичного сканирования CDC.
- Исправлено состояние гонки в асинхронной доставке изменений (асинхронные индексы, CDC).
- Исправлена редкая ошибка, из-за которой удаление по TTL приводило к аварийному завершению процесса.
- Исправлена ошибка отображения статуса PDisk в интерфейсе CMS.
- Исправлены ошибки, из-за которых мягкий перенос (drain) таблеток с узла мог зависать.
- Исправлена ошибка остановки interconnect proxy на узле, работающем без перезапусков, при добавлении другого узла в кластер.
- Исправлен учет свободной памяти в interconnect.
- Исправлены счетчики UnreplicatedPhantoms/UnreplicatedNonPhantoms в VDisk.
- Исправлена обработка пустых запросов сборки мусора на VDisk.
- Исправлено управление настройками TVDiskControls через CMS.
- Исправлена ошибка загрузки данных, созданных более новыми версиями VDisk.
- Исправлена ошибка выполнении запроса `REPLACE INTO` со значением по умолчанию.
- Исправлена ошибка исполнения запросов, в которых выполнялось несколько left join'ов к одной строковой таблице.
- Исправлена потеря точности для `float`, `double` типов при использовании CDC.

Версия 24.1

Версия 24.1.18.1

Дата выхода: 31 июля 2024.

Функциональность

- Реализована Knn UDF для точного поиска ближайших векторов.
- Разработан gRPC сервис QueryService, обеспечивающий возможность выполнения всех типов запросов (DML, DDL) и выборку неограниченных объёмов данных.
- Реализована интеграция с LDAP протоколом и возможность получения перечня групп из внешних LDAP-каталогов.

Встроенный UI

- Добавлен дашборд диагностики потребления ресурсов, который находится на вкладке с информацией о базе данных и позволяет определить текущее состояние потребление основных ресурсов: ядер процессора, оперативной памяти и места в сетевом распределенном хранилище.
- Добавлены графики для мониторинга основных показателей работы кластера YDB.

Производительность

- Оптимизированы таймауты сессий сервиса координации от сервера до клиента. Ранее таймаут составлял 5 секунд, что в худшем случае приводило к определению неработающего клиента (и освобождению удерживаемых им ресурсов) в течение 10 секунд. В новой версии время проверки зависит от времени ожидания сеанса, что обеспечивает более быстрое реагирование при смене лидера или захвате распределённых блокировок.
- Оптимизировано потребление CPU репликами SchemeShard, особенно при обработке быстрых обновлений для таблиц с большим количеством партиций.

Исправления ошибок

- Исправлена ошибка возможного переполнения очереди, Change Data Capture резервирует емкость очереди изменений при первоначальном сканировании.
- Исправлена потенциальная взаимоблокировка между получением записей CDC и их отправкой.
- Исправлена проблема потери очереди задач медиатора при переподключении медиатора, исправление позволяет обработать очередь задач медиатора при ресинхронизации.
- Исправлена редко возникающая ошибка, когда при включённых и используемых волатильных транзакциях возвращался успешный результат подтверждения транзакции до того, как она была успешно закоммичена. Волатильные транзакции по умолчанию выключены, находятся в разработке.
- Исправлена редко возникающая ошибка, приводившая к потере установленных блокировок и успешному подтверждению транзакций, которые должны были завершиться ошибкой Transaction Locks Invalidated.
- Исправлена редкая ошибка, приводящая к возможному нарушению гарантий целостности данных при конкурентной записи и чтении данных по определённому ключу.
- Исправлена проблема, из-за которой реплики для чтения переставали обрабатывать запросы.
- Исправлена редкая ошибка, которая могла привести к аварийному завершению процессов базы данных при наличии неподтверждённых транзакций над таблицей в момент её переименования.
- Исправлена ошибка в логике определения статуса статической группы, когда статическая группа не помечалась нерабочей, хотя должна была.
- Исправлена ошибка частичного коммита распределённой транзакции с незакоммиченными изменениями в случае некоторых гонок с рестартами.
- Исправлены аномалии с чтением устаревших данных, которые были обнаружены с помощью Jepsen.

Версия 23.4

Версия 23.4.11.1

Дата выхода: 14 мая 2024.

Производительность

- Исправлена проблема повышенного потребления вычислительных ресурсов актором топиков `PERSQUEUE_PARTITION_ACTOR`.
- Оптимизировано использование ресурсов репликами SchemeBoard. Наибольший эффект заметен при модификации метаданных таблиц с большим количеством партиций.

Исправления ошибок

- Исправлена ошибка возможной неполной фиксации накопленных изменений при использовании распределенных транзакций. Данная ошибка возникает при крайне редкой комбинации событий, включающей в себя перезапуск таблеток, обслуживающих вовлеченные в транзакцию партиции таблиц.
- Устранена гонка между процессами слияния таблиц и сборки мусора, из-за которой сборка мусора могла завершиться ошибкой нарушения инвариантов и, как следствие, аварийным завершением серверного процесса `ydbd`.
- Исправлена ошибка в Blob Storage, из-за которой информация о смене состава группы хранения могла не поступать своевременно на отдельные узлы кластера. В результате в редких случаях могли блокироваться операции чтения и записи данных, хранящихся в затронутой группе, и требовалось ручное вмешательство администратора.
- Исправлена ошибка в Blob Storage, из-за которой при корректной конфигурации могли не запускаться узлы хранения данных. Ошибка проявлялась для систем с явным образом включенной экспериментальной функцией "blob depot" (по умолчанию эта функция выключена).
- Исправлена ошибка, возникавшая в некоторых ситуациях записи в топик с пустым `producer_id` при выключенной дедупликации. Она могла приводить к аварийному завершению серверного процесса `ydbd`.
- Исправлена проблема, приводящая к падению процесса `ydbd` из-за ошибочного состояния сессии записи в топик.
- Исправлена ошибка отображения метрики количества партиций в топике, ранее в ней отображалось некорректное значение.
- Устранены утечки памяти, которые проявлялись при копировании данных топиков между кластерами YDB. Они могли приводить к завершению серверных процессов `ydbd` из-за исчерпания доступной оперативной памяти.

Версия 23.3

Версия 23.3.25.2

Дата выхода: 12 октября 2023.

Функциональность

- Реализована видимость собственных изменений внутри транзакций. Ранее при попытке прочитать данные, уже модифицированные в текущей транзакцией, запрос завершался ошибкой. Это приводило к необходимости упорядочивать чтения и записи внутри транзакции. С появлением видимости собственных изменений эти ограничения снимаются, и запросы могут читать измененные в данной транзакции строчки.
- Добавлена поддержка колоночных таблиц. Колоночные таблицы хорошо подходят для работы с аналитическими запросами (Online Analytical Processing), так как при выполнении запроса считываются только те столбцы, которые непосредственно участвуют в запросе. Колоночные таблицы YDB позволяют создавать аналитические отчёты с производительностью, сопоставимой со специализированными аналитическими СУБД.
- Добавлена поддержка Kafka API для топиков. Теперь с YDB-топиками можно работать через Kafka-совместимый API, предназначенный для миграции существующих приложений. Обеспечена поддержка протокола Kafka версии 3.4.0.
- Добавлена возможность записи в топик без дедупликации. Такой вид записи хорошо подходит для случаев, когда порядок обработки сообщений не критичен. Запись без дедупликации работает быстрее и потребляет меньше ресурсов на сервере, но упорядочение и дедупликация сообщений на сервере не происходит.
- В YQL добавлены возможности создавать, изменять и удалять топики.
- Добавлена возможность назначать и отзывать права доступа с помощью команд YQL GRANT и REVOKE.
- Добавлена возможность логгировать DML-операции в аудитном логе.
- ***(Экспериментально)*** При записи сообщений в топик теперь можно передавать метаданные. Для включения этой функциональности добавьте `enable_topic_message_meta: true` в конфигурационный файл.
- ***(Экспериментально)*** Добавлена возможность чтения из топиков и запись в таблицу в рамках одной транзакции. Новая возможность упрощает сценарий переноса данных из топика в таблицу. Для её включения добавьте `enable_topic_service_tx: true` в конфигурационный файл.
- ***(Экспериментально)*** Добавлена поддержка совместимости с PostgreSQL. Новый механизм позволяет выполнять SQL запросы в PostgreSQL диалекте на инфраструктуре YDB с использованием сетевого протокола PostgreSQL. Можно использовать привычные инструменты работы с PostgreSQL, такие, как psql и драйверы (pq для Golang и psycopg2 для Python), а также разрабатывать запросы на привычном PostgreSQL синтаксисе с горизонтальной масштабируемостью и отказоустойчивостью YDB.
- ***(Экспериментально)*** Добавлена поддержка федеративных запросов. Она позволяет получать информацию из различных источников данных без их переноса в YDB. Поддерживается взаимодействие с ClickHouse, PostgreSQL, S3 через YQL-запросы без дублирования данных между системами.

Встроенный UI

- В настройках селектора типа запроса добавлена новая опция `PostgreSQL`, которая доступна при включении параметра `Enable additional query modes`. Также в истории запросов теперь учитывается синтаксис, используемый при выполнении запроса.
- Обновлен шаблон YQL-запроса для создания таблицы. Добавлено описание доступных параметров.
- Сортировка и фильтрация для таблиц Storage и Nodes вынесена на сервер. Необходимо включить параметр `Offload tables filters and sorting to backend` в разделе экспериментов, чтобы использовать данный функционал.
- В контекстное меню были добавлены кнопки для создания, изменения и удаления топиков.
- Добавлена сортировка по критичности для всех issues в дереве в `Healthcheck`.

Производительность

- Реализованы итераторные чтения. Новая функциональность позволяет разделить чтения и вычисления между собой. Итераторные чтения позволяют даташардам увеличить пропускную способность читающих запросов.

- Оптимизирована производительность записи в топики YDB.
- Улучшена балансировка таблеток при перегрузке нод.

Исправления ошибок

- Исправлена ошибка возможной блокировки читающими итераторами снепшотов, о которых не знают координаторы.
- Исправлена утечка памяти при закрытии соединения в kafka proxy.
- Исправлена ошибка, при которой снепшоты, взятые через читающие итераторы, могут не восстанавливаться на рестартах.
- Исправлен некорректный residual предикат для условия `IS NULL` на колонку.
- Исправлена срабатывающая проверка `VERIFY failed: SendResult(): requirement ChunksLimiter.Take(sendBytes) failed`.
- Исправлен `ALTER TABLE` по `TTL` для колоночных таблиц.
- Реализован `FeatureFlag`, который позволяет отключать/включать работу с `CS` и `DS`.
- Исправлено различие координаторного времени между 23-2 и 23-3 на 50мс.
- Исправлена ошибка, при которой ручка `storage` возвращала лишние группы, когда в запросе параметр `node_id` во `viewer backend`.
- Добавлен `usage` фильтр в `/storage` во `viewer backend`.
- Исправлена ошибка в Storage v2, при которой возвращалось некорректное число в `Degraded`.
- Исправлена отмена подписки от сессий в итераторных чтениях при рестарте таблетки.
- Исправлена ошибка, при которой во время роллинг-рестарта при походе через балансер моргает `healthcheck` алертами про storage.
- Обновлены метрики `cpu usage` в ydb.
- Исправлено игнорирование `NULL` при указании `NOT NULL` в схеме таблицы.
- Реализован вывод записей об операциях `DDL` в общий лог.
- Реализован запрет для команды `ydb table attribute add/drop` работать с любыми объектами, кроме таблиц.
- Отключён `CloseOnIdle` для `interconnect`.
- Исправлено задваивание скорости чтения в UI.
- Исправлена ошибка, при которой могли теряться данные на `block-4-2`.
- Добавлена проверка имени топика.
- Исправлен возможный `deadlock` в акторной системе.
- Исправлен тест `KqpScanArrowInChanels::AllTypesColumns`.
- Исправлен тест `KqpScan::SqlInParameter`.
- Исправлены проблемы параллелизма для OLAP-запросов.
- Исправлена вставка `ClickBench parquet`.
- Добавлен недостающий вызов `CheckChangesQueueOverflow` в общем `CheckDataTxReject`.
- Исправлена ошибка возврата пустого статуса при вызовах `ReadRows API`.
- Исправлен некорректны ретрай экспорта в финальной стадии.
- Исправлена проблема с бесконечной квотой на число записей в CDC-топике.
- Исправлена ошибка импорта колонки `string` и `parquet` в колонку `string` OLAP.
- Исправлено падение `KqpOlapTypes.Timestamp` под tsan.
- Исправлено падение во `viewer backend` при попытке выполнить запрос к базе из-за несовместимости версий.
- Исправлена ошибка, при которой `viewer` не возвращал ответ от `healthcheck` из-за таймаута.
- Исправлена ошибка, при которой в Pdisk'ах могло сохраняться некорректное значение `ExpectedSerial`.
- Исправлена ошибка, при которой ноды базы падают по `segfault` в S3 акторе.
- Исправлена гонка в `ThreadSanitizer: data race KqpService::ToDictCache-UseCache`.
- Исправлена гонка в `GetNextReadId`.
- Исправлено завышение результата `SELECT COUNT(*)` сразу после импорта.
- Исправлена ошибка, при которой `TEvScan` мог вернуть пустой набор данных в случае сплита даташарда.
- Добавлен отдельный issue/код ошибки в случае исчерпания доступного места.
- Исправлена ошибка `GRPC_LIBRARY Assertion failed`.
- Исправлена ошибка, при которой при чтении по вторичному индексу в сканирующих запросах получался пустой результат.
- Исправлена валидация `CommitOffset` в `TopicAPI`.
- Уменьшено потребление `shared cache` при приближении к OOM.
- Смержена логика планировщиков из `data executer` и `scan executer` в один класс.
- Добавлены ручки `discovery` и `proxy` в процесс выполнения `query` во `viewer backend`.
- Исправлена ошибка, при которой ручка `/cluster` возвращает название корневого домена типа `/ru` во `viewer backend`.
- Реализована схема бесшовного обновления табличек для `QueryService`.
- Исправлена ошибка, при которой `DELETE` возвращал данные и НЕ удалял их.
- Исправлена ошибка работы `DELETE ON` в `query service`.
- Исправлено неожиданное выключение батчинга в дефолтных настройках схемы.
- Исправлена срабатывающая проверка `VERIFY failed: MoveUserTable(): requirement move.ReMapIndexesSize() == newTableInfo->Indexes.size()`.
- Увеличен дефолтный таймаут grpc-сриминга.
- Исключены неиспользуемые сообщения и методы из `QueryService`.
- Добавлена сортировка по `Rack` в `/nodes` во `viewer backend`.
- Исправлена ошибка, при которой запрос с сортировкой возвращает ошибку при убывании.
- Исправлено взаимодействие `QP` с `NodeWhiteboard`.
- Удалена поддержка старых форматов параметров.
- Исправлена ошибка, при которой `DefineBox` не применялся для дисков, на которых есть статическая группа.

- Исправлена ошибка `SIGSEGV` в диннодах при импорте `CSV` через `YDB CLI`.
- Исправлена ошибка с падением при обработке `NGRpcService::TRefreshTokenImpl`.
- Реализован `gossip` протокол обмена информацией о ресурсах кластера.
- Исправлена ошибка:

```
DeserializeValuePickleV1(): requirement data.GetTransportVersion() ==
(ui32) NDqProto::DATA_TRANSPORT_UV_PICKLE_1_0 failed
```

- Реализованы автоинкрементные колонки.
- Использовать статус `UNAVAILABLE` вместо `GENERIC_ERROR` при ошибке идентификации шарда.
- Добавлена поддержка `rope payload` в `TEvVGet`.
- Добавлено игнорирование устаревших событий.
- Исправлено падение write-сессий на невалидном имени топика.
- Исправлена ошибка:

```
CheckExpected(): requirement newConstr failed, message: Rewrite error,
missing Distinct((id)) constraint in node FlatMap
```

- Включён `self heal` по умолчанию.

Список изменений YDB CLI

Версия 2.33.0

Дата выхода 23 июня 2026. Для обновления до версии **2.33.0** перейдите в раздел Загрузки.

Функциональность

- AI-ассистент в интерактивном режиме `ydb` теперь умеет искать информацию в документации YDB, что делает ответы более точными.

Версия 2.32.0

Дата выхода 22 июня 2026. Для обновления до версии **2.32.0** перейдите в раздел Загрузки.

Функциональность

- Добавлен AI-режим в интерактивный режим `ydb`. Для переключения нажмите `Ctrl+T`.
- Добавлена команда `ydb workload fulltext` для нагрузочного тестирования и оценки качества полнотекстовых индексов.
- Добавлена команда `SET resource_pool` в интерактивный режим `ydb`.
- Добавлена опция `--resource-pool` в команду `ydb sql`.
- Добавлена опция `--no-consumer` в команду `ydb topic read` для чтения из топика без читателя.
- Добавлены опции `--partition-write-speed-mps` и `--partition-write-burst-messages` в команду `ydb topic create` и команду `ydb topic alter`.
- Добавлена опция `--partition-max-inflight-bytes` в команду `ydb workload topic`.
- Добавлены опции `--codec kafka-batch`, `--batch-inner-codec`, `--batch-flush-interval`, `--batch-flush-size` и `--batch-flush-message-count` в команды `ydb workload topic run write|full`.
- Добавлена статистика процессорного времени (CPU time) в команды `ydb workload * run`.

Улучшения

- Команда `ydb scheme describe` теперь выводит понятное человеку описание внешних источников данных (тип источника, расположение, способ аутентификации, база данных, свойства и время создания) вместо пустого вывода.

Версия 2.31.0

Дата выхода 20 апреля 2026. Для обновления до версии **2.31.0** перейдите в раздел Загрузки.

Функциональность

- Добавлена опция `--stats` в команды `ydb workload * run`, позволяющая включить сбор расширенной статистики выполнения (например, `--stats profile`).

Версия 2.30.0

Дата выхода 7 апреля 2026. Для обновления до версии **2.30.0** перейдите в раздел Загрузки.

Функциональность

- Добавлена команда `ydb config completion` для генерации скриптов автодополнения команд для bash и zsh.
- Добавлены команды `ydb export nfs` и `ydb import nfs`, позволяющие создавать и восстанавливать резервные копии непосредственно в/из общей NFS-директории, примонтированной на каждом хосте кластера.
- Добавлена опция `--compact` в команду `ydb workload tpcc import`.
- Добавлена опция `--tx-mode` в команды `ydb workload * run`, позволяющая задать режим транзакции (например, `no-tx`, `serializable-rw`, `snapshot-rw`).
- Добавлена поддержка новой операции компакшна в подкомандах `ydb operation`.

Улучшения

- При явном указании профиля с помощью опции `-p` / `--profile` активный профиль больше не используется: все параметры берутся только из указанного профиля, переменных окружения и командной строки. Это устраняет путаницу, когда выбранный профиль неожиданно дополнялся настройками из активного профиля.

Версия 2.29.0

Дата выхода 11 февраля 2026. Для обновления до версии **2.29.0** перейдите в раздел Загрузки.

Функциональность

- Улучшения интерактивного режима `ydb`:
  - Добавлена команда `/help` для получения справки по командам интерактивного режима.
  - Добавлена команда `/config`, предоставляющая интерактивный диалог для просмотра и изменения параметров YDB CLI:
    - Включение/выключение подсказок автодополнения.
    - Включение/выключение цветного вывода.
    - Интерактивный выбор цветовой темы из набора предустановленных вариантов с возможностью клонирования и настройки собственной темы.
- Добавлен прогресс-бар загрузки в команду `ydb update`.
- Добавлена опция `--include-index-data` в команду `ydb export s3`, позволяющая экспортировать данные индексов.
- Добавлена опция `--index-population-mode` в команду `ydb import s3`, позволяющая выбрать режим наполнения индекса (например, `build` или `import`).
- Добавлены поля `Created by`, `Create time` и `End time` для операций "build index" и "execute script" в подкомандах `ydb operation`.
- Добавлена поддержка единого формата временных интервалов во всех командах YDB CLI. Опции, принимающие временные интервалы, теперь поддерживают явное указание единиц времени (например, `5s`, `2m`, `1h`) с сохранением

обратной совместимости с числовыми значениями без суффикса, которые интерпретируются в исходных единицах по умолчанию.
- В описании семейств столбцов команды `ydb scheme describe` устаревшее поле "Keep in memory" заменено на "Cache mode".

Улучшения

- Улучшены команды `ydb init` и `ydb config profile` с добавлением интерактивных меню.
- Улучшены прогресс-бары: единообразные единицы MiB/GiB, стабильное отображение скорости, двойной прогресс-бар для команды `ydb import file`, показывающий как отправленные, так и подтверждённые байты.

Исправления ошибок

- Исправлена ошибка Out Of Memory в команде `ydb workload query run` при выполнении запросов с большими наборами результатов.
- Исправлен разбор параметров аутентификации по логину и паролю: пароль из профиля больше не используется, если имя пользователя получено из другого источника.

Версия 2.28.0

Дата выхода 19 декабря 2025. Для обновления до версии **2.28.0** перейдите в раздел Загрузки.

Функциональность

- Добавлены режимы транзакций `snapshot-ro` и `snapshot-rw` в опцию `--tx-mode` команды `ydb table query execute`.
- Добавлена поддержка переменной окружения `NO_COLOR` для отключения ANSI-цветов в YDB CLI (см. no-color.org).
- Добавлен простой прогресс-бар для неинтерактивного stderr.
- Добавлено свойство `omit-indexes` в опцию `--item` команды `ydb tools copy`, позволяющее копировать таблицы без индексов.
- Добавлена подкоманда `import files` в команду `ydb workload vector` для заполнения таблицы из CSV или Parquet файлов.
- Добавлена подкоманда `import generate` в команду `ydb workload vector` для заполнения таблицы случайными данными.
- ***(Требуется сервер v26.1+)*** Изменения в ранее добавленной команде `ydb admin cluster state fetch`:
  - Переименована в `ydb admin cluster diagnostics collect`.
  - Добавлена опция `--no-sanitize`, отключающая санитизацию и сохраняющая чувствительные данные в выводе.
  - Добавлена опция `--output` для указания пути к выходному файлу `.tar.bz2`.

Исправления ошибок

- Исправлена ошибка, при которой команда `ydb tools restore` могла аварийно завершаться с ошибкой `mutex lock failure (Invalid argument)` из-за внутреннего состояния гонки.
- Исправлено восстановление представлений (views), содержащих именованные выражения, и представлений, обращающихся к вторичным индексам, в команде `ydb tools restore`.

Версия 2.27.0

Дата выхода 30 октября 2025. Для обновления до версии **2.27.0** перейдите в раздел Загрузки.

Функциональность

- Добавлена опция `--exclude` в команду `ydb import s3`, позволяющая исключать схемные объекты из импорта по шаблону имени.
- Добавлена поддержка объектов типа трансфер при выполнении команды `ydb tools dump` и команды `ydb tools restore`.
- Добавлена новая опция `--retention-period` в подкоманды `ydb topic`. Использование устаревшей опции `--retention-period-hours` не рекомендуется.
- В команде `ydb topic consumer add` появилась новая опция `--availability-period`, которая переопределяет гарантию удержания для читателя.
- В командах `ydb workload vector` добавлены подкоманды `build-index` и `drop-index`.
- ***(Требуется сервер v26.1+)*** Добавлена команда `ydb admin cluster state fetch` для сбора информации о состоянии узлов кластера и метриках.

Исправления ошибок

- Исправлена ошибка, из-за которой команда `ydb debug ping` аварийно завершалась.

Версия 2.26.0

Дата выхода 25 сентября 2025. Для обновления до версии **2.26.0** перейдите в раздел Загрузки.

Функциональность

- Добавлены опции `--no-merge` и `--no-cache` в команду `ydb monitoring healthcheck`.
- Добавлена статистика времени компиляции запроса в команды `ydb workload * run`.
- Добавлена опция `--retries` в команду `ydb tools restore`, позваляющая задать количество повторных попыток для каждого запроса загрузки данных.
- ***(Требуется сервер v25.4+)*** Добавлена опция `--replace-sys-acl` в команду `ydb tools restore`, которая задаёт, нужно ли заменять ACL для системных объектов.

Версия 2.25.0

Дата выхода 1 сентября 2025. Для обновления до версии **2.25.0** перейдите в раздел Загрузки.

Функциональность

- Добавлена финальная статистика выполнения в команды `ydb workload * run`.

- Добавлена опция `--start-offset` в команду `ydb topic read`, которая задаёт начальную позицию для чтения из выбранной партиции.
- ***(Требуется сервер v25.3+)*** Добавлен новый способ указания путей в командах `ydb export s3` и `ydb import s3` с новой опцией `--include` вместо опции `--item`.
- ***(Требуется сервер v25.3+)*** Добавлена поддержка функций шифрования в командах `ydb export s3` и `ydb import s3`.
- ***(Требуется сервер v25.3+) (Экспериментально)*** Добавлены команды `ydb admin cluster bridge` для управления кластером в режиме bridge: `list`, `switchover`, `failover`, `takedown`, `rejoin`.

Улучшения

- Опции аутентификации по имени пользователя и паролю теперь обрабатываются независимо, что позволяет получать их из разных источников приоритета. Например, имя пользователя можно указать с помощью опции `--user`, а пароль получить из переменной окружения `YDB_PASSWORD`.
- Изменён уровень логирования по умолчанию с `EMERGENCY` на `WARN` для команд, поддерживающих несколько уровней логирования.

Изменения с потерей обратной совместимости

- Удалена опция `--float-mode` из команд `ydb workload tpch run` и `ydb workload tpcds run`. Режим работы с вещественными числами теперь определяется автоматически из схемы таблицы, созданной во время фазы `init`.

Исправления ошибок

- Исправлена ошибка, при которой команда `ydb import file csv` с опцией `--newline-delimited` могла зависать при некорректных входных данных.
- Исправлена ошибка с отображением прогресс-бара в команде `ydb workload clickbench import files` — неправильное процентное значение и избыточные переносы строк, приводящие к дублированию строк прогресса.
- Исправлена ошибка, при которой команда `ydb workload topic write` могла завершаться аварийно с ошибкой `Unknown AckedMessageId` из-за внутреннего состояния гонки.
- Исправлено сравнение десятичных типов в командах `ydb workload * run`.

Версия 2.24.1

Дата выхода 28 июля 2025. Для обновления до версии **2.24.1** перейдите в раздел Загрузки.

Исправления ошибок

- Исправлена ошибка, при которой команда `ydb tools dump` без уведомления пропускала объекты схемы неподдерживаемых типов и создавала для них пустые директории в целевой папке на файловой системе.

Версия 2.24.0

Дата выхода 23 июля 2025. Для обновления до версии **2.24.0** перейдите в раздел Загрузки.

Функциональность

- Добавлена возможность для команд `ydb workload tpch` и `ydb workload tpcds` использовать опцию `--scale` с дробным значением, что позволяет задавать долю от полного объёма данных и нагрузки бенчмарка.
- Добавлена команда `ydb workload tpcc check` для проверки целостности данных TPC-C.

Улучшения

- Тип хранения по умолчанию в командах `ydb workload * init` изменён на `column` (было `row`), а режим работы с датой и временем по умолчанию — на `datetime32` (было `datetime64`).

Исправления ошибок

- Исправлена проблема, из-за которой команда `ydb import file csv` могла зависать.

Версия 2.23.0

Дата выхода 16 июля 2025. Для обновления до версии **2.23.0** перейдите в раздел Загрузки.

Функциональность

- Добавлена команда `ydb workload tpcc` для проведения нагрузочного тестирования TPC-C.
- Добавлена команда `ydb workload vector select` для тестирования производительности и полноты векторного индекса.
- Добавлена команда `ydb tools infer csv` для генерации SQL-запроса `CREATE TABLE` на основе CSV-файла с данными.

Улучшения

- Расширена обработка специальных значений (`null`, `/dev/null`, `stdout`, `cout`, `console`, `stderr`, `cerr`) для опции `--output` в командах `ydb workload * run`.
- Команды `ydb workload` теперь работают с абсолютными путями до объектов схемы в базе данных.
- Улучшения в интерактивном режиме выполнения запросов `ydb`:
  - Добавлена проверка соединения с сервером и описание горячих клавиш.
  - Улучшены inline-подсказки.
  - Добавлено автодополнение имён столбцов таблицы.
  - Добавлено кеширование схемы таблиц.

Исправления ошибок

- Исправлена ошибка, из-за которой команда `ydb tools restore` не работала на Windows.

Версия 2.22.1

Дата выхода 17 июня 2025. Для обновления до версии **2.22.1** перейдите в раздел Загрузки.

Исправления ошибок

- Исправлена ошибка, из-за которой сертификат не читался из файла, если путь к файлу указан в профиле в поле `ca-file`.
- Исправлена ошибка, из-за которой команды `ydb workload query import` и `ydb workload clickbench import files` в состоянии отображали количество строк вместо количества байт.

Версия 2.22.0

Дата выхода 4 июня 2025. Для обновления до версии **2.22.0** перейдите в раздел Загрузки.

Функциональность

- Добавлено автодополнение имён схемных объектов в интерактивном режиме.
- Расширены возможности команды `ydb workload query`: добавлены команды `ydb workload query init`, `ydb workload query import` и `ydb workload query clean` и изменена команда `ydb workload query run`. Пользуясь ими можно инициализировать таблицы, заполнить их данными, провести нагрузочное тестирование и очистить данные за собой.
- В команды `ydb workload clickbench run`, `ydb workload tpch run`, `ydb workload tpcds run` добавлена опция `--threads`, позволяющая указывать количество потоков для отправки запросов.
- ***(Требуется сервер v25.1+) (Экспериментально)*** Добавлена команда `ydb admin cluster config version` для отображения версии конфигурации (V1/V2) на узлах.

Изменения с потерей обратной совместимости

- Из команд `ydb workload * run` удалена опция `--executor`. Теперь всегда используется исполнитель `generic`.

Исправления ошибок

- Исправлена ошибка, из-за которой команды `ydb workload * clean` удаляли все содержимое целевой директории, а не только таблицы, созданные командой init.

Версия 2.21.0

Дата выхода 22 мая 2024. Для обновления до версии **2.21.0** перейдите в раздел Загрузки.

Функциональность

- Добавлен глобальный параметр `--no-discovery`, позволяющий пропустить процесс discovery и подключиться напрямую к указанному пользователем эндпоинту.
- Добавлены новые опции для команд нагрузочного тестирования:
  - Добавлена опция `--scale` в команды `ydb workload tpch init` и `ydb workload tpcds init` для установки процента размера данных и нагрузки относительно максимальной нагрузки.
  - Добавлена опция `--retries` в команды `ydb workload <clickbench|tpch|tpcds> run` для указания максимального количества повторов каждого запроса.
  - Добавлена опция `--partition-size` в команды `ydb workload <clickbench|tpcds|tpch> init` для установки максимального размера партиции в мегабайтах для строчных таблиц.
  - Добавлены параметры диапазона дат (`--date-to`, `--date-from`) в операции `ydb workload log run` для поддержки равномерного распределения первичных ключей.
- Улучшена функциональность резервного копирования и восстановления:
  - Добавлены опции `--replace` и `--verify-existence` в команду `ydb tools restore` для управления удалением существующих объектов, совпадающих с объектами в резервной копии, перед восстановлением.
  - Улучшена команда `ydb tools dump`: таблицы-реплики у ASYNC REPLICATION и их потоки изменений не сохраняются в локальные резервные копии. Это предотвращает дублирование потоков изменений и уменьшает размер резервной копии на диске.
- Изменения, повышающие удобство использования CLI:
  - Вывод подробной справки (`-hh`) теперь показывает всё дерево подкоманд.
  - Добавлена автоматическая вставка парных скобок в интерактивном режиме `ydb`.
  - Добавлена поддержка файлов с BOM (Byte Order Mark) в командах `ydb import file`.
- ***(Требуется сервер v25.1+) (Экспериментально)*** Улучшена команда `ydb debug latency`:
  - Добавлен параметр `--min-inflight` для установки минимального количества одновременных запросов (по умолчанию: 1).
  - Добавлена опция `--percentile` для указания пользовательских процентилей задержки.
  - Вывод команды расширен дополнительными измерениями GRPC ping.

Исправления ошибок

- Команда `ydb operation get` теперь корректно отображает операции, которые ещё находятся в процессе выполнения.
- Исправлены ошибки в команде `ydb scheme rmdir`:
  - Исправлена ошибка, из-за которой команда пыталась удалить поддомены.
  - Исправлен порядок удаления: внешние таблицы теперь удаляются перед источниками данных из-за возможных зависимостей между ними.
  - Добавлена поддержка coordination nodes при рекурсивном удалении.
- Исправлен код возврата команды `ydb workload * run --check-canonical` при несовпадении результатов с каноническими.
- Исправлена проблема, когда CLI пытался читать параметры из stdin даже при отсутствии данных.
- ***(Требуется сервер v25.1+) (Экспериментально)*** Исправлена ошибка авторизации при выполнении команды `ydb admin database restore` при наличии нескольких учетных записей администраторов базы данных в восстанавливаемой резервной копии.

Версия 2.20.0

Дата выхода 5 марта 2024. Для обновления до версии **2.20.0** перейдите в раздел Загрузки.

Функциональность

- Добавлена поддержка топиков при выполнении команд `ydb tools dump` и `ydb tools restore`.
- Добавлена поддержка узлов координации при выполнении команд `ydb tools dump` и `ydb tools restore`.
- Добавлена новая команда `ydb workload log import generator`.
- Добавлены новые глобальные опции для пользовательских сертификатов при соединении через SSL/TLS:
  - `--client-cert-file`: файл, содержащий пользовательский сертификат для SSL/TLS соединения, закодированный в PEM или PKCS#12.
  - `--client-cert-key-file`: файл, содержащий приватный ключ к пользовательскому сертификату, закодированный в PEM.
  - `--client-cert-key-password-file`: файл, содержащий пароль для приватного ключа пользовательского сертификата.
- Запросы при выполнении команды `ydb workload run` теперь отправляются на сервер в произвольном порядке.
- ***(Требуется сервер v25.1+)*** Добавлена поддержка внешних источников данных и внешних таблиц при выполнении команд `ydb tools dump` и `ydb tools restore`.
- ***(Экспериментально)*** Добавлена команда `ydb admin node config init` для инициализации директории с конфигурационными файлами узла.
- ***(Требуется сервер v25.1+) (Экспериментально)*** Добавлена команда `ydb admin cluster config generate` для генерации файла конфигурации V2 из файла конфигурации V1.
- ***(Требуется сервер v25.1+) (Экспериментально)*** Добавлены команда `ydb admin cluster dump` и команда `ydb admin cluster restore` для создания дампа кластера. Дамп кластера содержит список баз данных с метаданными, пользователей и группы, но не содержит схемные объекты.
- ***(Требуется сервер v25.1+) (Экспериментально)*** Добавлены команды `ydb admin database dump` и `ydb admin database restore` для создания дампа базы данных. Такой дамп содержит метаданные базы данных, схемные объекты, данные в них, пользователей и группы.
- ***(Требуется сервер v25.1+) (Экспериментально)*** Для команды `ydb admin cluster config fetch` добавлены новые опции `--dedicated-storage-section` и `--dedicated-cluster-section`, позволяющие получать части конфигурации для кластера и хранилища отдельно.

Исправления ошибок

- Исправлена ошибка, из-за которой дважды отправлялся запрос аутентификации в команде `ydb auth get-token` при получении списка эндпойнтов (Discovery запрос) и при фактическом выполнении запроса на получение токена.
- Исправлена ошибка в команде `ydb import file csv`, при которой прогресс импорта сохранялся даже если отправка пакета данных завершилась ошибкой.
- Исправлена ошибка, из-за которой при выполнении команды `ydb tools restore` некоторые ошибки игнорировались.
- Исправлена утечка памяти при генерации данных для `ydb workload tpcds`.

Версия 2.19.0

Дата выхода 5 февраля 2025. Для обновления до версии **2.19.0** перейдите в раздел Загрузки.

Функциональность

- Добавлена поддержка потоков изменений (changefeeds) при выполнении команд `ydb tools dump` и `ydb tools restore`.
- Добавлена рекомендация с текстом `CREATE TABLE` при схемной ошибке во время выполнения команды `ydb import file csv`.
- Добавлен вывод статистики для текущего процесса при выполнении команды `ydb workload`.
- Добавлен текст запроса к сообщению, если запрос завершился ошибкой при выполнении команды `ydb workload run`.
- Добавлено сообщение в случае ошибки истечения глобального таймаута при выполнении команды `ydb workload run`.
- ***(Требуется сервер v25.1+)*** Добавлена поддержка представлений (VIEW) при выполнении операций `ydb export s3` и `ydb import s3`. Представления экспортируются как YQL-выражение `CREATE VIEW`, которое выполняется при импорте.
- ***(Требуется сервер v25.1+)*** Добавлена опция `--skip-checksum-validation` для команды `ydb import s3`, позволяющая отключить валидацию контрольной суммы на стороне сервера.
- ***(Требуется сервер v25.1+) (Экспериментально)*** Для команды `ydb debug ping` добавлены новые опции: `--chain-length`, `--chain-work-duration`, `--no-tail-chain`.
- ***(Требуется сервер v25.1+) (Экспериментально)*** Для команды `ydb admin storage fetch` добавлены новые опции: `--dedicated-storage-section` и `--dedicated-cluster-section`.
- ***(Требуется сервер v25.1+) (Экспериментально)*** Для команды `ydb admin storage replace` добавлены новые опции: `--filename`, `--dedicated-cluster-yaml`, `--dedicated-storage-yaml`, `--enable-dedicated-storage-section` и `--disable-dedicated-storage-section`.

Исправления ошибок

- Исправлена ошибка, из-за которой команда `ydb update` в arm64-версии исполняемого файла YDB CLI скачивала и заменяла себя исполняемым файлом amd64-версии. Чтобы обновить ранее установленный YDB CLI до последней arm64-версии (а не amd64), его нужно переустановить.
- Команда `ydb workload run` теперь возвращает корректный код возврата.
- Исправлена ошибка, из-за которой команды `ydb workload tpch import generator` и `ydb workload tpcds import generator` завершались с ошибкой из-за отсутствия необходимых таблиц в схеме.
- Исправлена ошибка с обратными слешами при указании путей в [команде]](./reference/ydb-cli/commands/workload/index.md) `ydb workload` на Windows.

Версия 2.18.0

Дата выхода 24 декабря 2024. Для обновления до версии **2.18.0** перейдите в раздел Загрузки.

Функциональность

- Добавлена поддержка представлений (VIEW) при выполнении операций резервного копирования `ydb tools dump` и восстановления `ydb tools restore`. Представления сохраняются в файл "create_view.sql" в виде запросов `CREATE`

`VIEW`, которые будут выполнены для восстановления.

- В команду `ydb workload topic run` добавлены опции `--tx-commit-interval` и `--tx-commit-messages`, которые задают интервал между коммитами транзакций в миллисекундах и в количестве записанных сообщений соответственно.
- В команде `ydb topic read` параметр `--consumer` перестал быть обязательным. В режиме чтения без подписчика обязательно должны быть указаны идентификаторы партиций с помощью параметра `--partition-ids`. Чтение в этом случае выполняется без сохранения коммита оффсетов.
- Команда `ydb import file csv` теперь сохраняет прогресс выполнения. Повторный запуск команды импорта продолжится с той строки, на которой она была прервана.
- В командах `ydb workload kv` и `ydb workload stock` значение параметра `--executer` по умолчанию изменено на "generic", благодаря чему они больше не используют устаревшую инфраструктуру выполнения запросов.
- Изменен формат загрузки данных в таблицы для нагрузочных тестов `ydb workload` с CSV на Parquet.
- ***(Требуется сервер v25.1+) (Экспериментально)*** Добавлена команда `ydb admin storage` с подкомандами `fetch` и `replace` для управления конфигурацией хранилища сервера.

Изменения с потерей обратной совместимости

- В команде `ydb workload * run` параметр `--query-settings` заменен на `--query-prefix`.

Исправления ошибок

- Исправлена ошибка, из-за которой команда `ydb workload * run` в режиме `--dry-run` могла приводить к сбою.
- Исправлена ошибка в `ydb import file csv`, из-за которой несколько столбцов с экранированными кавычками в одной строке обрабатывались неправильно.

Версия 2.17.0

Дата выхода 4 декабря 2024. Для обновления до версии **2.17.0** перейдите в раздел Загрузки.

Функциональность

- ***(Требуется сервер v25.1+) (Экспериментально)*** Добавлена команда `ydb debug ping` для проверки производительности и связанности.

Производительность

- Улучшена производительность параллельного восстановления из файловой системы с помощью команды `ydb tools restore`.

Исправления ошибок

- Исправлена ошибка в схеме таблиц, созданных командой `ydb workload tpch`, из-за которой таблица `partsupp` содержала неверный список ключевых столбцов.
- Исправлена ошибка, из-за которой команда `ydb tools restore` завершалась с ошибкой `Too much data`, если было установлено максимальное значение параметра `--upload-batchbytes` (16MB).

Версия 2.16.0

Дата выхода 26 ноября 2024. Для обновления до версии **2.16.0** перейдите в раздел Загрузки.

Функциональность

- Увеличена пропускная способность команды `ydb import file csv` примерно в 3 раза.
- Добавлена поддержка stock-нагрузки для колоночных таблиц.
- Реализована поддержка временных меток в формате ISO 8601 для команд `ydb topic`.
- В команду `ydb sql` добавлена опция `--explain-ast`, которая выводит AST запроса.
- Добавлена подсветка синтаксиса ANSI SQL в интерактивном режиме.
- В команды `ydb workload tpch` и `ydb workload tpcds` добавлена поддержка синтаксиса PostgreSQL.
- В команду `ydb workload tpcds run` добавлена опция `-c` для сравнения результата с ожидаемым значением и отображения различий.
- В команды `ydb tools dump` и `ydb tools restore` добавлено логирование событий.
- В команду `ydb tools restore` добавлено указание места возникновения ошибки.

Изменения с потерей обратной совместимости

- В команде `ydb topic write` для опции `--codec` значение по умолчанию изменено на `RAW`.

Исправления ошибок

- Исправлен progress bar для команды `ydb workload import`.
- Устранена ошибка восстановления из резервной копии с использованием опции `--import-data`, возникавшая при изменении партиционирования таблицы.

Версия 2.10.0

Дата выхода 24 июня 2024. Для обновления до версии **2.10.0** перейдите в раздел Загрузки.

Функциональность

- Добавлена команда `ydb sql`, работающая поверх QueryService, позволяющая выполнять любые DML/DDL команды.
- Добавлен режим `notx` для опции `--tx-mode` в команде `ydb table query execute`.
- Добавлены времена начала и конца в описании длительных операций (export, import).
- Добавлена поддержка описания объектов типа replication в командах `ydb scheme describe` и `ydb scheme ls`.
- Добавлена поддержка типов big datetime: `Date32`, `Datetime64`, `Timestamp64`, `Interval64`.
- Переработана команда `ydb workload`:
  - Добавлена опция `--clear` в подкоманде `init`, позволяющая удалить все существующие таблицы перед созданием новых.

- Добавлена команда `ydb workload * import` для заполнения таблиц начальным контентом перед началом нагрузки.

Изменения с потерей обратной совместимости

- Переработана команда `ydb workload`:
  - Опция `--path` перемещена на уровень конкретного типа нагрузки. Например: `ydb workload tpch --path some/tables/path init ...`.
  - Значение опции `--store=s3` переименовано в `--store=external-s3` в подкоманде `init`.

Исправления ошибок

- Исправлена работа с цветовыми схемами в формате `PrettyTable`

Версия 2.9.0

Дата выхода 25 апреля 2024. Для обновления до версии **2.9.0** перейдите в раздел Загрузки.

Функциональность

- Улучшены таблицы с логическими планами запросов: стали информативнее, добавлены цвета, исправлены некоторые ошибки.
- Для команды `ydb workload` поддержана опция `-v`, включающая вывод отладочной информации.
- Добавлена возможность запустить `ydb workload tpch --store s3` с источником s3 для измерения производительности федеративных запросов.
- Добавлена опция `--rate` для команды `ydb workload` для ограничения количества транзакций (запросов) в секунду.
- Добавлена опция `--use-virtual-addressing` для импорта/экспорта s3, позволяющая переключить режим virtual hosting of buckets для схемы путей s3.
- Улучшена производительность команды `ydb scheme ls` параллельным запуском листингов директорий.

Исправления ошибок

- Исправлено обрезание лишних символов при переносе строк в таблицах CLI
- Исправлена ошибка обращения к памяти в команде `tools restore`.
- Исправлено игнорирование опции `--timeout` в generic и scan запросах.
- Добавлен таймаут 60s на проверку версии и скачивание бинарного файла CLI для избежания бесконечного ожидания.
- Исправлен ряд незначительных ошибок: опечатки, обработка пустых файлов и т.д.

Версия 2.8.0

Дата выхода 12 января 2024. Для обновления до версии **2.8.0** перейдите в раздел Загрузки.

Функциональность

- Добавлены команды управления конфигурациями кластера ydb admin config и ydb admin volatile-config.
- Добавлена поддержка загрузки PostgreSQL-совместимых типов командой ydb import file csv|tsv|json. Только для строковых таблиц.
- Добавлена поддержка загрузки директории из S3-совместимого хранилища в команде ydb import s3. Пока доступна только под Linux и Mac OS.
- Добавлена поддержка вывода результата выполнения команд ydb table query execute, ydb yql и ydb scripting yql в формате Apache Parquet.
- В командах ydb workload добавлена опция `--executer`, задающая используемый тип запросов.
- Добавлена колонка медианного времени выполнения бенчмарка в таблице статистики в команде ydb workload clickbench.
- ***(Experimental)*** Добавлен тип запросов `generic` в команде ydb table query execute, позволяющий выполнять DDL и DML операции, с результатами произвольного размера и с поддержкой MVCC. Команда использует экспериментальное API, совместимость не гарантируется.
- ***(Experimental)*** В команде `ydb table query explain` добавлена опция `--collect-diagnostics` для сбора диагностики запроса и сохранения её в файл. Команда использует экспериментальное API, совместимость не гарантируется.

Исправления ошибок

- Исправлена ошибка вывода таблиц в `pretty` формате с Unicode символами.
- Исправлена ошибка подстановки неправильного первичного ключа в команде `ydb tools pg-convert`.

Версия 2.7.0

Дата выхода 23 октября 2023. Для обновления до версии **2.7.0** перейдите в раздел Загрузки.

Функциональность

- Добавлена команда `ydb tools pg-convert`, выполняющая подготовку дампа, полученного утилитой pg_dump, к загрузке в postgres-совместимую прослойку YDB.
- Добавлена команда нагрузочного тестирования `ydb workload query`, которая нагружает базу запросами выполнения скрипта в несколько потоков.
- Добавлена команда для просмотра списка разрешений `ydb scheme permissions list`.
- В командах ydb table query execute, ydb table query explain, ydb yql и ydb scripting yql добавлена опция `--flame-graph`, задающая путь до файла, в котором необходимо сохранить визуализацию статистики выполнения запросов.
- Специальные команды интерактивного режима выполнения запросов теперь не чувствительны к регистру.
- Добавлена валидация специальных команд и их параметров.
- Добавлено чтение из таблицы в сценарии с транзакциями в команде ydb workload transfer topic-to-table run.
- Добавлена опция `--commit-messages` в команде ydb workload transfer topic-to-table run, задающая число сообщений в одной транзакции.
- Добавлены опции `--only-table-in-tx` и `--only-topic-in-tx` в команде ydb workload transfer topic-to-table run, задающие ограничения на виды запросов в одной транзакции.

- Добавлены новые колонки `Select time` и `Upsert time` в таблице статистики в команде ydb workload transfer topic-to-table run.

Исправления ошибок

- Исправлена ошибка при загрузке пустого JSON списка командами ydb table query execute, ydb scripting yql и ydb yql.

Версия 2.6.0

Дата выхода 7 сентября 2023. Для обновления до версии **2.6.0** перейдите в раздел Загрузки.

Функциональность

- В ydb workload tpch run добавлена опция `--path`, содержащая путь до директории с таблицами, созданными командой ydb workload tpch init.
- Добавлена команда ydb workload transfer topic-to-table, которая нагружает базу запросами на чтение из топиков и запись в таблицу.
- Добавлена опция `--consumer-prefix` в командах ydb workload topic init, ydb workload topic run read|full, задающая префиксы имен читателей.
- Добавлена опция `--partition-ids` в команде ydb topic read, задающая список id партиций топика для чтения, разделенных запятой.
- Добавлена поддержка форматов параметров CSV и TSV в командах исполнения YQL запросов.
- Переработан интерактивный режим выполнения запросов. Добавлены новые специфичные команды интерактивного режима: `SET stats`, `EXPLAIN`, `EXPLAIN AST`. Добавлены сохранение истории между запусками CLI и автодополнение YQL запросов.
- Добавлена команда ydb config info, которая выводит текущие параметры соединения без подключения к базе данных.
- Добавлена команда ydb workload kv run mixed, которая нагружает базу запросами на запись и чтение.
- Опция `--percentile` в командах ydb workload topic run write|read|full теперь может принимать вещественные значения.
- Увеличены значения по умолчанию для опций `--seconds` и `--warmup` в командах ydb workload topic run write|read|full до 60 секунд и 5 секунд соответственно.
- Изменено значение по умолчанию для опции `--supported-codecs` на `RAW` в командах ydb topic create и ydb topic consumer add.

Исправления ошибок

- Исправлена потеря строк при загрузке командой ydb import file json.
- Исправлен неучет статистики во время прогрева команд ydb workload topic run write|read|full.
- Исправлен неполный вывод статистики в командах ydb scripting yql и ydb yql.
- Исправлен неправильный вывод progress bar'а в командах ydb tools dump и ydb tools restore.
- Исправлена ошибка загрузки больших файлов с заголовком в команде ydb import file csv|tsv.
- Исправлено зависание команды ydb tools restore --import-data.
- Исправлена ошибка `Unknown value Rejected` при выполнении команды ydb operation list buildindex.

Версия 2.5.0

Дата выхода 20 июня 2023. Для обновления до версии **2.5.0** перейдите в раздел Загрузки.

Функциональность

- Для команды `ydb import file` добавлен параметр --timeout, задающий время, в течение которого должна быть выполнена операция на сервере.
- Добавлен индикатор прогресса в командах ydb scheme rmdir --recursive и ydb import file.
- Добавлена команда ydb workload kv run read-rows, которая нагружает базу запросами на чтение строк, используя новый экспериментальный API вызов ReadRows (реализован только в ветке main), выполняющий более быстрое чтение по ключу, чем select.
- В ydb workload topic добавлены новые параметры `--warmup-time`, `--percentile`, `--topic`, задающие время прогрева теста, процентиль в выводе статистики и имя топика соответственно.
- Добавлена команда ydb workload tpch для запуска нагрузочного теста TPC-H.
- Добавлен флаг `--ordered` в команде ydb tools dump, сохраняющий порядок по первичному ключу в таблицах.

Производительность

- Увеличена скорость загрузки данных в команде `ydb import file` за счет добавления параллельной загрузки. Число потоков задается новым параметром --threads.
- Увеличена производительность команды ydb import file json, за счет уменьшения числа копирований данных.

Версия 2.4.0

Дата выхода 24 мая 2023. Для обновления до версии **2.4.0** перейдите в раздел Загрузки.

Функциональность

- Добавлена возможность загрузки нескольких файлов командой ydb import file.
- Добавлена поддержка удаления колоночных таблиц для команды ydb scheme rmdir --recursive.
- Повышена стабильность работы команды ydb workload topic.

Версия 2.3.0

Дата выхода 1 мая 2023. Для обновления до версии **2.3.0** перейдите в раздел Загрузки.

Функциональность

- Добавлен интерактивный режим выполнения запросов. Для перехода в интерактивный режим выполните команду ydb yql без аргументов. Режим экспериментальный, обратная совместимость пока не гарантируется.
- Добавлена команда ydb index rename для атомарной замены или переименования вторичного индекса.
- Добавлена команда `ydb workload topic` для запуска нагрузки, которая читает и записывает сообщения в топики.

- Для команды `ydb scheme rmdir` добавлен параметр --recursive, который позволяет рекурсивно удалить директорию вместе со всем содержимым.
- Для команды ydb scheme describe добавлена поддержка типов `topic` и `coordination node`.
- Для команды `ydb topic consumer` добавлен параметр --commit для подтверждения прочитанных сообщений.
- Для команды `ydb import file csv|tsv` добавлен параметр --columns, с помощью которого можно указать список колонок вместо заголовка в файле.
- Для команды `ydb import file csv|tsv` добавлен параметр --newline-delimited, который подтверждает отсутствие символа переноса строки в данных. Использование этого параметра ускоряет импорт за счет параллельного чтения из нескольких секций файла.

Исправления ошибок

- Исправлена ошибка, которая приводила к повышенному потреблению памяти и процессора при выполнении команды `ydb import file`.

Версия 2.2.0

Дата выхода 3 марта 2023. Для обновления до версии **2.2.0** перейдите в раздел Загрузки.

Функциональность

- Исправлена ошибка, когда невозможно было указать поддерживаемые алгоритмы сжатия при добавлении читателя топика.
- Добавлена поддержка потокового выполнения YQL-скриптов и запросов на основе параметров, передаваемых через `stdin`.
- Значения параметров YQL-запросов теперь могут быть переданы из файла.
- Запрос на ввод пароля теперь выводится в `stderr` вместо `stdout`.
- Путь к корневому CA сертификату теперь может быть сохранен в профиле.
- Добавлен глобальный параметр --profile-file для использования указанного файла в качестве хранилища для настроек профилей.
- Добавлен новый тип нагрузочного тестирования ydb workload clickbench.

Версия 2.1.1

Дата выхода 30 декабря 2022. Для обновления до версии **2.1.1** перейдите в раздел Загрузки.

Улучшения

- Добавлена поддержка параметра `--stats` команды ydb scheme describe для колоночных таблиц.
- Добавлена поддержка файлов в формате Parquet для импорта командой ydb import.
- Поддержаны дополнительное логирование и ретраи для команды ydb import.

Версия 2.1.0

Дата выхода 18 ноября 2022. Для обновления до версии **2.1.0** перейдите в раздел Загрузки.

Функциональность

- Добавлена возможность неинтерактивного создания профиля.
- Добавлены команды ydb config profile update и ydb config profile replace для изменения и замены профилей.
- Для команды ydb scheme ls добавлен параметр `-1`, включающая режим вывода по одному объекту на строку.
- URL сервиса IAM теперь можно сохранять в профиле.
- Добавлена возможность использовать аутентификацию по логину и паролю без указания пароля.
- Добавлена поддержка профилей AWS в команде ydb export s3.
- Добавлена возможность создания профиля используя `stdin`. Например, можно передать вывод команды YC CLI `yc ydb database get information` на вход команде `ydb config profile create`.

Исправления ошибок

- Исправлена ошибка, когда некорректно выводился результат запроса в формате JSON-array, если он состоял из нескольких ответов сервера.
- Исправлена ошибка, приводящая к невозможности изменить профиль, используя при этом некорректный профиль.

Версия 2.0.0

Дата выхода 20 сентября 2022. Для обновления до версии **2.0.0** перейдите в раздел Загрузки.

Функциональность

- Добавлена возможность работы с топиками:
  - `ydb topic create` — создание топика;
  - `ydb topic alter` — изменение топика;
  - `ydb topic write` — запись данных в топик;
  - `ydb topic read` — чтение данных из топика;
  - `ydb topic drop` — удаление топика.
- Добавлен новый тип нагрузочного тестирования:
  - `ydb workload kv init` — создание таблицы для тестирования kv нагрузки;
  - `ydb workload kv run` — запуск одной из 3 видов нагрузки: запуск нескольких сессий вставки `UPSERT`, запуск нескольких сессий вставки `INSERT` или запуск нескольких сессий с GET-запросами по первичному ключу;
  - `ydb workload kv clean` — удаление тестовой таблицы.
- Добавлена возможность деактивировать текущий активный профиль (см. команду `ydb config profile deactivate`).
- Добавлена возможность неинтерактивного удаления профиля без подтверждения (см. параметр `--force` команды `ydb config profile remove`).
- Добавлена поддержка CDC для команды `ydb scheme describe`.

- Добавлена возможность просмотра текущего статуса БД (см. команду `ydb monitoring healthcheck`).
- Добавлена возможность просмотра аутентификационной информации (токена), с которой будут отправляться запросы к БД при текущих настройках аутентификации (см. команду `ydb auth get-token`).
- Добавлена возможность чтения данных из стандартного потока ввода для команды `ydb import`.
- Добавлена возможность импорта данных в формате JSON из файла или стандартного потока ввода (см. команду `ydb import file json`).

Улучшения

- Улучшен процессинг команд. Парсинг и валидация пользовательского ввода теперь более точные.

Версия 1.9.1

Дата выхода 25 июня 2022. Для обновления до версии **1.9.1** перейдите в раздел Загрузки.

Функциональность

- Добавлена возможность сжатия данных при экспорте в S3-совместимое хранилище (см. параметр `--compression` команды ydb export s3).
- Добавлена возможность управления автоматической проверкой доступности новой версии YDB CLI (см. параметры `--disable-checks` и `--enable-checks` команды ydb version).

Список изменений безопасности

Исправлено в YDB 22.4.44, 2022-11-28

CVE-2022-28228

В сервере YDB обнаружено чтение за пределами допустимого адресного пространства. Злоумышленник с помощью специально сконструированного запроса с оператором insert может получить доступ к конфиденциальной информации или вызвать сбой.

Ссылка на CVE: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-28228.

Обнаружено благодаря Максиму Арнольду.

Исправлено в YDB Go SDK v3.53.3, 2023-10-17

CVE-2023-45825

Токен авторизации может утекать через логи

Ссылка на CVE: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2023-45825.

Обнаружено благодаря Сергею Фостер.

Исправлено в YDB 25.3.1.25, 2026-04-03

CVE-2026-10549

LDAP-инъекция в фильтре позволяет злоумышленнику с действительными LDAP-учётными данными обойти проверку членства в группе и получить несанкционированный доступ к базе данных.

Ссылка на CVE: https://www.cve.org/CVERecord?id=CVE-2026-10549.