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 документацией для DevOps-инженеров 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

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.

> **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.

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
```

5. Примените манифест для создания кластера YDB:

```
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. Созданная база данных будет доступна изнутри кластера Kubernetes по DNS-имени `database-minikube-sample` на портах 2135 и 8765.
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
```

6. Примените манифест для создания хранилища данных YDB:

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

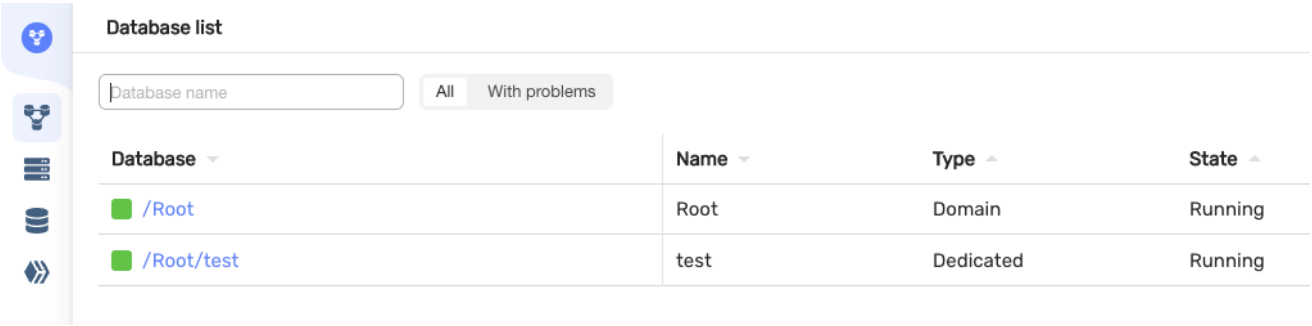
7. Подождите, пока `kubectl get storages.ydb.tech` станет `Ready`.
8. Примените манифест для создания базы данных:

```
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. Созданная база данных будет доступна изнутри кластера Kubernetes по DNS-имени `database-kind-sample` на портах 2135 и 8765.
11. Получите доступ к порту 8765 снаружи Kubernetes через `kubectl port-forward database-kind-sample-0 8765` для продолжения.

Запустите первый запрос "Hello, world!"

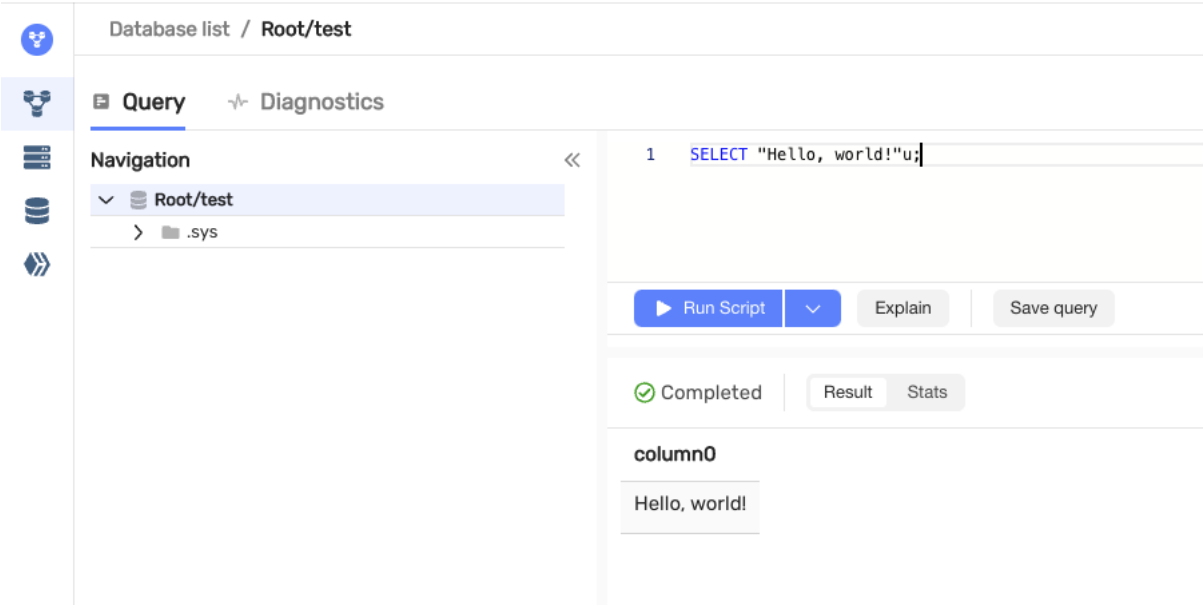
Самый простой способ выполнить свой первый запрос к YDB - это встроенный веб-интерфейс. Он запускается по умолчанию на порту 8765 сервера YDB, поэтому, если вы запустили его локально, вам нужно открыть localhost:8765 в вашем веб-браузере. Если нет, замените `localhost` на имя хоста вашего сервера в этом 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, а `/Root/test` или `/local` (имя зависит от выбранного способа установки) - это «игровая площадка», созданная в процессе установки на предыдущем шаге. Давайте нажмём на последнюю, `/Root/test` или `/local`, затем введем наш первый запрос и нажмем 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 кластерYDB, 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.

Обзор YDB

YDB — это горизонтально масштабируемая распределенная отказоустойчивая СУБД. YDB спроектирована с учетом требований высокой производительности — например, обычный сервер может обрабатывать десятки тысяч запросов в секунду. В дизайн системы заложена работа с объемами данных в сотни петабайт. YDB может эксплуатироваться в однодатацентровом и геораспределенном (кросс-датацентровый) режимах на кластере из тысяч серверов.

YDB обеспечивает:

- строгую консистентность с возможностью ослабления для увеличения производительности;
- поддержку запросов YQL (диалект SQL для работы с большими данными);
- автоматическую репликацию данных;
- высокую доступность с автоматической обработкой отказов вычислительных узлов, стоек, или зон доступности;
- автоматическое партицирование данных при увеличении их объема или увеличении нагрузки.

Для взаимодействия с YDB доступен YDB CLI, а также SDK для C++, C#, Go, Java, Node.js, PHP, Python и Rust.

YDB поддерживает реляционную модель данных и оперирует строковыми и колоночными таблицами с предопределенной схемой. Для удобства организации таблиц поддерживается создание директорий по аналогии с файловой системой. Помимо таблиц YDB поддерживает топики, которые представляют собой сущность для хранения неструктурированных сообщений и доставки их множеству подписчиков.

В качестве основного способа формирования команд к базе данных используется язык YQL, являющийся диалектом SQL. Таким образом, пользователю предлагается мощный и, в то же время, привычный способ взаимодействия с БД.

В YDB поддерживаются высокопроизводительные распределенные ACID-транзакции, которые могут затрагивать несколько записей из разных таблиц одного типа. Обеспечивается самый строгий уровень изоляции транзакций — serializable. Также имеется возможность ослабления уровня изоляции для увеличения производительности.

В дизайн YDB заложена поддержка разных сценариев нагрузки, таких как OLTP и OLAP. В текущей реализации поддержка аналитических запросов ограничена. Поэтому можно говорить, что в данный момент YDB — это OLTP-база данных.

YDB — открытая система. Исходный код YDB доступен под лицензией Apache License 2.0. Взаимодействие клиентских приложений с YDB построено на базе протокола gRPC, спецификация которого открыта, что позволяет реализовать SDK для любого языка программирования.

Сценарии применения

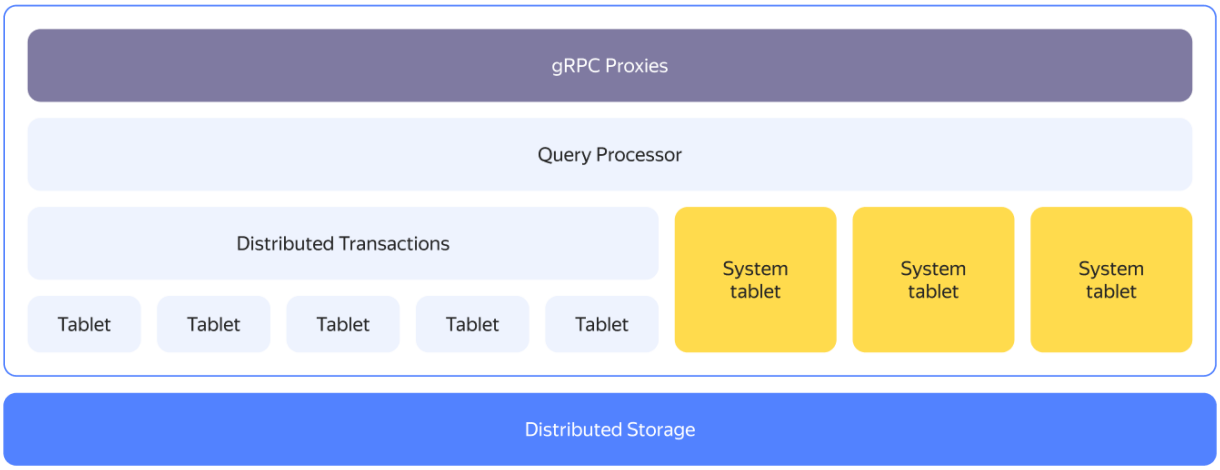
YDB является альтернативой имеющимся решениям в следующих случаях:

- При использовании NoSQL-систем, когда требуется строгая консистентность данных.
- При использовании NoSQL-систем, когда требуется транзакционное изменение данных, хранящихся в разных строках одной или нескольких таблиц.
- В системах, требующих обработки и хранения большого объема данных, с возможностью практически неограниченного горизонтального масштабирования (в эксплуатации находятся промышленные кластеры более чем из 5000 узлов, обрабатывающие нагрузку в миллионы RPS и хранящие петабайты данных).
- В системах с незначительной нагрузкой, когда поддержка отдельного инстанса базы данных будет расточительна с материальной точки зрения (предлагается использовать YDB в режиме бессерверных вычислений).
- В системах с плохо предсказуемой или сезонно меняющейся нагрузкой (используя возможность добавления/уменьшения вычислительных ресурсов по запросу и/или в режиме бессерверных вычислений).
- В высоконагруженных системах, которые шардируют нагрузку между инстансами реляционной базы данных.
- При разработке нового продукта, для которого нет надежного прогноза будущей нагрузки, или ожидается большая нагрузка, превышающая возможности традиционных реляционных баз данных.
- В проектах, где требуются одновременная работа с транзакционными и аналитическими нагрузками.

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.

Архитектура YDB

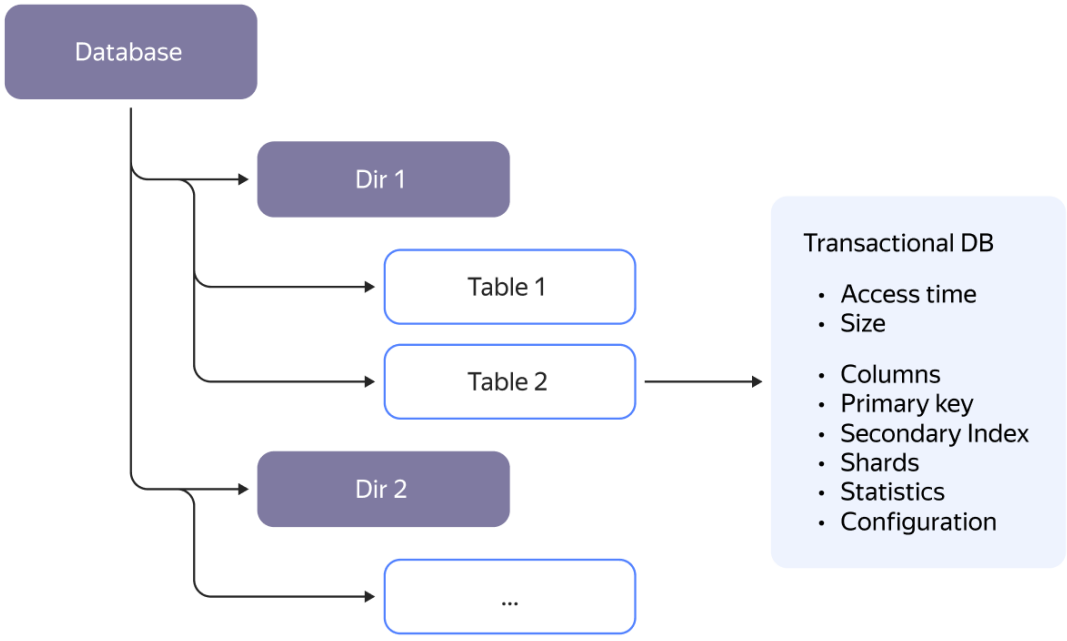


Кластеры YDB обычно работают с shared nothing архитектурой на обычном оборудовании. Уровни вычислений и хранения являются разнёсенными. Они могут работать как на отдельных наборах узлов, так и быть совмещёнными.

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

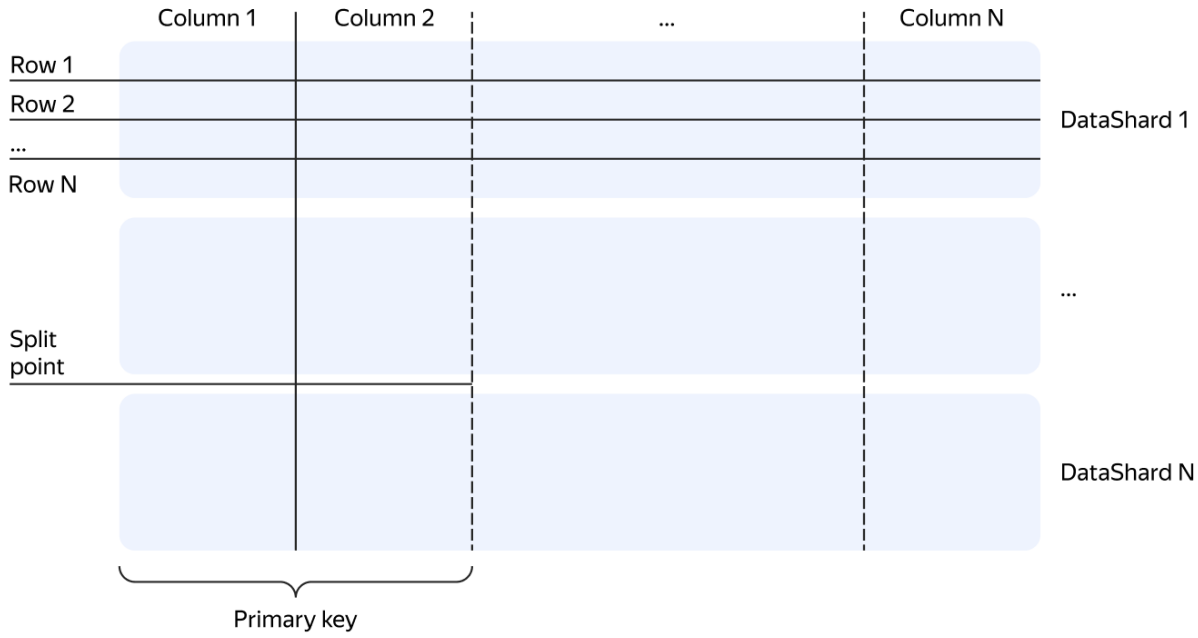
Более подробная общая архитектура YDB объясняется в разделе Общая схема YDB.

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



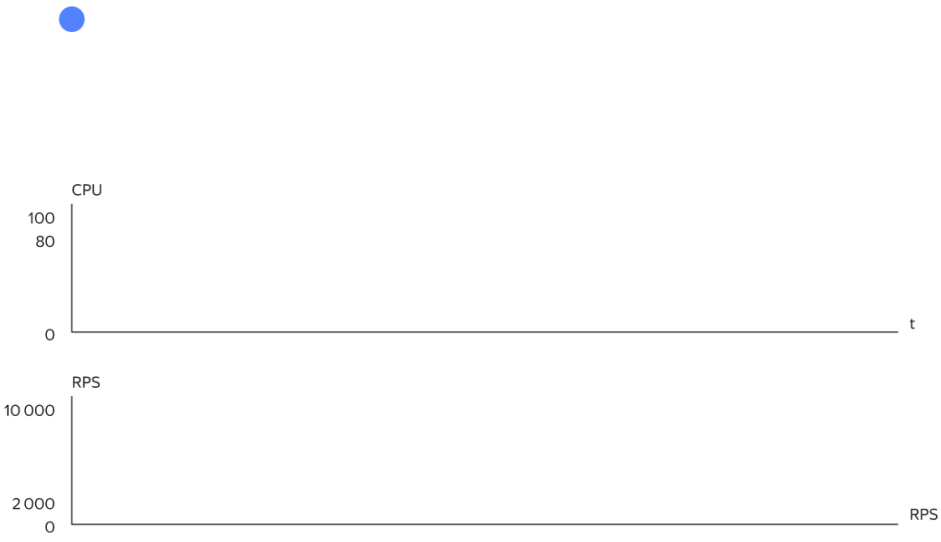
YDB предоставляет пользователям хорошо известную абстракцию — таблицы. В YDB существует два основных типа таблиц:

- 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

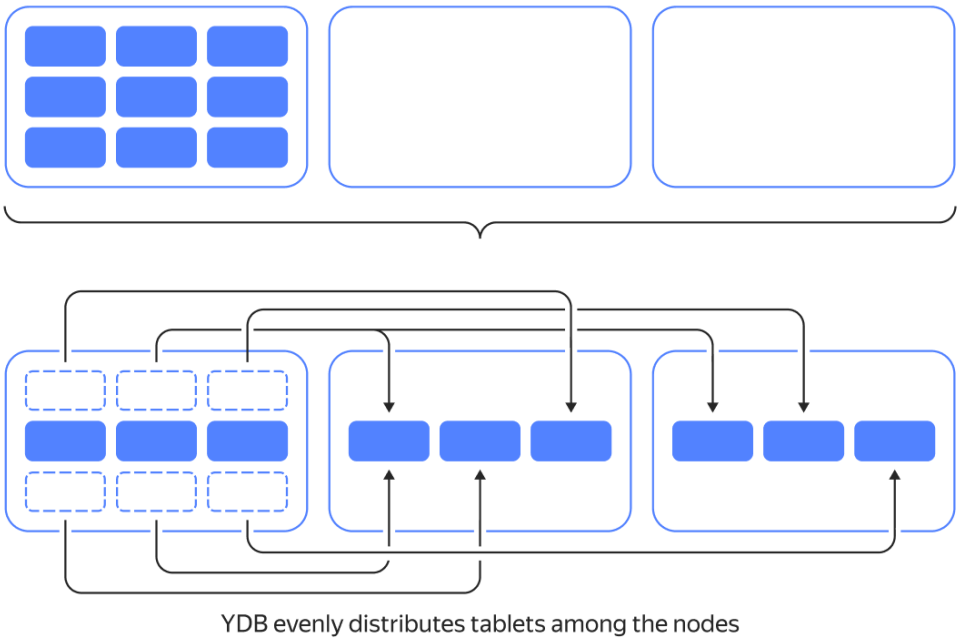


Data shard автоматически разделяются на большее количество при увеличении нагрузки. Они автоматически объединяются в нужное количество, когда пиковая нагрузка уходит.

Разделение по размеру

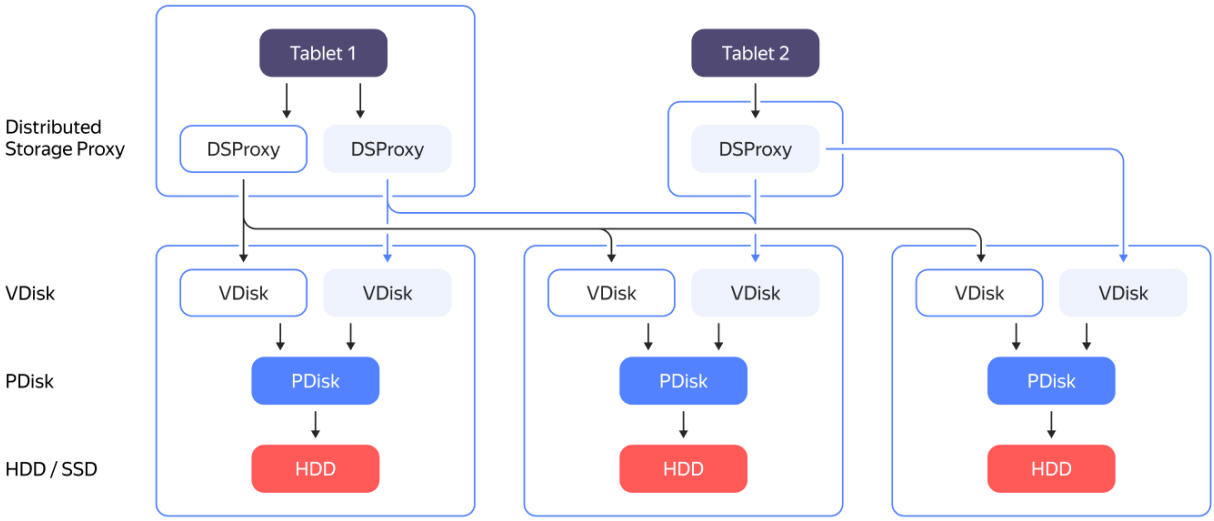
Data shard также автоматически разделяются при увеличении размера данных. Они автоматически сливаются обратно, если достаточное количество данных будет удалено.

Автоматическая балансировка



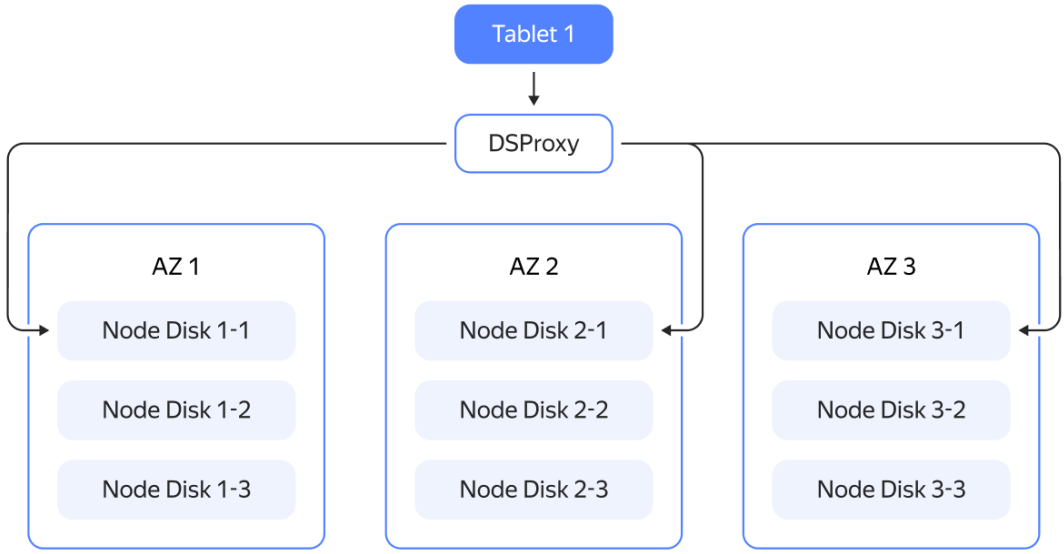
YDB равномерно распределяет таблетки среди доступных узлов. Она перемещает тяжело загруженные таблетки с перегруженных узлов. Метрики CPU, памяти и сети отслеживаются для облегчения этого процесса.

Внутреннее устройство распределенного хранилища



YDB не полагается на сторонние файловые системы. Она хранит данные, работая непосредственно с дисковыми накопителями как блочными устройствами. Поддерживаются все основные типы дисков: NVMe, SSD или HDD. За работу с конкретным блочным устройством отвечает компонент PDisk. Уровень абстракции выше PDisk называется VDisk. Также есть специальный компонент, называемый DSProxy, между таблеткой и VDisk. DSProxy анализирует доступность и характеристики дисков и выбирает, какие диски будут обрабатывать запрос, а какие нет.

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.

YDB для DevOps-инженеров

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 для DevOps-инженеров — дополнения к общему разделу концепций, актуальные для DevOps-инженеров.
- 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.

YDB для разработчиков приложений

В этом разделе документации YDB описано всё, что нужно знать для разработки приложений, взаимодействующих с YDB.

Основные материалы:

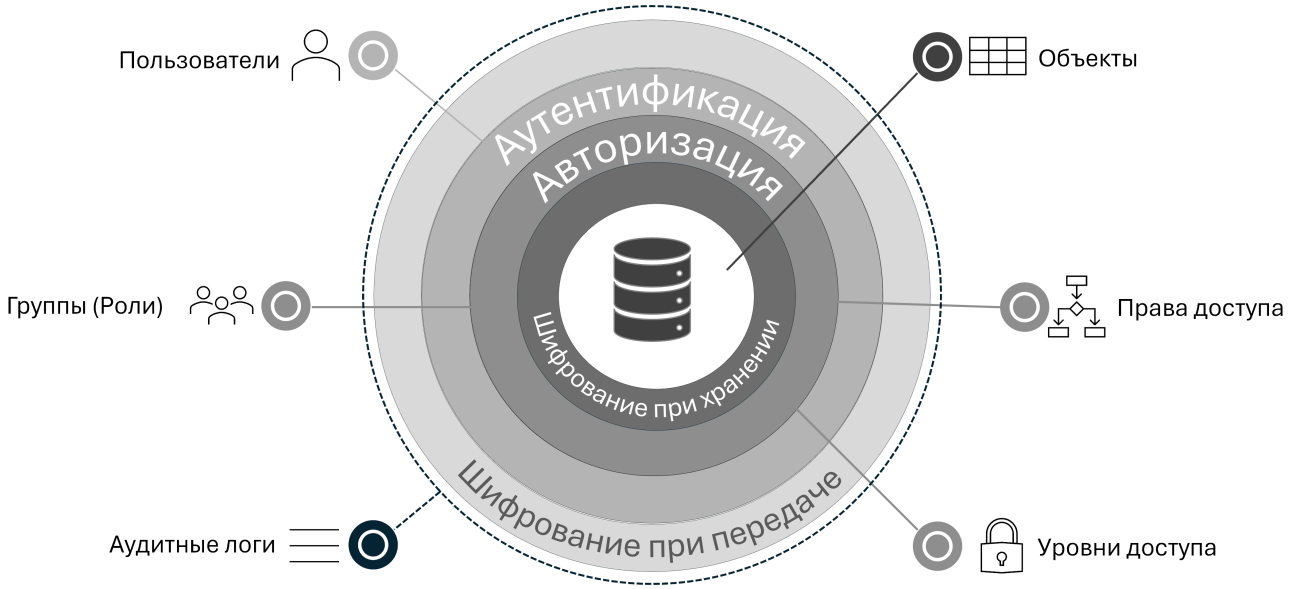
- Начало работы с YDB в роли разработчика приложений
- Тестовое приложение
- Туториал по YQL - Обзор
- Выбор первичного ключа для:
  - Строчных таблиц
  - Колоночных таблиц
- Вторичные индексы
- Загрузка данных в YDB
- Постраничный вывод
- Использование таймаутов
- Системные представления базы данных
- Change Data Capture
- Пользовательские атрибуты таблиц
- Workload Manager — управление потреблением ресурсов
- Справка:
  - YQL - Обзор
  - Справка по YDB SDK
  - YDB CLI
  - Совместимость YDB с PostgreSQL
  - Kafka API

Если же вам интересная разработка ядра YDB или сопутствующих проектов, обратитесь к документации для контрибьюторов.

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 три уровня доступа:
  - **наблюдатель** даёт возможность просмотра состояния системы, закрытого от публичного доступа;
  - **оператор** даёт дополнительные возможности просмотра и выполнения действий, изменяющих состояние системы;
  - **администратор** даёт право на выполнение административных действий с базами или кластером.

  Как и права доступа, один или несколько уровней доступа могут быть выданы конкретному пользователю или группе. Субъект доступа, не имеющий ни одного уровня доступа, может просматривать публично доступную информацию в системе. Уровни доступа последовательно наращивают возможности субъекта доступа. Чтобы получить максимальные возможности, субъект доступа должен иметь все три уровня доступа.

  Подробнее об уровнях доступа см. в разделе Настройки административных и других привилегий.
- **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**. 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 Секция конфигурации `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.

Разработка YDB

Данный раздел содержит материалы для разработчиков и контрибьюторов YDB.

- Сборка и тестирование с использованием Ya Make
- Работа над изменениями
- Управление релизами

Справка по YDB

В этом разделе документации содержится справочная информация по различным аспектам YDB, включая:

- YQL - Обзор
- Совместимость YDB с PostgreSQL
- Использование встроенного web-интерфейса
- Интеграции YDB
- YDB CLI
- Справка по YDB SDK
- Языки и API
- Kafka API
- Параметры конфигурации кластера
- Справка по наблюдаемости YDB
- Обзор утилиты YDB DSTool
- Обзор утилиты ydbops
- Справка по работе YDB в Docker

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 на YDB SDK
- YDB CLI рецепты
- YQL рецепты
- Для аналитиков

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

Вопросы и ответы про YDB

- Общие
- SDK
- Ошибки
- YQL
- Аналитика

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

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 инженеров DevOps.

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.

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.

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.

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).

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.

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 выделяется специальный вид **вторичного индекса** - векторный 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 выделяется отдельно от вторичного индекса, так как решает иные задачи.

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.

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 предназначены для отслеживания состояния базы данных. Эти таблицы доступны из корня дерева базы данных и используют системный префикс пути `.sys`. Они описаны более подробно в Системные представления базы данных.

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.

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.

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.

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.

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 схемы кластера 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 три уровня доступа:

- наблюдатель;
- оператор;
- администратор.

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.

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.

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.

Подписчик таблетки

Подписчик 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 подписчиков. Подписчики выполняют две основные функции:

- В случае завершения или отказа лидера подписчики являются предпочтительными кандидатами на роль нового лидера, так как они могут стать лидером намного быстрее, чем другие кандидаты, поскольку они применили большую часть журнала.
- Подписчики могут отвечать на запросы только для чтения, если клиент явно выбирает опциональный расслабленных 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.

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.

NodeBroker (#node-broker)

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.
- Подписчики 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.

Компакшн

Компакшн, **комактизация** 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 отличии 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 распределённого хранилища. 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).

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, KQP 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.

KQP

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.

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.

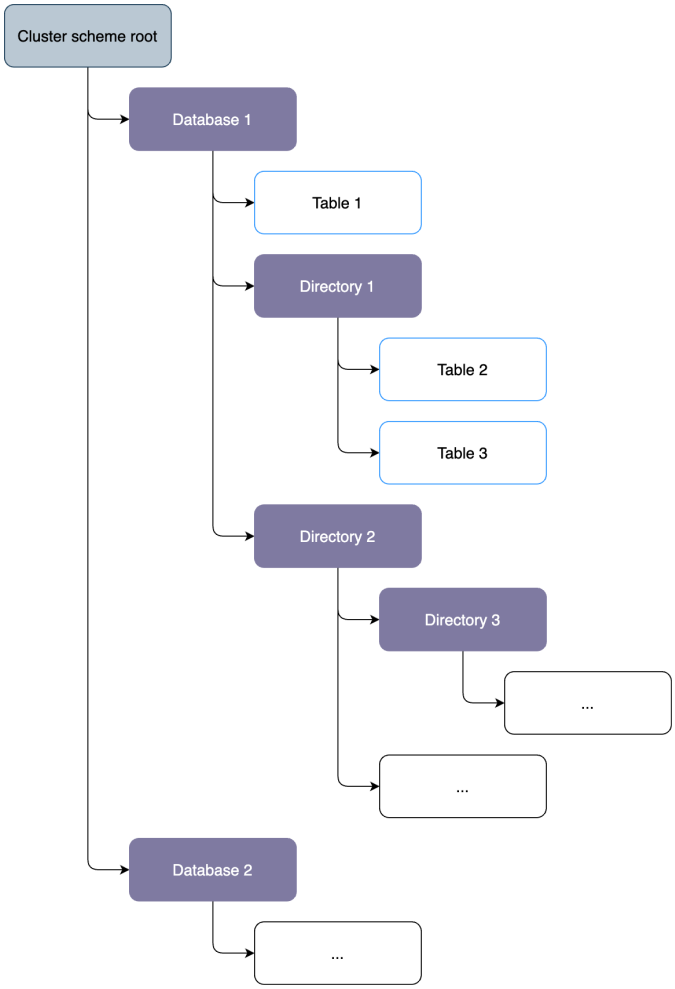
В состав операционных систем, на которых запускается клиент, уже входит набор корневых сертификатов от основных мировых центров сертификации. Однако, владелец кластера YDB может использовать свой центр сертификации, не связанный ни с одним из мировых, что часто встречается в корпоративных окружениях, 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).

Структура кластера

В разделе собраны описания сущностей схемы кластера, которыми оперирует YDB.

Схема кластера YDB

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 с помощью вложенных директорий.



Схемные объекты YDB

В YDB поддерживается следующий перечень схемных объектов:

- Директории
- Таблицы
- Представления (VIEW)
- Топики
- Секреты
- Подключения к внешним БД
- Внешние источники данных

Все схемные объекты в YDB имеют одинаковые требования к своим наименованиям.

Правила наименования схемных объектов

У каждого схемного объекта базы данных в YDB есть имя. В YQL-выражениях имена схемных объектов указываются с помощью идентификаторов, заключённых в обратные кавычки (`) или без этих символов. Для более подробной информации об идентификаторах, см. Ключевые слова и идентификаторы.

Имена схемных объектов в YDB должны соответствовать следующим требованиям:

- Имя объекта может состоять из следующих символов:
  - прописные латинские буквы;
  - строчные латинские буквы;
  - цифры;
  - специальные символы: `.`, `-` и `_`.

- Длина имени объекта не должна превышать 255 символов.
- Объекты не должны создаваться в папках, имена которых начинаются с точки, таких как `.sys`, `.medatata`, `.sys_health`.

Правила наименования колонок

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_`.

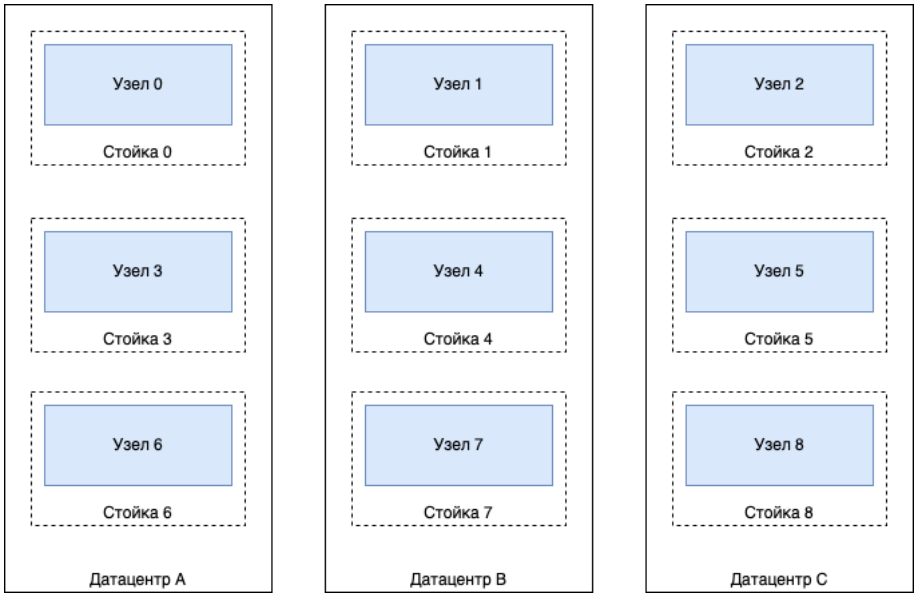
Топология кластера YDB

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 одним из ключевых решений при планировании нового развёртывания YDB.

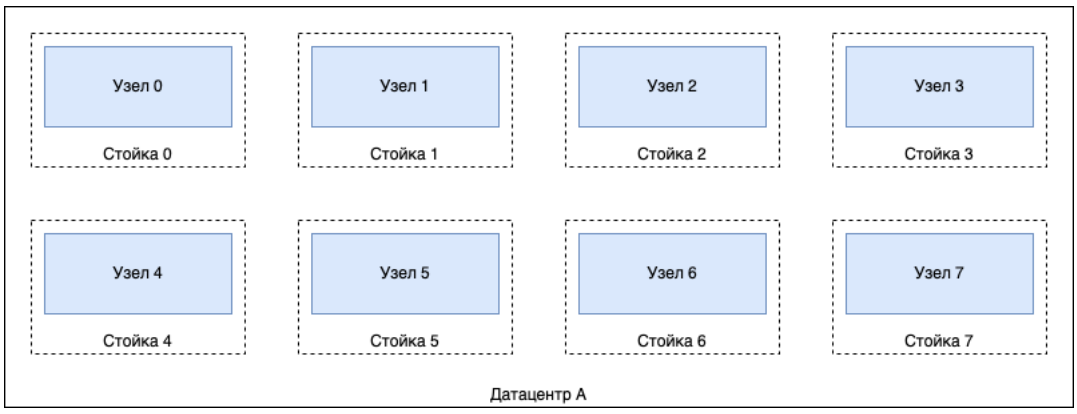
Режимы работы кластера

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 нескольких 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)	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)	Стойка	Дата-центр	1	8
`block-4-2` *(упрощённый)*, переживает отказ 1 стойки	1.5	10	½ стойки	Дата-центр	1	5
`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.

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).

Минимальная отказоустойчивая конфигурация кластера YDB представляет собой 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.

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 для DevOps-инженеров
- Конфигурация Blob Storage
- Примеры конфигурационных файлов кластера
- Дисковая подсистема кластера aka YDB distributed storage

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

По умолчанию транзакции в YDB выполняются в режиме *Serializable*, который предоставляет самый строгий уровень изоляции для пользовательских транзакций. В этом режиме гарантируется, что результат успешно выполненных параллельных транзакций эквивалентен определенному последовательному порядку их выполнения, при этом для успешных транзакций отсутствуют аномалии чтений.

В случае, когда требования консистентности или свежести читаемых транзакцией данных могут быть ослаблены, пользователь имеет возможность использовать режимы выполнения с пониженными гарантиями:

- *Online Read-Only* — каждое из чтений в транзакции читает последние на момент своего выполнения данные. Консистентность полученных данных определяется настройкой *allow_inconsistent_reads*:
  - *false* (consistent reads) — каждое из чтений по отдельности возвращает консистентные данные, но консистентность данных между разными чтениями не гарантируется. Дважды выполненное чтение одного и того же диапазона таблицы может вернуть разные результаты.
  - *true* (inconsistent reads) — данные даже для отдельно взятого чтения могут содержать неконсистентные результаты.
- *Stale Read-Only* — чтения данных в транзакции возвращают результаты с возможным отставанием от актуальных (доли секунды). Данные в каждом отдельно взятом чтении консистентны, между разными чтениями консистентность данных не гарантируется.
- *Snapshot Read-Only* — все чтения транзакции производятся из снапшота базы данных, при этом все чтения данных консистентны. Взятие снапшота происходит в момент старта транзакции, т.е. транзакция видит все изменения, закоммиченные до момента своего начала.

Режим выполнения транзакции задается в настройках транзакции при ее создании. Примеры для YDB SDK смотрите в статье Установка режима выполнения транзакции.

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, которые могут быть неочевидны на первый взгляд и 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: `BEGIN; выполнить SELECT; проанализировать результаты SELECT на клиенте; ...; выполнить UPDATE; COMMIT`. 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.
- Реализация YQL в YDB использует механизм Optimistic Concurrency Control. На затронутые в ходе транзакции сущности 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`.
- Все изменения, производимые в рамках транзакции, накапливаются в памяти сервера базы данных и применяются в момент завершения транзакции. Если взятые блокировки не были инвалидированы, то все накопленные изменения применяются атомарно, если хотя бы одна блокировка была инвалидирована, то ни одно из изменений не будет применено. Описанная схема накладывает некоторые ограничения: объем изменений, осуществляемых в рамках одной транзакции, должен умещаться в оперативную память.

Для наиболее эффективного выполнения транзакций следует формировать их таким образом, чтобы в первой части транзакции выполнялись только чтения, а во второй части транзакции только модификации. Структура запроса тогда выглядит следующим образом:

```
SELECT ...;
....
SELECT ...;
UPDATE/REPLACE/DELETE ...;
COMMIT;
```

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

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.

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*.

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 будет отменён со статусом `PRECONDITION_FAILED`. Таким образом, пользовательский код должен быть готов к обработке этого статуса.

Уникальный вторичный индекс является синхронным индексом, поэтому процесс обновления происходит так же, как описано в разделе Синхронный вторичный с точки зрения транзакции.

Ограничения

В настоящее время уникальный индекс не может быть добавлен в существующую таблицу.

Векторный индекс

Векторный индекс является особым видом вторичного индекса.

В отличие от вторичных индексов, оптимизирующих поиск по равенству или диапазону, векторные индексы позволяют выполнять векторный поиск на основе функций расстояния или схожести.

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.

> **Примечание**
>
> Пользователь, от имени которого создаётся вторичный индекс, должен обладать следующими правами:
>
> - `ydb.generic.read`;
> - `ydb.generic.write`.

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 не осуществляют полный перебор исходных данных. Это позволяет значительно ускорить процесс поиска, хотя и может привести к некоторому снижению качества результатов.

Скалярное квантование — это метод уменьшения размерности векторов, при котором множество координат отображается в пространство меньшей размерности.

YDB поддерживает векторный поиск по векторам типов `Float`, `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` каждый вектор сокращается в 32 раза.

Приближенный векторный поиск без индекса использует очень простую дополнительную структуру данных - множество векторов с другими 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;
- в текущей версии не поддерживаются операции модификации данных: вставка, обновление, удаление.

Change Data Capture (CDC)

Внимание

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

- Количество партиций топика фиксируется на момент создания потока изменений и остается неизменным (топики, в отличие от таблиц, не являются эластичными).
- В потоке изменений поддерживаются записи о следующих видах операций:
  - обновление;
  - удаление.

Таким образом, добавление строки является частным случаем обновления, и в потоке изменений запись о добавлении строки будет выглядеть аналогично записи об обновлении.

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).

Примечание

По умолчанию виртуальные метки времени не выгружаются в поток изменений. Для их включения 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.

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>},
    "erase": {},
    "newImage": {<columns>},
    "oldImage": {<columns>},
    "ts": [<step>, <txId>]
}
```

- `key` — массив значений компонент первичного ключа. Присутствует всегда.
- `update` — признак обновления. Присутствует, если запись соответствует операции обновления. В режиме `UPDATES` так же содержит названия и значения изменившихся столбцов.
- `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`).

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]
}
```

Примечание

- Одна и та же запись не может содержать поля `update` 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 или удалением).
- В режиме `UPDATES` для операций обновления поле `update` выполняет роль не только признака операции (обновление), но и содержит названия и значения изменившихся столбцов.
- Поля JSON-объекта, содержащие названия и значения столбцов (`newImage` , `oldImage` и `update` 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>
        }
    }
}
```

- `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.

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.

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`;
- 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.

Скан запросы (Scan Query) в YDB

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 5 минутами.
- Многие операции (включая сортировку) выполняются целиком в памяти, поэтому на сложных запросах можно получить ошибку нехватки ресурсов.
- Для джойнов 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());
    ...
};
```

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
База данных	Максимальная глубина пути	32	Максимальное количество	MaxDepth	SCHEME_ERROR
База данных	Максимальное количество путей (объектов в схеме)	10 000	Максимальное количество элементов пути (директорий, таблиц) в базе данных	MaxPathElements	GENERIC_ERROR
			таблеток), которое может работать в базе данных. На запрос	MaxShards	GENERIC_ERROR

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

Параметр	Значение
Максимальный размер строки	8 Мб
Максимальный размер вставляемого блока данных	8 Мб

Параметр	Значение	Пояснение	Статус в случае нарушения ограничения
Длительность запроса	1800 секунд (30 минут)	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 ранее подготовленного запроса.	BAD_REQUEST
Максимальный размер строки	50 Мб	Ограничение на суммарный размер всех полей одной строки, возвращаемой или формируемой запросом.	PRECONDITION_FAILED

База данных	Максимальная длина имени объекта	255	Ограничивает количество символов в имени объекта схемы, например директории или таблицы.	MaxPathElementLength	SCHEME_ERROR

База данных	Максимальный размер ACL	10 Кб	Максимальный суммарный размер всех	MaxAclBytesSize	GENERIC_ERROR

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

Д					
Т					
Таблица					
	количество столбцов		общее количество столбцов в таблице.		
Таблица	Максимальная длина имени столбца	255	Ограничивает количество символов в имени столбца.	MaxTableColumnNameLength	GENERIC_ERROR
Таблица	Максимальное количество столбцов в первичном ключе	20	Каждая таблица должна иметь первичный ключ. Количество столбцов, входящих в	MaxTableKeyColumns	GENERIC_ERROR

Параметр	Значение
Максимальный размер передаваемого сообщения	12 Мб

			превышать это ограничение.		
Таблица	Максимальное	20	Максимальное	MaxTableIndices	GENERIC_ERROR

Параметр	Значение
Максимальное число классификаторов пулов ресурсов	1000

			первичного ключа, которые могут быть созданы на таблице.		
Таблица	Максимальное количество фолловеров	3	Максимальное количество read-only реплик, которое можно указать при создании таблицы с фолловерами.	MaxFollowersCount	GENERIC_ERROR
Таблица	Максимальное количество копируемых таблиц	10 000	Ограничение на размер списка таблиц для операций консистентного копирования таблиц.	MaxConsistentCopyTargets	GENERIC_ERROR

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

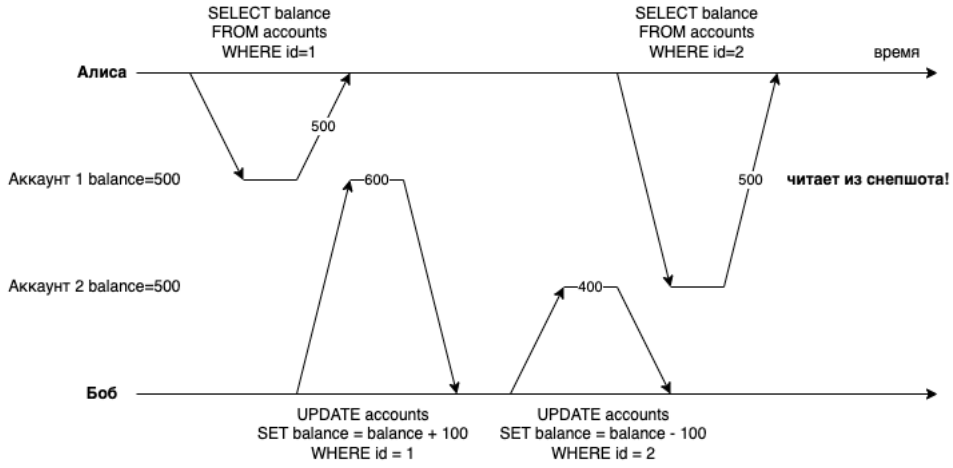
Шарды таблиц YDB хранят данные в отсортированном хранилище пар ключ-значение, реализованном как оптимизированное для записи LSM-дерево. Ранние версии YDB не использовали MVCC. Последовательность выполнения транзакций является предопределенной (что обеспечивается Координаторами -- аналогами компонентов "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.

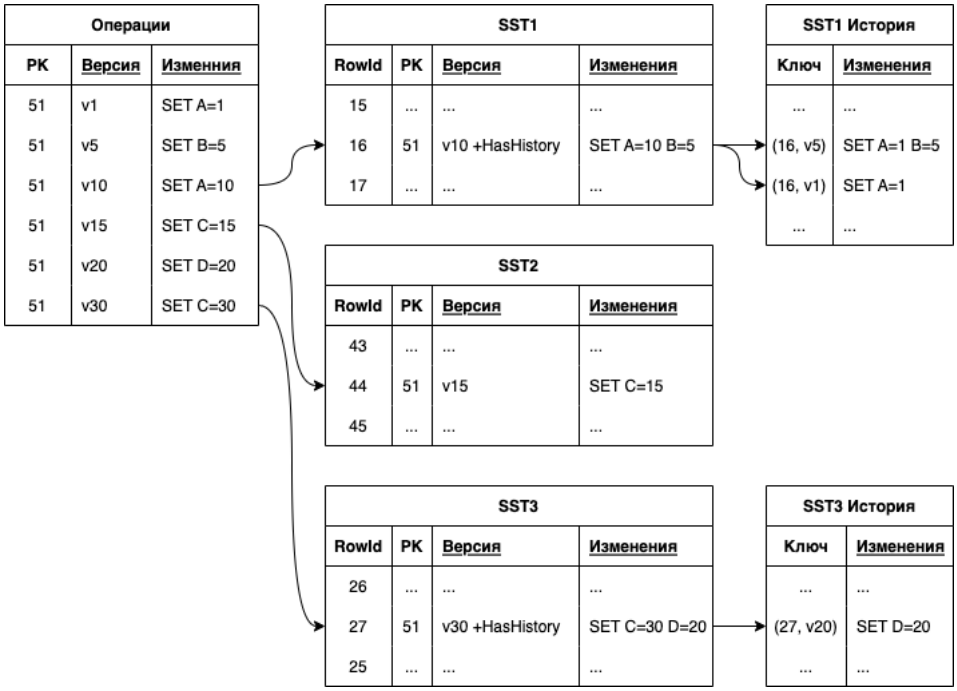


Организация хранения данных MVCC

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.

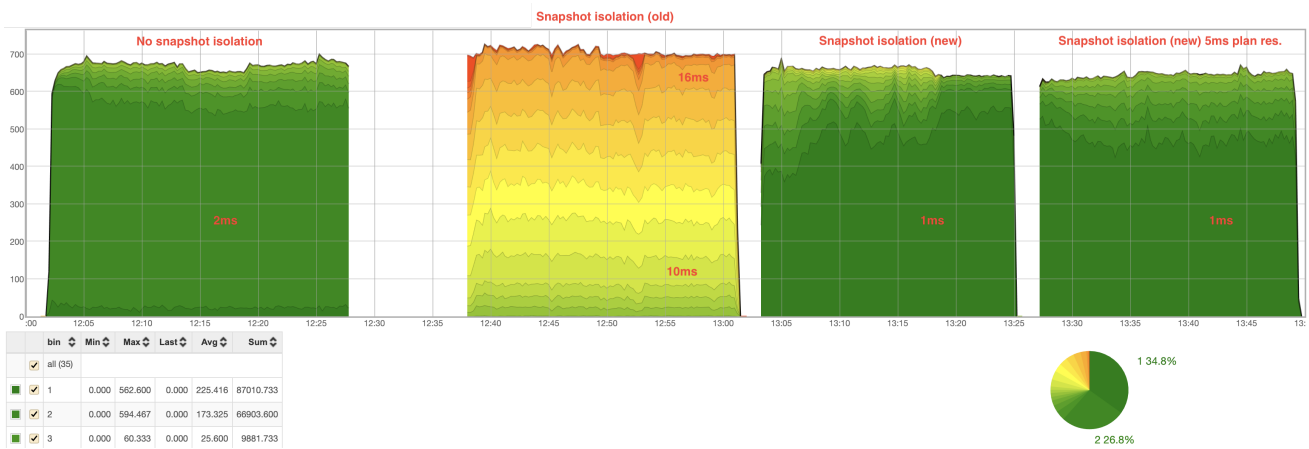
Приоритет операций чтения

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

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.
- Репликация первоначальных данных.
- 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);
  - объекты-реплики (в приемнике).

> **Примечание**
>
> Объекты-реплики создаются от имени пользователя, создавшего экземпляр асинхронной репликации.

Репликация первоначальных данных

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 репликации первоначальных данных можно получить из описания экземпляра асинхронной репликации.

Репликация изменений

После завершения первоначального сканирования читатели получают только данные об изменениях и записывают 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.

Ограничения

- Набор исходных объектов задается при создании экземпляра асинхронной репликации и не может быть изменен.
- В качестве исходных объектов поддерживаются:
  - строковые таблицы;
  - 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.

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.

> **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 консистентность 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.

Для удаления асинхронной репликации используйте выражение DROP ASYNC REPLICATION.

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.

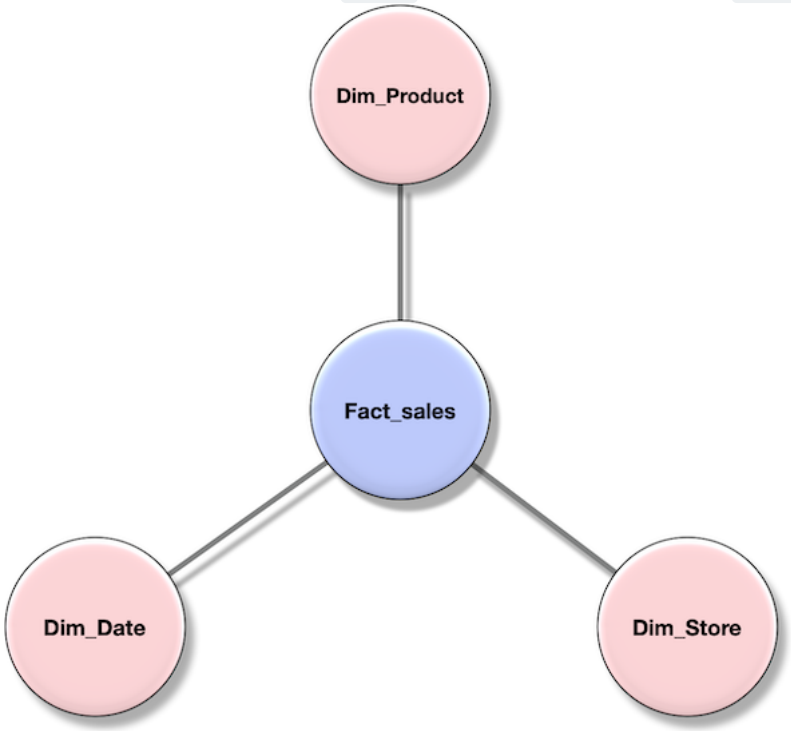
Перебор порядка соединений

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 как по таблице целиком, так и по отдельным колонкам. YDB собирает и поддерживает статистку в актуальном состоянии в фоновом режиме. Форсировать сбор статистики можно с помощью команды ANALYZE.

Текущий набор статистик по таблицам:

- Количество записей таблицы;
- Размер таблицы в байтах.

Текущий набор статистик по колонкам:

- Count-min sketch.

Текущие уровни стоимостной оптимизации

В YDB можно выставить уровень стоимостной оптимизации через прагму CostBasedOptimizationLevel

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.

Строковые таблицы

Строковые таблицы хорошо подходят для транзакционных запросов, которые генерируют Online Transaction Processing (OLTP) системы, например, бекэнды 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, состоящий 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 партиционированна 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, чтоб 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 выполняет разделение и объединение партиций, ориентируясь на размер в 2000 MB. При этом YDB не ограничивает пользовательские настройки, а работает параллельно с ними. Например, если задать размер партиции 100 MB и ограничение на 10 партиций, то, когда у таблицы станет 10 партиций по 100 MB, она перестанет делиться, и размер партиций начнет увеличиваться. Когда размер какой-либо партиции превысит 2000 MB, она разделится, и в таблице станет 11 партиций. Если размер двух соседних партиций уменьшится до 1000 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 партиции 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 он ставится в очередь на разделение (split). Если в течение часа суммарная нагрузка на два или более соседних партиций утилизировала менее 35% одного ядра CPU, то они ставятся в очередь на объединение (merge).

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 одного или нескольких фолловеров 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 фолловеров 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 запросов с фолловеров 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 (фолловерам) 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	Да	Нет

Внутреннее состояние каждого из фолловеров восстанавливается в точности по состоянию лидера и полностью консистентно.

Кроме состояния данных в сторадже на фолловеров 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 фолловерах относительно их коммита на лидере (типично в десятки миллисекунд, но может увеличиваться в случае проблем связности в кластере). В связи с этим, чтение с фолловеров обеспечивается только в режиме транзакций `StaleReadOnly()`.

Если фолловеров несколько, то отставание их от лидера может различаться, т.е. хотя каждый фолловер 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 фолловеров 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

Фильтр Блума

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.

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

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 насколько 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.

Для группы колонок устанавливаются атрибуты, влияющие на хранение 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).

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.

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.

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.

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

> **Важно**
>
> Колоночные таблицы YDB доступны в режиме Preview.

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.
- В колоночных таблицах поддерживается ограниченный набор типов данных:
  - Доступно и в первичном ключе и в остальных колонках: `Date`, `Datetime`, `Timestamp`, `Int32`, `Int64`, `Uint8`, `Uint16`, `Uint32`, `Uint64`, `Utf8`, `String`;
  - Доступно только в колонках, не входящих в первичный ключ: `Bool`, `Decimal`, `Double`, `Float`, `Int8`, `Int16`, `Interval`, `JsonDocument`, `Json`, `Uuid`, `Yson`.
- В колоночных таблицах поддерживаются группы колонок, но пока это используется только для задания сжатия на колонках.

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);
```

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

- Чтение с реплик.
- Вторичные индексы.
- Векторные индексы.
- Фильтры Блума.
- Change Data Capture.
- Переименование таблиц.
- Пользовательские атрибуты таблиц.
- Изменение списка колонок данных.
- Добавление данных в колоночные таблицы с помощью SQL-оператора INSERT.
- Удаление данных из колоночных таблиц с помощью SQL-оператора DELETE. Фактически, удаление возможно только по истечению TTL времени хранения данных.

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.

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.

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 и группы сообщений

Идентификатор источника, `producer_id`, и идентификатор группы сообщений, `message_group_id`, — это способ упорядочить набор сообщений. Порядок записанных сообщений сохраняется в пределах пар: `<producer_id, message_group_id>`.

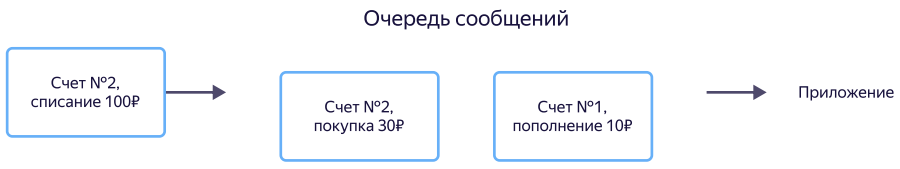
При первом использовании пара <идентификатор источника, группа сообщений> привязывается к партиции топика по алгоритму round-robin и все сообщения с этой парой попадают в одну и ту же партицию. Привязка удаляется при отсутствии новых сообщений, использующих этот идентификатор источника, в течение 14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

> **Важно**
>
> Рекомендуемое максимальное количество пар `<producer_id, message_group_id>` - до 100'000 на одну партицию за последние 14 дней.

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.

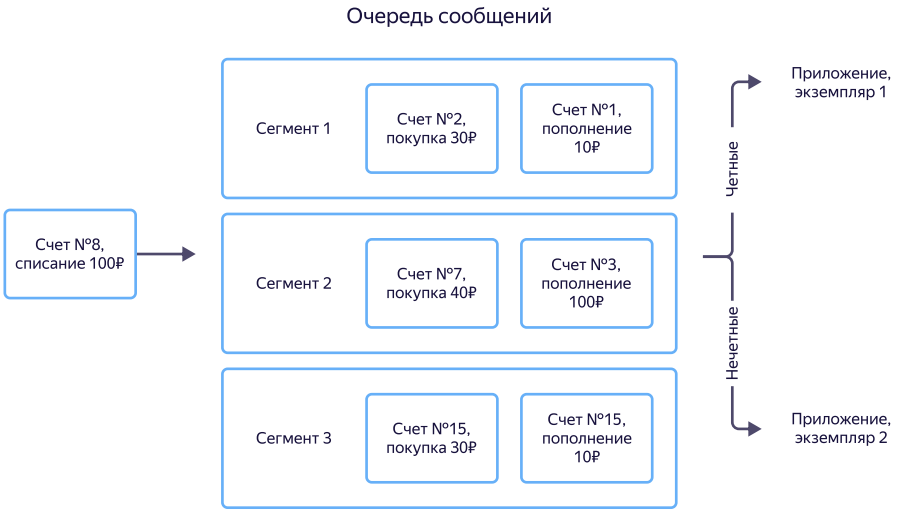


Для правильного вычисления баланса важен порядок обработки сообщений. Если пользователь сначала пополняет счет, а затем совершает покупку, то и сообщения с информацией об этих операциях должны быть обработаны приложением в такой же последовательности. Иначе может произойти ошибка бизнес-логики, и, например, 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 (например, сообщения о транзакциях по одному счету) приходили в один и тот же экземпляр приложения. Источник сообщения определяется идентификатором источника (source_id), а для защиты от дублей используется порядковый номер сообщения от источника. YDB так настраивает потоки данных, чтобы сообщения от одного источника попадают в одну и ту же партицию. Таким образом сообщения о транзакциях по заданному счету будут попадать всегда в одну и ту же партицию и обрабатываться экземпляром приложения, который связан с этим сегментом. Каждый из экземпляров обслуживает свое подмножество партиций и задача синхронизации между экземплярами не возникает.

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` или `source_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.

> **Важно**
>
> Настоятельно не рекомендуем использовать случайные или псевдослучайные идентификаторы источников. Рекомендуем использовать не более 100000 различных идентификаторов источников на одну партицию.

Идентификатор источника

Идентификатор источника является произвольной строкой длины до 2048 символов, обычно в качестве идентификатора источника используют идентификатор сервера с файлами или другой идентификатор.

Примеры идентификаторов источников

Тип	Идентификатор	Описание
Файл	Идентификатор сервера	Для хранения журналов работы приложений используют файлы. В этом случае удобно в качестве идентификатора источника использовать идентификатор сервера.
Действия пользователя	Идентификатор класса действий пользователя: "просмотр страницы", "покупка" и тд.	Важно обрабатывать действия пользователя в порядке их выполнения пользователем, при этом нет необходимости обрабатывать в одном приложении абсолютно все действия пользователя. В таком случае удобно действия пользователя группировать по классам.

Идентификатор группы сообщений

Идентификатор группы сообщений является произвольной строкой длины до 2048 символов, обычно в качестве идентификатора группы сообщений используют имя файла или идентификатор пользователя.

Примеры идентификаторов групп сообщений

Тип	Идентификатор	Описание
Файл	Полный путь к файлу	Все данные с сервера и файла на нем будут отправлены в одну партицию.
Действия пользователя	Идентификатор пользователя	Важно обрабатывать действия пользователя в порядке их выполнения, в этом случае в качестве идентификатора источника удобно использоваться идентификатор самого пользователя.

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 еще не были прочитаны "важным" читателем — они будут храниться до тех пор, пока читатель их не прочитает.

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.

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.

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

Протоколы для работы с топиками

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

Также ограниченно поддержан протокол Kafka API версии 3.4.0. (см. Работа с Kafka API).

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.

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

Важно

Данная функциональность находится в режиме "Preview".

Для работы с внешними источниками данных в YDB используются федеративные запросы. Для аутентификации во внешних системах федеративные запросы используют различные реквизиты доступа. Реквизиты доступа хранятся в отдельных объектах - секретах. Секреты доступны только для записи и обновления, получить значение секрета нельзя.

Важно

Текущий систаксис работы с секретами является временным, в будущих релизах YDB он будет изменен.

Создание секретов

Создание секрета выполняется с помощью SQL-запроса:

```
CREATE OBJECT `MySecretName` (TYPE SECRET) WITH value=`MySecretData`;
```

Управление доступом

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 `MySecretName` пользователю `another_user` необходимо создать объект `SECRET_ACCESS` с именем `MySecretName:another_user`.

```
CREATE OBJECT `MySecretName:another_user` (TYPE SECRET_ACCESS)
```

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_NAME="test_password_name",
  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:

- ClickHouse
- PostgreSQL
- Подключения к S3 (Object Storage)

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 данных из внешней таблицы, ведущей на S3 (Object Storage).

Федеративные запросы

Федеративные запросы - это способ получать информацию из различных источников данных без необходимости переноса данных этих источников внутрь YDB. В настоящее время федеративные запросы поддерживают взаимодействие с базами данных ClickHouse, PostgreSQL и с хранилищами данных класса S3. При помощи YQL запросов вы сможете обращаться к этим 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 объекты-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:

- ClickHouse
- Greenplum
- MySQL
- PostgreSQL
- S3
- 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.

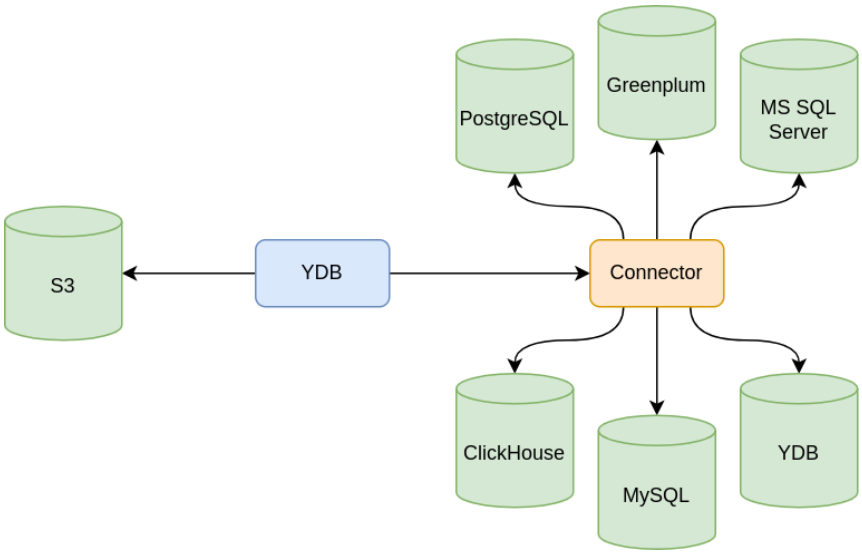
Коннекторы

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

Источник	Поддержка
ClickHouse	Через коннектор fq-connector-go
Greenplum	Через коннектор fq-connector-go
Microsoft SQL Server	Через коннектор fq-connector-go
MySQL	Через коннектор fq-connector-go

PostgreSQL	Через коннектор fq-connector-go
S3	Встроенная в `ydbd`
YDB	Через коннектор fq-connector-go

Работа с базами данных ClickHouse

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 OBJECT clickhouse_datasource_user_password (TYPE SECRET) 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_NAME="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
Фильтров вида `IS NULL` / `IS NOT NULL`	`WHERE column1 IS NULL` или `WHERE column1 IS NOT NULL`
Логических условий `OR` , `NOT` , `AND` .	`WHERE column IS NULL OR column2 is NOT NULL` .
Условий сравнения `=` , `<>` , `<` , `<=` , `>` , `>=` с другими колонками или константами.	`WHERE column3 > column4 OR column5 <= 10` .

Поддерживаемые типы данных для пушдауна фильтров:

Тип данных YDB
`Bool`
`Int8`
`Uint8`
`Int16`
`Uint16`
`Int32`
`Uint32`
`Int64`
`Uint64`
`Float`
`Double`

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

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 OBJECT greenplum_datasource_user_password (TYPE SECRET) 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_NAME="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
Фильтров вида `IS NULL` / `IS NOT NULL`	`WHERE column1 IS NULL` или `WHERE column1 IS NOT NULL`
Логических условий `OR`, `NOT`, `AND`.	`WHERE column IS NULL OR column2 is NOT NULL`.
Условий сравнения `=`, `<>`, `<`, `<=`, `>`, `>=` с другими колонками или константами.	`WHERE column3 > column4 OR column5 <= 10`.

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

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 OBJECT ms_sql_server_datasource_user_password (TYPE SECRET) 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_NAME`. 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_NAME="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
Фильтров вида `IS NULL` / `IS NOT NULL`	`WHERE column1 IS NULL` или `WHERE column1 IS NOT NULL`
Логических условий `OR`, `NOT`, `AND`.	`WHERE column IS NULL OR column2 is NOT NULL`.
Условий сравнения `=`, `<>`, `<`, `<=`, `>`, `>=` с другими колонками или константами.	`WHERE column3 > column4 OR column5 <= 10`.

Поддерживаемые типы данных для пушдауна фильтров:

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

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 OBJECT mysql_datasource_user_password (TYPE SECRET) 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_NAME`. 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_NAME="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
Фильтров вида `IS NULL` / `IS NOT NULL`	`WHERE column1 IS NULL` или `WHERE column1 IS NOT NULL`
Логических условий `OR`, `NOT`, `AND`.	`WHERE column IS NULL OR column2 is NOT NULL`.
Условий сравнения `=`, `<>`, `<`, `<=`, `>`, `>=` с другими колонками или константами.	`WHERE column3 > column4 OR column5 <= 10`.

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

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 OBJECT postgresql_datasource_user_password (TYPE SECRET) 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_NAME="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
Фильтров вида `IS NULL` / `IS NOT NULL`	`WHERE column1 IS NULL` или `WHERE column1 IS NOT NULL`
Логических условий `OR`, `NOT`, `AND`.	`WHERE column IS NULL OR column2 is NOT NULL`.
Условий сравнения `=`, `<>`, `<`, `<=`, `>`, `>=` с другими колонками или константами.	`WHERE column3 > column4 OR column5 <= 10`.

Поддерживаемые типы данных для пушдауна фильтров:

Тип данных YDB
`Bool`
`Int8`
`Int16`
`Int32`
`Int64`
`Float`
`Double`

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>`	

Работа с базами данных YDB

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 OBJECT ydb_datasource_user_password (TYPE SECRET) 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_NAME`. 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_NAME="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

Фильтров вида `IS NULL` / `IS NOT NULL`	`WHERE column1 IS NULL` или `WHERE column1 IS NOT NULL`
Логических условий `OR`, `NOT`, `AND`.	`WHERE column IS NULL OR column2 is NOT NULL`.
Условий сравнения `=`, `<>`, `<`, `<=`, `>`, `>=` с другими колонками или константами.	`WHERE column3 > column4 OR column5 <= 10`.

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`

Работа с бакетами S3 (Yandex Object Storage)

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_NAME` — ссылка на имя секрета, в котором хранится `AWS_ACCESS_KEY_ID`.
- `AWS_SECRET_ACCESS_KEY_SECRET_NAME` — ссылка на имя секрета, 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 внешнее подключение с именем `object_storage`, которое будет указывать на конкретный S3-бакет с именем `bucket`.

```
CREATE EXTERNAL DATA SOURCE object_storage WITH (
  SOURCE_TYPE="ObjectStorage",
  LOCATION="https://object_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 OBJECT aws_access_id (TYPE SECRET) WITH (value=`<id>`);
    CREATE OBJECT aws_access_key (TYPE SECRET) WITH (value=`<key>`);
```

Следующим шагом создаётся внешнее подключение с именем `object_storage`, которое будет указывать на конкретный S3-бакет с именем `bucket`, а также использовать `AUTH_METHOD="AWS"`, для которого заполняются параметры `AWS_ACCESS_KEY_ID_SECRET_NAME`, `AWS_SECRET_ACCESS_KEY_SECRET_NAME`, `AWS_REGION`. Значения этих параметров описаны выше.

```
CREATE EXTERNAL DATA SOURCE object_storage WITH (
  SOURCE_TYPE="ObjectStorage",
  LOCATION="https://object_storage_domain/bucket/",
  AUTH_METHOD="AWS",
  AWS_ACCESS_KEY_ID_SECRET_NAME="aws_access_id",
  AWS_SECRET_ACCESS_KEY_SECRET_NAME="aws_access_key",
  AWS_REGION="ru-central-1"
);
```

Использование внешнего подключения к S3-бакету

При работе с Yandex Object Storage с помощью внешних источников данных удобно выполнять прототипирование, первоначальную настройку подключений к данным.

Пример запроса для чтения данных:

```
SELECT
  *
FROM
  object_storage.`*.tsv`
WITH
(
  FORMAT = "tsv_with_names",
  SCHEMA =
  (
    ts Uint32,
    action Utf8
  )
);
```

Список поддерживаемых форматов и алгоритмов сжатия данных для чтения данных в S3 (Yandex Object Storage), приведен в разделе Форматы данных и алгоритмы сжатия.

Модель данных

В Yandex Object Storage данные хранятся в файлах. Для чтения данных необходимо указать формат данных в файлах, сжатие, списки полей. Для этого используется следующее SQL-выражение:

```
SELECT
  <expression>
FROM
  <object_storage_connection_name>.`<file_path>`
WITH(
  FORMAT = "<file_format>",
  COMPRESSION = "<compression>",
  SCHEMA = (<schema_definition>),
  <format_settings>)
WHERE
  <filter>;
```

Где:

- `object_storage_connection_name` — название внешнего источника данных, ведущего на бакет с S3 (Yandex Object Storage).
- `file_path` — путь к файлу или файлам внутри бакета. Поддерживаются wildcards `*`, подробнее в разделе.
- `file_format` — формат данных в файлах.
- `compression` — формат сжатия файлов.
- `schema_definition` — описание схемы хранимых данных в файлах.
- `format_settings` — опциональные параметры форматирования

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.

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
  <object_storage_connection_name>.`<file_path>`
WITH(
  FORMAT = "<file_format>",
  COMPRESSION = "<compression>",
  WITH_INFER = "true")
WHERE
  <filter>;
```

Где:

- `object_storage_connection_name` — название внешнего источника данных, ведущего на S3 бакет (Yandex Object Storage).
- `file_path` — путь к файлу или файлам внутри бакета. Поддерживаются wildcards `*`, подробнее ниже.
- `file_format` — формат данных в файлах. Поддерживаются все форматы, кроме `raw` и `json_as_string`.
- `compression` — формат сжатия файлов.

В результате выполнения такого запроса будут автоматически выведены названия и типы полей.

Форматы путей к данным задаваемых в параметре `file_path`

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 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 не завершается символом `/` и не содержит символов макроподстановок	Путь к отдельному файлу	Путь `/a/b.csv` адресует конкретный файл `/a/b.csv`

Параметры форматирования

В YDB поддерживаются следующие параметры форматирования:

Имя параметра	Описание	Принимаемые значения
`file_pattern`	Шаблон имени файла	Строка шаблона имени. Поддерживаются wildcards `*`.
`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)

Параметр `file_pattern` можно использовать только в том случае, если `file_path` – путь к каталогу. В строках форматирования 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 (Yandex Object Storage):

```
SELECT
  *
FROM
  connection.`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:

- `connection` — название внешнего источника данных, ведущего на бакет S3 (Yandex Object Storage).
- `folder/` — путь к папке с данными в бакете S3 (Yandex Object Storage).
- `SCHEMA` — описание схемы данных в файле.
- `*.csv.gz` — шаблон имени файлов с данными.
- `%Y-%m-%d` — формат записи данных типа `Date` в S3.

Чтение данных из внешней таблицы, ведущей на S3 (Object Storage)

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 (Object Storage)

Чтобы создать внешнюю таблицу, описывающую бакет S3 (Object Storage), 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;
- `bucket` - имя внешнего источника данных к S3 (Object Storage);
- `folder` - путь внутри бакета с данными;
- `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 (Object Storage) выполняется с помощью обычных SQL-запросов, как к обычной таблице.

```
SELECT
    <expression>
FROM
    `s3_test_data`
WHERE
    <filter>;
```

Ограничения

При работе с бакетами S3 (Object Storage) 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 (Yandex Object Storage)

В YDB для записи данных в бакет Yandex Object Storage можно использовать внешние соединения или внешние таблицы.

Запись данных через внешнее соединение

Запись данных с помощью соединений удобно использовать для прототипирования и первоначальной настройки работы с записью данных. SQL-выражение иллюстрирует запись данных во внешний источник данных напрямую.

```
INSERT INTO `connection`.`test/`
WITH
(
  FORMAT = "csv_with_names"
)
SELECT
    "value" AS value, "name" AS name
```

Запись будет произведена в указанный путь. При этом режиме результирующие файлы не будут партиционированы. Если необходимо партиционировать результирующие файлы, то используйте запись через внешние таблицы. Создаваемые при записи файлы получают случайные имена.

При работе с внешними соединениями 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 в S3 (Yandex Object Storage) и используйте обычное SQL-выражение `INSERT INTO` :

```
INSERT INTO `test`
SELECT
    "value" AS value, "name" AS name
```

Форматы данных и алгоритмы сжатия

В данном разделе описываются поддерживаемые в YDB форматы данных, хранимых в S3, и поддерживаемые алгоритмы сжатия.

Поддерживаемые форматы данных

Список поддерживаемых в YDB форматов данных приведен в таблице ниже.

Формат	Чтение	Запись
`csv_with_names`	✓	✓
`tsv_with_names`	✓	
`json_list`	✓	
`json_each_row`	✓	
`json_as_string`	✓	
`parquet`	✓	✓
`raw`	✓	

Формат csv_with_names

Данный формат основан на формате CSV. Данные размещены в колонках, разделены запятыми, 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
    AVG(Price)
FROM `connection`.`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
    AVG(Price)
FROM `connection`.`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 }
```

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
    AVG(Price)
FROM `connection`.`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, "Manufacturer": "Man_1", "Model": "Model_1", "Price": 3000.0 }
{ "Year": 1999, "Manufacturer": "Man_2", "Model": "Model_2", "Price": 4900.00 }
```

Пример запроса

```
SELECT
    *
FROM `connection`.`path`
WITH
(
    FORMAT = "json_as_string",
    SCHEMA =
    (
        Data Json
    )
)
```

Результат выполнения запроса:

#	Data
1	`{"Manufacturer": "Man_1", "Model": "Model_1", "Price": 3000, "Year": 1997}`
2	`{"Manufacturer": "Man_2", "Model": "Model_2", "Price": 4900, "Year": 1999}`

Формат parquet

Данный формат позволяет считывать содержимое файлов в формате Apache Parquet.

Поддерживаемые алгоритмы сжатия данных внутри файлов Parquet:

- Без сжатия
- SNAPPY
- GZIP
- LZO
- BROTLI
- LZ4
- ZSTD
- LZ4_RAW

Пример запроса

```
SELECT
    AVG(Price)
FROM `connection`.`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 не достаточно.

Пример запроса

```
SELECT
    *
FROM `connection`.`path`
WITH
(
    FORMAT = "raw",
    SCHEMA =
    (
        Data String
    )
)
```

Результат выполнения запроса:

```
Year,Manufacturer,Model,Price
1997,Man_1,Model_1,3000.00
1999,Man_2,Model_2,4900.00
```

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	Название в YDB	Чтение	Запись
Raw	raw	✓	
Snappy	snappy	✓	✓

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.

Партицирование данных в S3 (Yandex Object Storage)

В S3 (Yandex Object Storage) 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 (Yandex Object Storage). 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
    objectstorage.'/'
WITH
(
    SCHEMA =
    (
        data String,
        year Int32,
        month Int32
    )
)
WHERE
    year=2021
    AND month=02
```

Если схема партицирования данных не указана, то из S3 (Yandex Object Storage) 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 (Yandex Object Storage) размещены в каталогах по годам и месяцам

```
SELECT
    *
FROM
    objectstorage.'/'
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 (Yandex Object Storage) 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 соединений. Такой пример выбран только для иллюстративных целей. Мы настоятельно рекомендуем для работы с данными использовать "привязки к данным" и не использовать прямую работу с соединениями.

Синтаксис

При работе на уровне соединений партицирование задается с помощью параметра `partitioned_by`, где в JSON-формате задается список колонок.

```
SELECT
    *
FROM
    <connection>.<path>
WITH
(
    SCHEMA=(<field1>, <field2>, <field3>),
    PARTITIONED_BY="['field2', 'field3']"
)
```

В параметре `partitioned_by` перечисляются колонки схемы данных, по которым партицированы хранимые в S3 (Yandex Object Storage) данные. Порядок указания полей в параметре partitioned_by определяет вложенность каталогов S3 (Yandex Object Storage) 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 (Yandex Object Storage).

Предположим, что данные в S3 (Yandex Object Storage) 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
    objectstorage.'/'
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 (Yandex Object Storage), 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 (Yandex Object Storage), 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 каталога S3 (Yandex Object Storage).

Синтаксис

Расширенное партицирование называется "partition projection" и задается через параметр `projection`.

Пример указания расширенного партицирования:

```
SELECT
    *
FROM
    <connection>.`/`
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
    <connection>.<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 (Yandex Object Storage) 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`.

Концепции для DevOps-инженеров

Данный раздел дополняет общий раздел Обзор YDB теоретическими материалами, актуальными в первую очередь для DevOps-инженеров.

Основные темы:

- Системные требования и рекомендации для YDB
- Версионирование YDB
- Обслуживание кластера без потери доступности

Смотрите также:

- Способы развёртывания YDB
- Конфигурация кластеров YDB
- Обзор наблюдаемости
- Резервное копирование и восстановление

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, используемой в версиях YDB до 25.1.
- Конфигурация V2 — раздел о конфигурации V2, рекомендуемой 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.

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.

Обзор наблюдаемости

Данный раздел содержит описания по работе с инструментами наблюдения за кластером YDB.

Основные подразделы:

- Настройка мониторинга кластера YDB
- Логирование в YDB
- Трассировка в YDB
- Системные представления кластера

Резервное копирование и восстановление

Резервное копирование применяется для защиты от потери данных, позволяя восстановить их из резервной копии.

YDB предоставляет несколько решений для выполнения резервного копирования и восстановления:

- Резервное копирование в файлы и восстановление с помощью YDB CLI.
- Резервное копирование в S3-совместимое хранилище и восстановление с помощью YDB CLI.

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.

> **Примечание**
>
> Скорость операций резервного копирования и восстановления в/из S3-совместимого хранилища подобрана таким образом, чтобы минимизировать влияние на пользовательскую нагрузку. Для управления скоростью операций настройте лимиты для соответствующей очереди брокера ресурсов.

< path="_includes/backup_and_recovery/others_overlay.md" keyword="undefined">

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.

- **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.

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. YDB использует аллокатор памяти TCMalloc, для его эффективной работы рекомендуется включить Transparent Huge Pages и Memory overcommitment.

Если сервер имеет более 32 ядер CPU, для повышения производительности YDB динноды следует запускать в раздельных taskset/cpuset, размером от 10 до 32 ядер. Например, в случае 128 ядер оптимально запустить 4 динноды: каждую в своём taskset, размером 32 ядра.

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.

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. Номер мажорной версии внутри года;
3. Номер минорной версии внутри мажорной;
4. Номер патча внутри минорной;
5. Тип релиза.

Таким образом, мажорные релизы обычно идентифицируются двумя компонентами, например, `24.3` — третий мажорный релиз в 2024 году. Минорные релизы идентифицируются тремя компонентами, например, `24.3.14` — четырнадцатый релиз в рамках третьей мажорной версии в 2024 году.

Список доступных версий можно получить на странице загрузки. Релизная политика YDB описана более детально в статье Управление релизами в разделе документации для разработчиков YDB.

Совместимость версий

Примечание

Раньше версия сервера YDB состояла из 3 чисел (например, `v24.3.3`), начиная с мажорной версии `25.1`, добавлена четвертое число, которое обозначает номер патча (например, `v25.1.1.3`). Подробнее про изменения в именовании версий тут.

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: `24.3.13.6-testing`
- Стабильная версия: `24.3.14.2-stable`
- LTS версия: `24.3.14.2-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.
- **Force**: принудительный режим, модель отказа игнорируется. *Не рекомендуется к использованию*.

Приоритет

У задачи обслуживания можно указать приоритет. Меньшее значение означает более высокий приоритет.

Действия задачи не могут быть выполнены до завершения всех конфликтующих действий из задач с более высоким приоритетом. Задачи с одинаковым приоритетом не имеют преимущества друг перед другом.

Лимиты недоступных узлов

В конфигурации CMS можно настроить лимиты на количество недоступных узлов для базы данных (тенанта) или для кластера в целом. Поддерживаются относительные и абсолютные лимиты.

По умолчанию допускается не более 13 % недоступных узлов для каждой базы данных и кластера в целом.

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.

Практика

- Для кластеров, развёрнутых вручную

Конфигурация V1

Важно

Данная статья описывает работу с кластерами YDB, использующими конфигурацию V1. Конфигурация V1 является основным способом конфигурирования кластеров под управлением YDB версии ниже v25.1. Начиная с версии v25.1, YDB поддерживает конфигурацию V2, которая является рекомендуемым способом конфигурирования для кластеров YDB версии v25.1 и выше. Для кластеров, использующих конфигурацию V2, данная инструкция **не применима**.

Если ваш кластер уже обновлён до версии v25.1 или выше, но ещё использует конфигурацию V1, рекомендуется запланировать миграцию на конфигурацию V2, так как поддержка конфигурации V1 будет прекращена в будущих версиях YDB.

В этом разделе документации YDB описана Конфигурация V1, являющаяся основным способом конфигурирования кластеров YDB развернутых с использованием версий YDB ниже v25.1.

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.

Начиная с версии v25.1, YDB поддерживает Конфигурацию V2 — унифицированный подход к конфигурации, в формате единого файла. При использовании Конфигурации V2 становится возможным применение автоматическая конфигурация статической группы и State Storage. При развертывании новых кластеров на версии YDB v25.1 и выше рекомендуется использовать Конфигурацию V2.

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
- Перемещение State Storage
- Перемещение статической группы
- Замена FQDN узла
- Аутентификация и авторизация узлов баз данных

Конфигурация V2

В этом разделе документации описан способ конфигурации кластеров YDB под названием V2, являющаяся рекомендуемым способом конфигурирования кластеров YDB версий YDB v25.1 и выше.

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.

Основные материалы:

- Обзор конфигурации V2
- Обновление конфигурации кластеров YDB
- DSL конфигурации кластера
- Параметры конфигурации
- Расширение кластера
- Перемещение State Storage
- Перемещение статической группы
- Замена FQDN узла
- Аутентификация и авторизация узлов баз данных

Миграция конфигурации кластера

В YDB поддерживается два механизма управления конфигурацией: V1 и V2 (рекомендуемый, 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 (например, был развернут на версии YDB до 25.1 и был обновлён до 25.1), вы можете перейти на механизм конфигурации V2.
- **На конфигурацию V1:** Если после перехода на конфигурацию V2 возникли непредвиденные проблемы, или вам необходимо откатить версию YDB ниже 25.1, 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.

Проверка версии конфигурации

Существует два основных способа проверить, какую версию механизма конфигурации (V1 или V2) используют узлы вашего кластера YDB:

1. Через Embedded UI
2. Через метрики кластера

Через Embedded UI

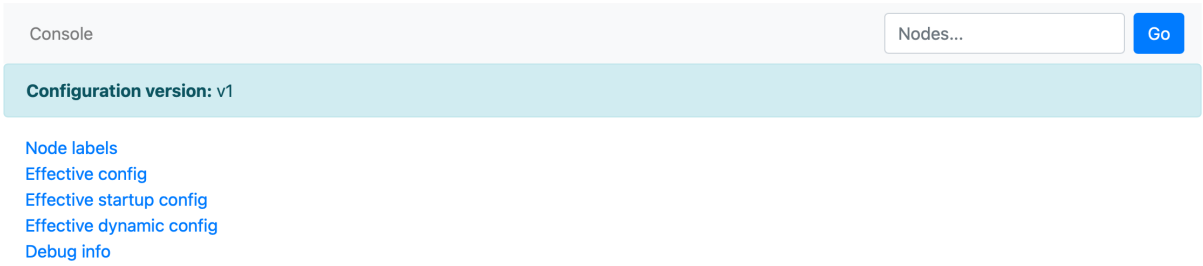
Данный способ может быть применен, если сбор метрик с кластера YDB в систему мониторинга не настроен. Вы можете проверить версию конфигурации для конкретного узла или переключаться между узлами через встроенный веб-интерфейс Embedded UI:

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).

> **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 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.
Рекомендуется для версий YDB	Все версии до 25.1.	Версия 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.

Если ваш кластер работает на конфигурации V1, рекомендуется выполнить миграцию на конфигурацию V2.

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.

Использование конфигурации V2 является рекомендуемым для всех кластеров YDB версии 25.1 и выше.

Обзор конфигурации V1

Важно

Данная статья описывает работу с кластерами YDB, использующими конфигурацию V1. Конфигурация V1 является основным способом конфигурирования кластеров под управлением YDB версии ниже v25.1. Начиная с версии v25.1, YDB поддерживает конфигурацию V2, которая является рекомендуемым способом конфигурирования для кластеров YDB версии v25.1 и выше. Для кластеров, использующих конфигурацию V2, данная инструкция **не применима**.

Если ваш кластер уже обновлён до версии v25.1 или выше, но ещё использует конфигурацию V1, рекомендуется запланировать миграцию на конфигурацию V2, так как поддержка конфигурации V1 будет прекращена в будущих версиях YDB.

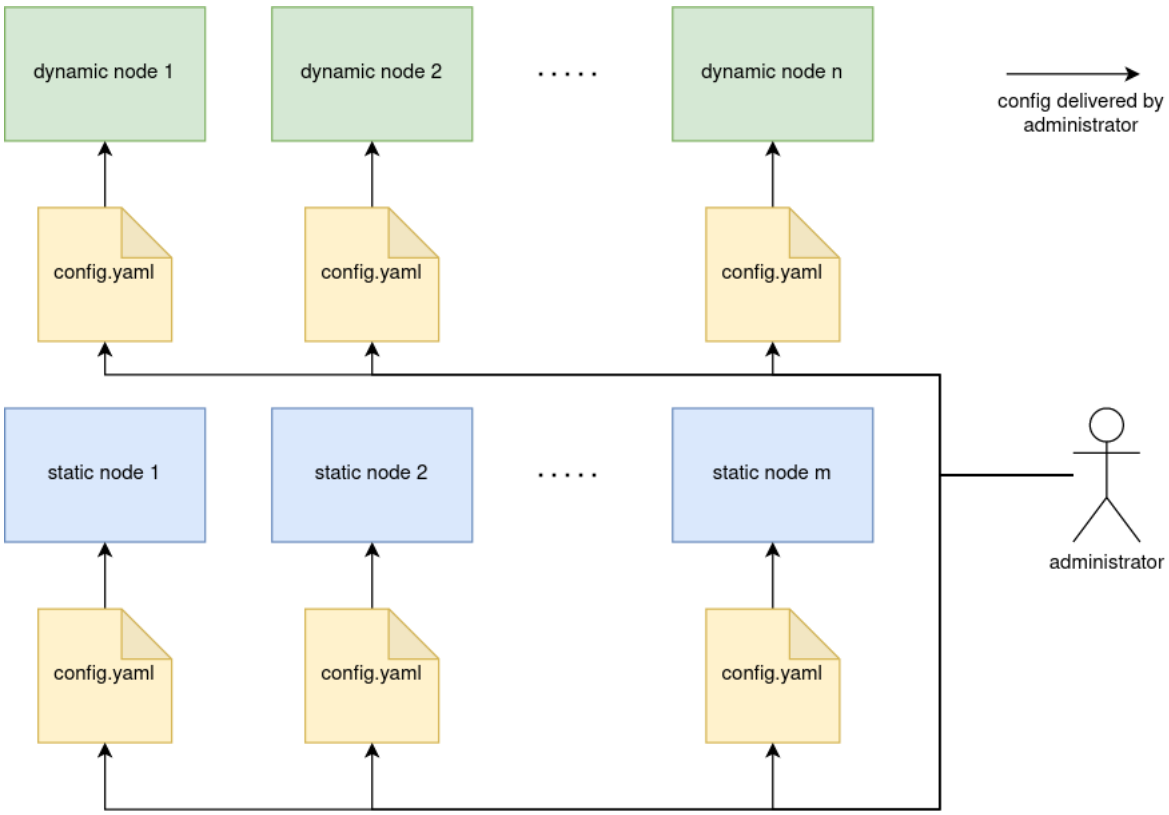
Для запуска узла YDB требуется конфигурация. Существуют два типа конфигурации:

- **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.

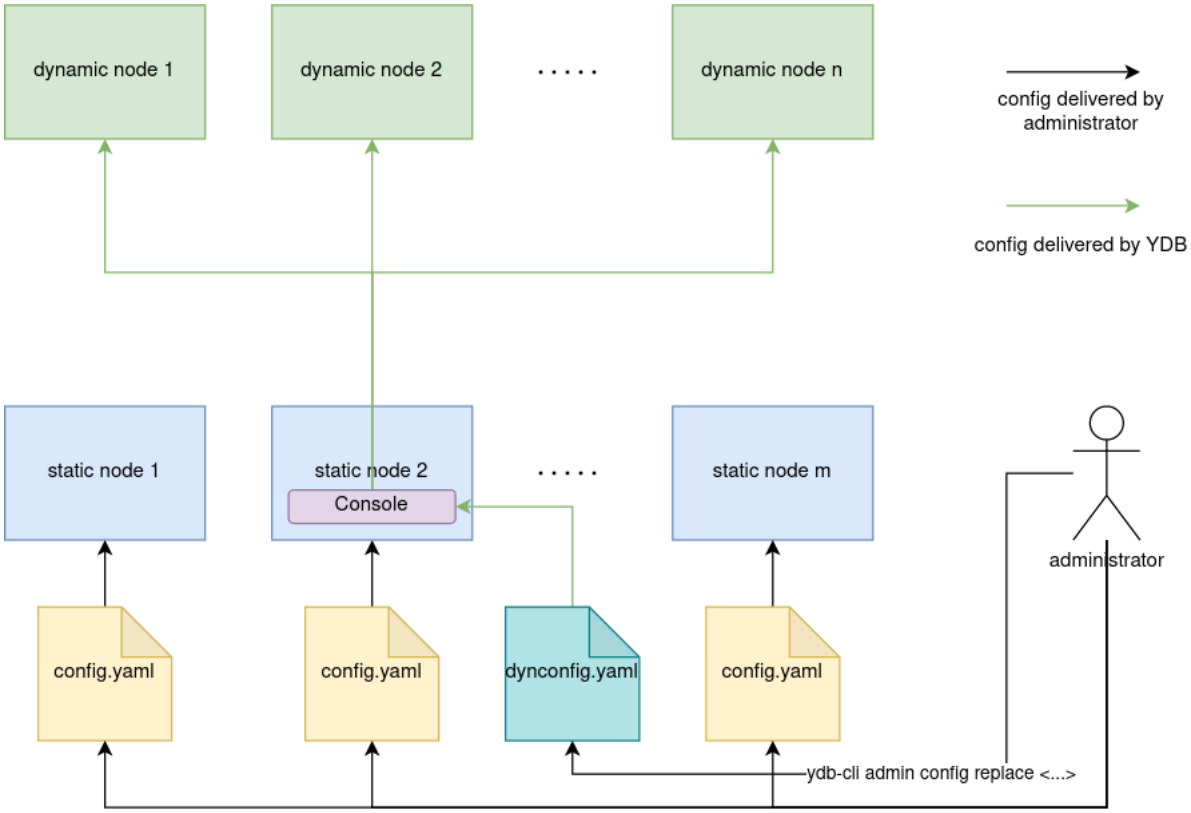


Базовый сценарий использования

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

В данной статье приведено описание основных групп конфигурируемых параметров в файле конфигурации кластера.

host_configs — типовые конфигурации хостов

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>
  - 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 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)

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
```

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 — статические узлы кластера

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 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.

log_config — конфигурация логирования

Секция `log_config` управляет тем, как YDB обрабатывает и управляет своими логами. Она позволяет настраивать уровни логирования, форматы и места назначения для различных компонентов.

Подробную информацию о параметрах конфигурации логирования смотрите в Секция конфигурации `log_config`.

Топология кластера

Для конфигурации топологии кластера YDB доступны следующие режимы отказоустойчивости:

Режим	Описание
`none`	Избыточность отсутствует. Применяется для тестирования.
`block-4-2`	Избыточность с коэффициентом 1,5, применяется для однодатацентровых кластеров.
`mirror-3-dc`	Избыточность с коэффициентом 3, применяется для мультидатацентровых кластеров.

Топология кластера задается в качестве параметра `erasure` и принимает одно из значений, указанных в таблице выше.

Конфигурация Blob Storage

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

```
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

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 учетом 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 3 работающих узла (из 5) остается при сбое двух доменов.
Геораспределенный	9	Выберите три узла в разных доменах отказа внутри каждой из трех зон доступности пула хранения mirror-3-dc для гарантии, что большинство в 5 работающих узлов (из 9) остается при сбое зоны доступности + домена отказа.

При развертывании 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 нечетные числа, так как использование четных чисел не улучшает отказоустойчивость по сравнению с ближайшим меньшим нечетным числом.

domains_config — домен кластера

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.
Георасределенный	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.

Примеры

`block-4-2`

```
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
```

`mirror-3-dc`

```
domains_config:
  domain:
  - name: global
    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
```

`none` (без отказоустойчивости)

```
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 пулов

```
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
```

Настройка акторной системы

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 Embedded UI.

Автоматическое конфигурирование

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:

```yaml
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.

Контроллер памяти

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. 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.
- KQP: хранит промежуточные результаты обработки запросов.
- 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:

- KQP.

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.

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 KQP:

```
memory_controller_config:
  query_execution_limit_percent: 25
```

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 KQP.

blob_storage_config — статическая группа кластера

Укажите конфигурацию статической группы кластера. Статическая группа необходима для работы базовых таблеток кластера, в том числе `Hive`, `SchemeShard`, `BlobstorageContoller`.
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 нодах, 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 AZ необходимо указать 3 кольца. Для конфигураций, расположенных в одной AZ, указывается ровно одно кольцо.

Настройка стабильных имен узлов кластера

Присвоение имен узлам осуществляет Node Broker — системная таблетка, которая отвечает за регистрацию динамических узлов в кластере YDB.

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, необходимо добавить в конфигурацию кластера следующее:

```
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 — брокер ресурсов

Брокер ресурсов — это 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`	2	—	Операции восстановления из резервной копии.
`queue_build_index`	10	—	Операции онлайн-создания вторичного индекса.
`queue_cdc_initial_scan`	4	—	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
```

Настройка Health Check

В этом разделе настраиваются пороговые значения и таймауты, используемые сервисом Health Check YDB. Эти параметры помогают настраивать возможные проблемы, такие как чрезмерные перезапуски или расхождение по времени между динамическими узлами.

Синтаксис

```
healthcheck_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)

Примеры конфигураций кластеров

В репозитории можно найти модельные примеры конфигураций кластеров для самостоятельного развертывания. Ознакомьтесь с ними перед развертыванием кластера.

Динамическая конфигурация кластера

Важно

Данная статья описывает работу с кластерами YDB, использующими конфигурацию V1. Конфигурация V1 является основным способом конфигурирования кластеров под управлением YDB версии ниже v25.1. Начиная с версии v25.1, YDB поддерживает конфигурацию V2, которая является рекомендуемым способом конфигурирования для кластеров YDB версии v25.1 и выше. Для кластеров, использующих конфигурацию V2, данная инструкция **не применима**.

Если ваш кластер уже обновлён до версии v25.1 или выше, но ещё использует конфигурацию V1, рекомендуется запланировать миграцию на конфигурацию V2, так как поддержка конфигурации V1 будет прекращена в будущих версиях YDB.

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 (feature flags) 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 выставлен
  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 селекторами.
# 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

Важно

Данная статья описывает работу с кластерами YDB, использующими конфигурацию V1. Конфигурация V1 является основным способом конфигурирования кластеров под управлением YDB версии ниже v25.1. Начиная с версии v25.1, YDB поддерживает конфигурацию V2, которая является рекомендуемым способом конфигурирования для кластеров YDB версии v25.1 и выше. Для кластеров, использующих конфигурацию V2, данная инструкция **не применима**.

Если ваш кластер уже обновлён до версии v25.1 или выше, но ещё использует конфигурацию V1, рекомендуется запланировать миграцию на конфигурацию V2, так как поддержка конфигурации V1 будет прекращена в будущих версиях YDB.

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

Важно

Данная статья описывает работу с кластерами YDB, использующими конфигурацию V1. Конфигурация V1 является основным способом конфигурирования кластеров под управлением YDB версии ниже v25.1. Начиная с версии v25.1, YDB поддерживает конфигурацию V2, которая является рекомендуемым способом конфигурирования для кластеров YDB версии v25.1 и выше. Для кластеров, использующих конфигурацию V2, данная инструкция **не применима**.

Если ваш кластер уже обновлён до версии v25.1 или выше, но ещё использует конфигурацию V1, рекомендуется запланировать миграцию на конфигурацию V2, так как поддержка конфигурации V1 будет прекращена в будущих версиях YDB.

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 одновременно 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

Изначальная конфигурация	Переопределение	Результирующая конфигурация

`config:`
`  some_config:`
`  array:`
`  - abc: 2`
	`config:`
	`  some_config: !inherit`
	`  array: !inherit:abc`
	`  - abc: 1`
		`config:`
		`  some_config:`
		`  array:`
		`  - abc: 2`

`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

Важно

Данная статья описывает работу с кластерами YDB, использующими конфигурацию V1. Конфигурация V1 является основным способом конфигурирования кластеров под управлением YDB версии ниже v25.1. Начиная с версии v25.1, YDB поддерживает конфигурацию V2, которая является рекомендуемым способом конфигурирования для кластеров YDB версии v25.1 и выше. Для кластеров, использующих конфигурацию V2, данная инструкция **не применима**.

Если ваш кластер уже обновлён до версии v25.1 или выше, но ещё использует конфигурацию V1, рекомендуется запланировать миграцию на конфигурацию V2, так как поддержка конфигурации V1 будет прекращена в будущих версиях YDB.

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

Важно

Данная статья описывает работу с кластерами YDB, использующими конфигурацию V1. Конфигурация V1 является основным способом конфигурирования кластеров под управлением YDB версии ниже v25.1. Начиная с версии v25.1, YDB поддерживает конфигурацию V2, которая является рекомендуемым способом конфигурирования для кластеров YDB версии v25.1 и выше. Для кластеров, использующих конфигурацию V2, данная инструкция **не применима**.

Если ваш кластер уже обновлён до версии v25.1 или выше, но ещё использует конфигурацию V1, рекомендуется запланировать миграцию на конфигурацию V2, так как поддержка конфигурации V1 будет прекращена в будущих версиях YDB.

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

Важно

Данная статья описывает работу с кластерами YDB, использующими конфигурацию V1. Конфигурация V1 является основным способом конфигурирования кластеров под управлением YDB версии ниже v25.1. Начиная с версии v25.1, YDB поддерживает конфигурацию V2, которая является рекомендуемым способом конфигурирования для кластеров YDB версии v25.1 и выше. Для кластеров, использующих конфигурацию V2, данная инструкция **не применима**.

Если ваш кластер уже обновлён до версии v25.1 или выше, но ещё использует конфигурацию V1, рекомендуется запланировать миграцию на конфигурацию V2, так как поддержка конфигурации V1 будет прекращена в будущих версиях YDB.

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 во встроенном 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 во встроенном 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 ItemConfigGenerationExpected# 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 во встроенном 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.

Перемещение State Storage

Важно

Данная статья описывает работу с кластерами YDB, использующими конфигурацию V1. Конфигурация V1 является основным способом конфигурирования кластеров под управлением YDB версии ниже v25.1. Начиная с версии v25.1, YDB поддерживает конфигурацию V2, которая является рекомендуемым способом конфигурирования для кластеров YDB версии v25.1 и выше. Для кластеров, использующих конфигурацию V2, данная инструкция **не применима**.

Если ваш кластер уже обновлён до версии v25.1 или выше, но ещё использует конфигурацию V1, рекомендуется запланировать миграцию на конфигурацию V2, так как поддержка конфигурации V1 будет прекращена в будущих версиях YDB.

Если нужно вывести из эксплуатации хост кластера YDB, на котором располагается часть State Storage, необходимо переместить её на другой хост.

Важно

Неправильная последовательность действий или ошибка в конфигурации 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

Важно

Данная статья описывает работу с кластерами YDB, использующими конфигурацию V1. Конфигурация V1 является основным способом конфигурирования кластеров под управлением YDB версии ниже v25.1. Начиная с версии v25.1, YDB поддерживает конфигурацию V2, которая является рекомендуемым способом конфигурирования для кластеров YDB версии v25.1 и выше. Для кластеров, использующих конфигурацию V2, данная инструкция **не применима**.

Если ваш кластер уже обновлён до версии v25.1 или выше, но ещё использует конфигурацию V1, рекомендуется запланировать миграцию на конфигурацию V2, так как поддержка конфигурации V1 будет прекращена в будущих версиях YDB.

Если нужно вывести из эксплуатации хост кластера YDB, на котором располагается часть статической группы, необходимо переместить её на другой хост.

Важно

Неправильная последовательность действий или ошибка в конфигурации 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 Embedded 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

> **Важно**
>
> Данная статья описывает работу с кластерами YDB, использующими конфигурацию V1. Конфигурация V1 является основным способом конфигурирования кластеров под управлением YDB версии ниже v25.1. Начиная с версии v25.1, YDB поддерживает конфигурацию V2, которая является рекомендуемым способом конфигурирования для кластеров YDB версии v25.1 и выше. Для кластеров, использующих конфигурацию V2, данная инструкция **не применима**.
>
> Если ваш кластер уже обновлён до версии v25.1 или выше, но ещё использует конфигурацию V1, рекомендуется запланировать миграцию на конфигурацию V2, так как поддержка конфигурации V1 будет прекращена в будущих версиях YDB.

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

Важно

Данная статья описывает работу с кластерами YDB, использующими конфигурацию V1. Конфигурация V1 является основным способом конфигурирования кластеров под управлением YDB версии ниже v25.1. Начиная с версии v25.1, YDB поддерживает конфигурацию V2, которая является рекомендуемым способом конфигурирования для кластеров YDB версии v25.1 и выше. Для кластеров, использующих конфигурацию V2, данная инструкция **не применима**.

Если ваш кластер уже обновлён до версии v25.1 или выше, но ещё использует конфигурацию V1, рекомендуется запланировать миграцию на конфигурацию V2, так как поддержка конфигурации V1 будет прекращена в будущих версиях YDB.

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

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. По техническим причинам в этом списке также должен присутствовать субъект доступа `root@builtin`. Пример:

   ```
   security_config:
     enforce_user_token_requirement: true
     ...
     register_dynamic_node_allowed_sids:
     - "root@builtin" # требуется по техническим причинам
     - "registerNode@cert"
   ```

   Более подробная информация по настройке параметров аутентификации кластера приведена в соответствующем разделе документации.

3. Обновите файлы статической конфигурации на всех узлах кластера вручную либо с помощью плейбука Ansible.
4. Выполните поэтапный перезапуск узлов хранения кластера с помощью ydbops либо с использованием плейбука Ansible.
5. Выполните поэтапный перезапуск узлов баз данных кластера с помощью ydbops либо с использованием плейбука Ansible.

Обзор конфигурации V2

Для развёртывания кластера YDB, добавления в кластер новых узлов и изменения параметров требуется конфигурация.

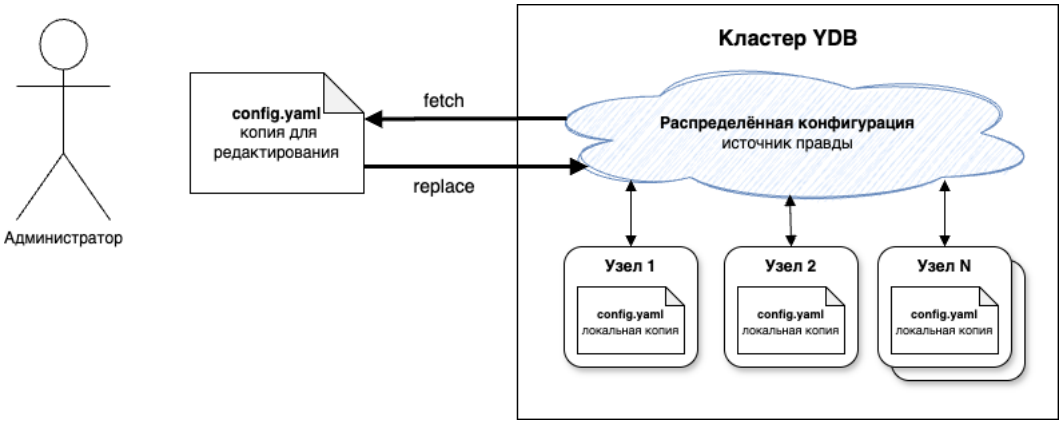
> **Совет**
>
> Новые кластеры YDB рекомендуется разворачивать сразу с использованием конфигурации V2. Если кластер был развёрнут на конфигурации V1, то она продолжит использоваться даже после обновления на версию YDB 25.1 или выше. После такого обновления рекомендуется запланировать и провести миграцию на V2, так как в будущих версиях YDB поддержка V1 будет прекращена. Узнать, какая версия конфигурации используется на кластере, можно по инструкции.

Конфигурация 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 Embedded 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.

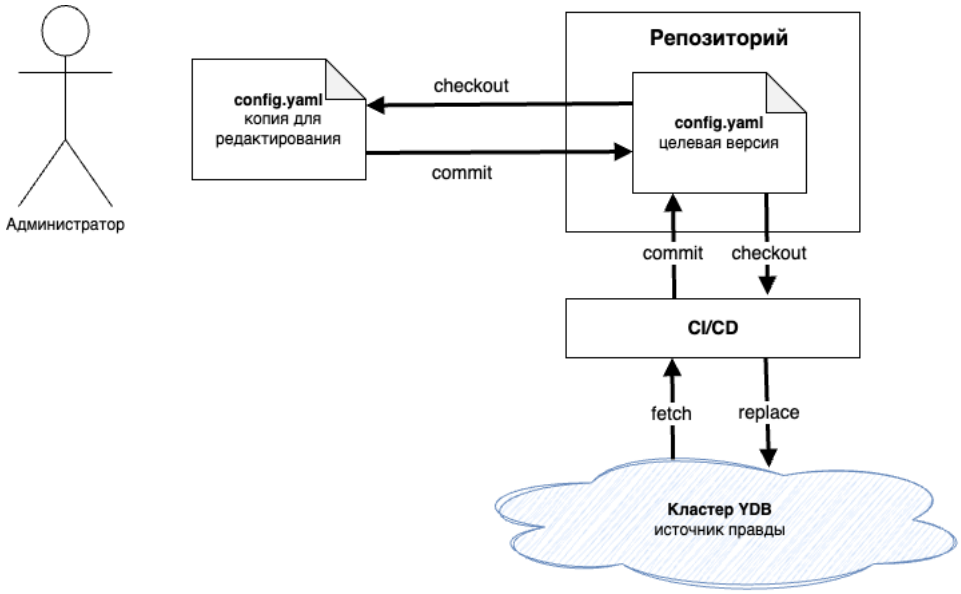
Пример изменения конфигурации

```
$ 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.

> **Примечание**
>
> Если ваш кластер развёрнут на более ранней версии YDB, чем v25.1 и/или использует конфигурацию V1, данной инструкцией пользоваться нельзя. Используйте раздел для конфигурации V1 или смигрируйте кластер на V2.

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 одновременно 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

В данной статье приведено описание основных групп конфигурируемых параметров в файле конфигурации кластера.

host_configs — типовые конфигурации хостов

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>
  - 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 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)

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
```

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 — статические узлы кластера

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 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.

log_config — конфигурация логирования

Секция `log_config` управляет тем, как YDB обрабатывает и управляет своими логами. Она позволяет настраивать уровни логирования, форматы и места назначения для различных компонентов.

Подробную информацию о параметрах конфигурации логирования смотрите в Секция конфигурации `log_config`.

Топология кластера

Для конфигурации топологии кластера YDB доступны следующие режимы отказоустойчивости:

Режим	Описание
`none`	Избыточность отсутствует. Применяется для тестирования.
`block-4-2`	Избыточность с коэффициентом 1,5, применяется для однодатацентровых кластеров.
`mirror-3-dc`	Избыточность с коэффициентом 3, применяется для мультидатацентровых кластеров.

Топология кластера задается в качестве параметра `erasure` и принимает одно из значений, указанных в таблице выше.

Конфигурация Blob Storage

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

```
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

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 учетом 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 3 работающих узла (из 5) остается при сбое двух доменов.
Геораспределенный	9	Выберите три узла в разных доменах отказа внутри каждой из трех зон доступности пула хранения mirror-3-dc для гарантии, что большинство в 5 работающих узлов (из 9) остается при сбое зоны доступности + домена отказа.

При развертывании 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 нечетные числа, так как использование четных чисел не улучшает отказоустойчивость по сравнению с ближайшим меньшим нечетным числом.

domains_config — домен кластера

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.
Георасределенный	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.

Примеры

`block-4-2`

```
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
```

`mirror-3-dc`

```
domains_config:
  domain:
  - name: global
    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
```

`none` (без отказоустойчивости)

```
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 пулов

```
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
```

Настройка акторной системы

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 Embedded UI.

Автоматическое конфигурирование

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:

```yaml
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.

Контроллер памяти

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. 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.
- KQP: хранит промежуточные результаты обработки запросов.
- 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:

- KQP.

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.

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 KQP:

```
memory_controller_config:
  query_execution_limit_percent: 25
```

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 KQP.

blob_storage_config — статическая группа кластера

Укажите конфигурацию статической группы кластера. Статическая группа необходима для работы базовых таблеток кластера, в том числе `Hive`, `SchemeShard`, `BlobstorageContoller`.
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 нодах, 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 AZ необходимо указать 3 кольца. Для конфигураций, расположенных в одной AZ, указывается ровно одно кольцо.

Настройка стабильных имен узлов кластера

Присвоение имен узлам осуществляет Node Broker — системная таблетка, которая отвечает за регистрацию динамических узлов в кластере YDB.

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, необходимо добавить в конфигурацию кластера следующее:

```
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 — брокер ресурсов

Брокер ресурсов — это 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`	2	—	Операции восстановления из резервной копии.
`queue_build_index`	10	—	Операции онлайн-создания вторичного индекса.
`queue_cdc_initial_scan`	4	—	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
```

Настройка Health Check

В этом разделе настраиваются пороговые значения и таймауты, используемые сервисом Health Check YDB. Эти параметры помогают настраивать возможные проблемы, такие как чрезмерные перезапуски или расхождение по времени между динамическими узлами.

Синтаксис

```
healthcheck_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)

Примеры конфигураций кластеров

В репозитории можно найти модельные примеры конфигураций кластеров для самостоятельного развертывания. Ознакомьтесь с ними 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 во встроенном 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 во встроенном 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 во встроенном 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.

Перемещение State Storage

Статья находится в разработке

> **Примечание**
>
> Если ваш кластер развёрнут на более ранней версии YDB, чем v25.1 и/или использует конфигурацию V1, данной инструкцией пользоваться нельзя. Используйте раздел для конфигурации V1 или смигрируйте кластер на V2.

Если нужно вывести из эксплуатации узел кластера YDB, на котором располагается часть State Storage, необходимо переместить её на другой узел.

> **Важно**
>
> Неправильная последовательность действий или ошибка в конфигурации 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

Перемещение статической группы

Статья находится в разработке

> **Примечание**
>
> Если ваш кластер развёрнут на более ранней версии YDB, чем v25.1 и/или использует конфигурацию V1, данной инструкцией пользоваться нельзя. Используйте раздел для конфигурации V1 или смигрируйте кластер на V2.

При использовании Конфигурации V2 управление статической группой осуществляется автоматически и механизм Self Heal выполнит реконфигурацию при выходе из строя одного узла статической группы.

При необходимости ручного управления конфигурацией статической группы, необходимо выключить автоматическое урпавление статисческой группой, получить текущую конфигурацию статической группы, внести изменения и применить измененную конфигурацию в качестве целевой конфигурации статической группы. Затем необходимо убрать целевую конфигурацию статической группы из файла конфигурации и включить автоматическое управление конфигурацией статической группы.

> **Важно**
>
> Неправильная последовательность действий или ошибка в конфигурации 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` мы используем 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. Выключить автоматическое урпавление статисческой группой
2. Получить текущую конфигурацию статической группы
3. Внести изменения и применить 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 Embedded UI и убедитесь, что VDisk статической группы появился на целевом физическом диске и реплицируется. Подробнее см. Мониторинг статической группы.
5. Убрать целевую конфигурацию статической группы из файла конфигурации и включить автоматическое управление конфигурацией статической группы.

Замена FQDN узла

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

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. По техническим причинам в этом списке также должен присутствовать субъект доступа `root@builtin`. Пример:

   ```
   security_config:
     enforce_user_token_requirement: true
     ...
     register_dynamic_node_allowed_sids:
     - "root@builtin" # требуется по техническим причинам
     - "registerNode@cert"
   ```

   Более подробная информация по настройке параметров аутентификации кластера приведена в соответствующем разделе документации.

3. Обновите файлы статической конфигурации на всех узлах кластера вручную либо с помощью плейбука Ansible.
4. Выполните поэтапный перезапуск узлов хранения кластера с помощью ydbops либо с использованием плейбука Ansible.
5. Выполните поэтапный перезапуск узлов баз данных кластера с помощью ydbops либо с использованием плейбука Ansible.

Миграция на конфигурацию V2

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.
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. Добавить в полученный на шаге 1 или 2 файл `config.yaml` следующее поле:

   ```
   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.

4. Разместить файл `config.yaml` на всех узлах кластера, заменив им предыдущий файл конфигурации.

5. 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.

6. 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
```

7. Загрузить полученный ранее конфигурационный файл `config.yaml` в систему с помощью команды ydb admin cluster config replace:

```
ydb -e grpc://<node.ydb.tech>:2135 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.

8. 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.

9. 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.

10. Загрузить обновлённый конфигурационный файл в кластер с помощью ydb admin cluster config replace:

```
ydb -e grpc://<node.ydb.tech>:2135 cluster config replace -f config.yaml
```

11. Перезапустить все узлы хранения кластера с помощью процедуры rolling restart.
12. При наличии секции `config.domains_config.security_config` в файле `config.yaml` вынести её на уровень выше — в секцию `config`.
13. Удалить из файла `config.yaml` секции `config.blob_storage_config` и `config.domains_config`.
14. Загрузить обновлённый конфигурационный файл в кластер:

    ```
    ydb -e grpc://<node.ydb.tech>:2135 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 --for-v1-migration > config.yaml
   ```

   Подробнее

   Аргумент `--for-v1-migration` 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.

Работа с YDB с помощью Ansible

Этот раздел документации YDB содержит набор статей, предназначенных для работы DevOps-инженеров с кластерами 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

Работа с YDB в Kubernetes

Этот раздел документации YDB содержит коллекцию статей, предназначенных для DevOps-инженеров, работающих с кластерами 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

Обзор управления кластером YDB вручную

В этом 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 встроенного web-интерфейса

Развёртывание YDB кластера с помощью Ansible

В инструкции изложен процесс развёртывания YDB кластера на группе серверов с помощью Ansible. Рекомендуемая схема разворачивания кластера YDB для начала работы состоит из 3 серверов с 3 дисками для пользовательских дисков в каждом. Для обеспечения отказоустойчивости этим серверам желательно иметь независимую инфраструктуру: располагаться в отдельных датацентрах или зонах доступностях, либо хотя бы отдельных серверных стойках.

Для более масштабных кластеров, рекомендуется использовать от 9 серверов для отказоустойчивых кластеров (`mirror-3-dc`) или 8 серверов для однодатацентровых кластеров (`block-4-2`). В обоих этих случаях серверам достаточно иметь по одному диску для пользовательских данных, но желательно также иметь отдельный небольшой диск для операционной системы. Подробнее о моделях избыточности можно узнать из статьи Топология кластера YDB. В процессе эксплуатации кластер может быть расширен без приостановки доступа пользователей к базам данных.

Примечание

Рекомендуемые требования к серверам:

- 16 CPU (рассчитывается исходя из утилизации 8 CPU сторадж нодой и 8 CPU динамической нодой).
- 16 GB RAM (рекомендуемый минимальный объем RAM).
- SSD-диски для пользовательских данных, размером от 120 GB.
- Доступ по SSH.
- Сетевая связность машин в кластере.
- OS: Ubuntu 18+, Debian 9+.
- Доступ в интернет для обновления репозиториев и скачивания нужных пакетов.

Скачать репозиторий с примерами для установки YDB на кластер можно с GitHub – `git clone https://github.com/ydb-platform/ydb-ansible-examples.git`. Этот репозиторий содержит: шаблоны установки YDB на кластер в поддиректориях, а также скрипты для генерации TLS-сертификатов и файлы requirements для установки необходимых Python пакетов. В этой статье будет использоваться поддиректория `3-nodes-mirror-3-dc` для самого простого способа развёртывания. Также можно аналогичным образом использовать директории `8-nodes-block-4-2` или `9-nodes-mirror-3-dc`, если вам доступно необходимое количество подходящих серверов.

Структура репозитория

```
├── 3-nodes-mirror-3-dc / 9-nodes-mirror-3-dc / 8-nodes-block-4-2
│   ├── ansible.cfg #Конфигурационный файл Ansible, который содержит настройки подключения к серверам и опции
структуры проекта.
│   ├── ansible_vault_password_file #Пароль для пользователя root.
│   ├── creds #Переменные окружения, задающие имя пользователя и пароль для YDB.
│   ├── files
│   │   ├── config.yaml #Конфигурационный файл YDB.
│   ├── inventory #Директория с инвентаризационными файлами.
│   │   ├── 50-inventory.yaml #Основной инвентаризационный файл.
│   │   └── 99-inventory-vault.yaml #Зашифрованный инвентаризационный файл, содержащий пароль root пользователя от
YDB.
├── README.md #Описание репозитория.
├── requirements.txt #Файл со списком зависимостей для установки Python пакетов в виртуальное окружение.
├── requirements.yaml #Файл с указателем на актуальные Ansible коллекции.
├── TLS #Директория, в которой создаётся поддиректория CA с TLS сертификатами.
│   ├── ydb-ca-nodes.txt #Список FQDN серверов для генерации TLS сертификатов
│   └── ydb-ca-update.sh #Скрипт для генерации TLS сертификатов из списка ydb-ca-nodes.txt
```

Для работы с проектом на локальной (промежуточной или инсталляционной) машине понадобится: Python 3 версии 3.10+ и Ansible core не ниже версии 2.15.2. Ansible можно установить и запустить глобально (устанавливается в систему) или в виртуальном окружении. Если Ansible уже установлен – можно переходить к шагу «Настройка Ansible проекта», если Ansible ещё не установлен, установите его одним из предложенных способов:

Установка Ansible в систему (глобально, Ubuntu 22.04 LTS)

- Обновите список пакетов apt репозитория – `sudo apt update`.
- Обновите пакеты – `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` (имейте ввиду, что установка просто `ansible` приведёт к неподходящей устаревшей версии).
- Проверьте версию Ansible core – `ansible --version`

Установка Ansible в виртуальное окружение Python

- Обновите список пакетов apt репозитория – `sudo apt-get update`.
- Установите пакет `venv` для Python3 – `sudo apt-get install python3-venv`
- Создайте директорию, где будет создано виртуальное окружение и куда будут скачаны плейбуки. Например, `mkdir venv-ansible`.
- Создайте виртуальное окружение – `python3 -m venv venv-ansible`.
- Активируйте виртуальное окружение – `source venv-ansible/bin/activate`. В дальнейшем все действия по работе с Ansible выполняются внутри виртуального окружения. Выйти из него можно командой `deactivate`.
- Установите Ansible рекомендуемой версии с помощью команды `pip3 install -r requirements.txt`, находясь в корневой директории скачанного репозитория.
- Проверьте версию Ansible core – `ansible --version`

Перейдите в корневой каталог загруженного репозитория и выполните команду `ansible-galaxy install -r requirements.yaml` – она скачает коллекции Ansible `ydb_platform.ydb` и `community.general`, которые содержат роли и плагины для установки YDB.

Настройка проекта Ansible

Редактирование инвентори-файлов

Независимо от выбранной топологии кластера (`3-nodes-mirror-3-dc`, `9-nodes-mirror-3-dc` или `8-nodes-block-4-2`), основные параметры для установки и настройки YDB содержатся в инвентори-файле `50-inventory.yaml`, который находится в каталоге `inventory/`.

В инвентори-файле `50-inventory.yaml` необходимо указать текущий список FQDN серверов, на которых будет установлен YDB. По умолчанию список выглядит следующим образом:

```
all:
  children:
    ydb:
      static-node-1.ydb-cluster.com:
      static-node-2.ydb-cluster.com:
      static-node-3.ydb-cluster.com:
```

Далее необходимо внести следующие изменения в разделе `vars` инвентори-файла:

- `ansible_user` – укажите пользователя, от имени которого Ansible будет подключаться через SSH.
- `ansible_ssh_common_args: "-o ProxyJump=<ansible_user>@<static-node-1-IP>"` – опция для подключения Ansible к серверу по IP, с которого будет установлен YDB (включая сервер ProxyJump). Используется при установке YDB с локальной машины, не входящей в частную DNS-зону.
- `ansible_ssh_private_key_file` – измените путь до приватного SSH-ключа на фактическое: `"../<ssh-private-key-name>"`.
- Выберите один из доступных вариантов развёртывания исполняемых файлов YDB:
  - `ydb_version`: автоматически загрузить один из официальных релизов YDB по номеру версии. Например, `23.4.11`.
  - `ydb_git_version`: автоматически скомпилировать исполняемые файлы YDB из исходного кода, загруженного из официального репозитория GitHub. Значение настройки – это имя ветки, тега или коммита. Например, `main`.
  - `ydb_archive`: локальный путь к архиву с дистрибутивом YDB, загруженного или подготовленного заранее.
  - `ydbd_binary` и `ydb_cli_binary`: локальные пути к исполняемым файлам сервера и клиента YDB, загруженным или подготовленным заранее.

Установка fq-connector-go

Для запуска федеративных запросов может потребоваться установка коннектора. Плейбук поддерживает развертывание коннектора fq-connector-go на хостах с динамическими нодами. Используйте следующие настройки:

- `ydb_install_fq_connector` - установите в `true`, если необходимо установить коннектор.
- 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 к архиву с дистрибутивом fq-connector-go, загруженному или подготовленному заранее.
  - `ydb_fq_connector_binary`: локальный путь к исполняемому файлу fq-connector-go, загруженному или подготовленному заранее.

Дополнительные изменения в инвентори-файлах

При необходимости можно изменить эти настройки, но это не обязательно в простых случаях:

- `ydb_cores_static` – количество процессорных ядер, выделенных для статических узлов.
- `ydb_cores_dynamic` – количество процессорных ядер, выделенных для динамических узлов.

- `ydb_tls_dir` – укажите локальный путь к папке с заранее подготовленными TLS-сертификатами. Она должна содержать файл `ca.crt` и подкаталоги с именами, соответствующими именам узлов, содержащие сертификаты для каждого узла. Если параметр не указан, самоподписанные TLS-сертификаты будут сгенерированы автоматически для всего кластера YDB.
- `ydb_brokers` – укажите список FQDN брокерных узлов. 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. При условии наличия только одной базы данных в кластере используйте следующую логику:

- Для промышленного развёртывания используйте диски ёмкостью более 800 ГБ с высокой производительностью IOPS, затем выберите значение для этого параметра на основе топологии кластера:
  - Для `block-4-2` установите `ydb_database_groups` на уровне 95% от общего количества дисков, округляя вниз.
  - Для `mirror-3-dc` установите `ydb_database_groups` на уровне 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 YDB кластер. По умолчанию в `system_ntp_servers` указан набор NTP-серверов без учёта географического расположения инфраструктуры, на которой будет развёртываться YDB кластер. Мы настоятельно рекомендуем использовать локальный NTP-сервер для on-premise инфраструктуры и следующие NTP-серверы для облачных провайдеров:

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.

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 прочесть в данной статье.

Alibaba

- Специфика подключения к NTP-серверам в Alibaba описана в этой статье.

Изменения других секций конфигурационного файла `50-inventory.yaml` не требуются.

Изменение пароля пользователя root

Далее можно изменить стандартный пароль root пользователя YDB, который содержится в зашифрованном инвентаризационном файле `99-inventory-vault.yaml` и файле `ansible_vault_password_file.txt`. Для изменения пароля укажите новый пароль в файле `ansible_vault_password_file.txt` и продублируйте его в файле `99-inventory-vault.yaml` в формате:

```
all:
  children:
    ydb:
      vars:
        ydb_password: <new-password>
```

Для шифрования `99-inventory-vault.yaml` выполните команду `ansible-vault encrypt inventory/99-inventory-vault.yaml`.

После изменения инвентаризационных файлов можно переходить к подготовке конфигурационного файла YDB.

Подготовка конфигурационного файла YDB

Конфигурационный файл YDB содержит настройки нод YDB и располагается в поддиректории `/files/config.yaml`. С подробным описанием секций настройки конфигурационного файла YDB можно ознакомиться в статье Параметры конфигурации кластера.

Дефолтный конфигурационный файл YDB уже содержит почти все необходимые настройки для развёртывания кластера. Необходимо заменить стандартные FQDN хостов на актуальные FQDN 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.

Конфигурационный файл fq-connector-go

Конфигурационный файл fq-connector-go располагается в `/files/fq-connector-go/config.yaml` и в простых случаях не требует никаких изменений.

Развёртывание кластера YDB

> **Примечание**
>
> Минимальное количество серверов в кластере YDB – восемь серверов для модели избыточности `block-4-2` и девять серверов для модели избыточности `mirror-3-dc`.

Репозиторий содержит два готовых набора шаблонов для развёртывания кластера YDB из восьми (модель избыточности `block-4-2`) и девяти серверов (`mirror-3-dc`). Любой из предложенных вариантов может быть масштабирован на любое нужное количество серверов с учётом ряда технических требований.

Для подготовки собственного шаблона можно воспользоваться следующей инструкцией:

1. Создайте копию директории с готовым примером (`9-nodes-mirror-3-dc` или `8-nodes-block-4-2`).
2. Укажите FQDN серверов в файле `TLS/ydb-ca-nodes.txt` и выполните скрипт `ydb-ca-update.sh` для генерации наборов TLS сертификатов.
3. Внесите изменения в инвентаризационные файлы шаблона в соответствии с инструкцией.
4. Внесите изменения в конфигурационный файл YDB в соответствии с инструкцией.
5. Выполните команду `ansible-playbook ydb_platform.ydb.initial_setup`, находясь в директории клонированного шаблона.

План выполнения сценария установки YDB

Последовательность выполнения ролей и их краткое описание:

1. Роль `packages` настраивает репозитории, управляет предпочтениями и конфигурациями APT, а также исправляет неконфигурированные пакеты и устанавливает необходимые программные пакеты в зависимости от версии дистрибутива.
2. Роль `system` устанавливает системные настройки, включая конфигурацию часов и временной зоны, синхронизацию времени через NTP с помощью `systemd-timesyncd`, настройку `systemd-journald` для управления журналами, конфигурацию загрузки модулей ядра и оптимизацию параметров ядра через `sysctl`, а также настройку производительности процессора с использованием `cpufrequtils`.
3. Роль `ydb` выполняет задачи по проверке необходимых переменных, установке базовых компонентов и зависимостей, настройке системных пользователей и групп, развертыванию и конфигурированию YDB, включая управление сертификатами TLS и обновление конфигурационных файлов.
4. Роль `ydb_fq_connector` (опционально) отвечает за подготовку и запуск процессов fq-connector-go, включая проверку необходимых переменных, установку необходимых бинарных файлов, файлов конфигурации, создание и запуск `systemd unit`. Ознакомиться с `systemd unit` fq-connector-go можно по ссылке.
5. Роль `ydb-static` отвечает за подготовку и запуск статических нод YDB, включая проверку необходимых переменных и секретов, форматирование и подготовку дисков, создание и запуск `systemd unit` для узла хранения, а также инициализацию хранилища и управление доступом к базе данных. Ознакомиться с `systemd unit` сторадж ноды можно по ссылке.
6. Роль `ydb-dynamic` настраивает и управляет динамическими узлами YDB, включая проверку необходимых переменных, создание конфигурационных файлов и `systemd unit` файлов для каждого динамического узла, запуск этих узлов, получение токена для доступа к YDB, создание базы данных в YDB. Шаблон файла `systemd unit` динамической ноды находится по ссылке.

Подробное пошаговое описание установки YDB

1. Роль `packages`. Задачи:

- `check dpkg audit` – проверка состояния dpkg с помощью команды `dpkg --audit` и сохранение результатов команды в переменной `dpkg_audit_result`. Задача завершится с ошибкой, если команда `dpkg_audit_result.rc` вернет значение отличное от 0 или 1.
- `run the equivalent of "apt-get clean" as a separate step` – очистка кеша apt, аналогично команде `apt-get clean`.
- `run the equivalent of "apt-get update" as a separate step` – обновление кеша apt, аналогично команде `apt-get update`.
- `fix unconfigured packages` – исправление неконфигурированных пакетов с помощью команды `dpkg --configure --pending`.
- `set vars_for_distribution_version variables` – установка переменных для конкретной версии дистрибутива.
- `setup apt repositories` – настройка репозиториев apt из заданного списка.
- `setup apt preferences` – настройка предпочтений apt (содержание переменных указано в `roles/packages/vars/distributions/<distributive name>/<version>/main.yaml`).
- `setup apt configs` – настройка конфигураций apt.
- `flush handlers` – принудительный запуск всех накопленных обработчиков (хендлеров). В данном случае запускается хендлер, который обновляет кеш apt.
- `install packages` – установка пакетов apt с учетом заданных параметров и времени валидности кеша.

Ссылки на списки пакетов, которые будут установлены для Ubuntu 22.04 или Astra Linux 1.7:

- Список пакетов для Ubuntu 22.04;
- Список пакетов для Astra Linux 1.7.

1. Роль `system`. Задачи:

- `configure clock` – блок задач настройки системных часов:
  - `assert required variables are defined` – проверка переменной `system_timezone` на существование. Эта проверка гарантирует, что необходимая переменная доступна для следующей задачи в блоке.
  - `set system timezone` – установка системного часового пояса. Часовой пояс задаётся значением переменной `system_timezone`, а аппаратные часы (`hwclock`) устанавливаются на UTC. После выполнения задачи отправляется уведомление для перезапуска службы `cron`.
  - `flush handlers` – принудительное выполнение накопленных обработчиков директивой `meta`. Будет произведен рестарт следующих процессов: `timesyncd`, `journald`, `cron`, `cpufrequtils`, и выполнена команда `sysctl -p`.
- `configure systemd-timesyncd` – блок задач настройки `systemd-timesyncd`:
  - `assert required variables are defined` - утверждает, что количество NTP серверов (`system_ntp_servers`) больше одного, если переменная `system_ntp_servers` определена. Если переменная `system_ntp_servers` не определена, выполнение блока задач `configure systemd-timesyncd` будет пропущено, включая проверку количества NTP серверов и настройку `systemd-timesyncd`.
  - `create conf.d directory for timesyncd` - создаёт директорию `/etc/systemd/timesyncd.conf.d`, если переменная `system_ntp_servers` определена.
  - `configure systemd-timesyncd` - создаёт конфигурационный файл `/etc/systemd/timesyncd.conf.d/ydb.conf` для службы `systemd-timesyncd` с основным и резервными NTP серверами. Задача будет выполнена, если переменная `system_ntp_servers` определена. После выполнения задачи отправляется уведомление для перезапуска службы `timesyncd`.
  - `flush handlers` - вызываются накопленные обработчики. Выполняется обработчик `restart timesyncd`, который выполняет перезапуск сервиса `systemd-timesyncd.service`.
  - `start timesyncd` - запуск и активация службы `systemd-timesyncd.service`. Далее служба будет стартовать автоматически при загрузке системы.
- `configure systemd-journald` – блок задач по настройке сервиса `systemd-journald`:
  - `create conf.d directory for journald` - создание директории `/etc/systemd/journald.conf.d` для хранения конфигурационных файлов `systemd-journald`.
  - `configure systemd-journald` - создание конфигурационного файла в `/etc/systemd/journald.conf.d/ydb.conf` для `systemd-journald`, в котором указывается секция `Journal` с опцией `ForwardToWall=no`. Параметр `ForwardToWall=no` в конфигурации `systemd-journald` означает, что журнал сообщений (логи) системы не будет перенаправляться как сообщения "wall" всем вошедшим в систему пользователям. После выполнения задачи отправляется уведомление для перезапуска службы `journald`.
  - `flush handlers` - вызывает накопленные хендлеры. Будет выполнен хендлер `restart journald`, который перезапустит сервис `systemd-journald`.
  - `start journald` - запуск и активация службы `systemd-journald.service`. Далее служба будет стартовать автоматически при загрузке системы.
- `configure kernel` – блок задач конфигурирования ядра:
  - `configure /etc/modules-load.d dir` - создание директории `/etc/modules-load.d` с правами владельца и группы для пользователя root и разрешениями `0755`.
  - `setup conntrack module` - копирование строки `nf_conntrack` в файл `/etc/modules-load.d/conntrack.conf` для загрузки модуля `nf_conntrack` при старте системы.
  - `load conntrack module` - загрузка модуля `nf_conntrack` в текущей сессии.

    - `setup sysctl files` - применяет шаблоны для создания конфигурационных файлов в `/etc/sysctl.d/` для различных системных настроек (таких, как настройки безопасности, сети и файловой системы). Список файлов включает `10-console-messages.conf`, `10-link-restrictions.conf` и другие. После выполнения этой задачи отправляется уведомление для применения изменений в настройках ядра.
    - `flush handlers` - вызывает накопленные хендлеры. Будет вызван хендлер `apply kernel settings`, который выполнит команду `sysctl -p` для применения параметров ядра, указанных в файле `/etc/sysctl.conf` или в других файлах в каталоге `/etc/sysctl.d/`.
- `configure cpu governor` – блок задач настройки режима управления частотой процессора:
    - `install cpufrequtils` - установка пакета `cpufrequtils` из apt. В задаче установлены параметры проверки кеша apt и таймаут выполнения задачи в 300 секунд, чтобы ускорить выполнения задачи и избежать бесконечного цикла ожидания обновления пакетов apt.
    - `use performance cpu governor` - создание файла `/etc/default/cpufrequtils` с содержимым "GOVERNOR=performance", что устанавливает режим работы CPU governor в "performance" (Отключения режима энергосбережения при простое ядер CPU). После выполнения задачи отправляется уведомление для перезапуска службы `cpufrequtils`.
    - `disable ondemand.service` - отключение сервиса `ondemand.service`, если он присутствует в системе. Сервис останавливается, его автоматический запуск отключается, и он маскируется (предотвращается его запуск). После выполнения задачи отправляется уведомление для перезапуска cpufrequtils.
    - `flush handlers` - вызывает накопленные хендлеры. Будет вызван хендлер `restart cpufrequtils`, который перезапустит сервис `cpufrequtils`.
    - `start cpufrequtils` - запуск и активация службы `cpufrequtils.service`. Далее служба будет стартовать автоматически при загрузке системы.

1. Роль `ydbd`. Задачи:

- `check if required variables are defined` – проверка, что переменные `ydb_archive`, `ydb_config`, `ydb_tls_dir` определены. Если какая-либо из них не определена, Ansible выведет соответствующее сообщение об ошибке и остановит выполнение плейбука.
- `set vars_for_distribution variables` – установка переменных из указанного файла в переменной `vars_for_distribution_file` во время выполнения плейбука. Задача управляет набором переменных, зависящих от конкретного дистрибутива Linux.
- `ensure libaio is installed` – проверка, что пакет `libaio` установлен.
- `install custom libidn from archive` – установка пользовательской версии библиотеки `libidn` из архива.
- `create certs group` – создание системной группы `certs`.
- `create ydb group` – создание системной группы `ydb`.
- `create ydb user` – создание системного пользователя `ydb` с домашней директорией.
- `install YDB server binary package from archive` – установка YDB из скаченного архива.
- `create YDB audit directory` – создание поддиректории `audit` в директории установки YDB.
- `setup certificates` – блок задач по установки сертификатов безопасности:
    - `create YDB certs directory` – создание поддиректории `certs` в директории установки YDB.
    - `copy the TLS ca.crt` – копирование корневого сертификата `ca.crt` на сервер.
    - `copy the TLS node.crt` – копирование TLS-сертификата `node.crt` из директории сгенерированных сертификатов.
    - `copy the TLS node.key` – копирование TLS-сертификата `node.key` из директории сгенерированных сертификатов.
    - `copy the TLS web.pem` – копирование TLS pem ключа `web.pem` из директории сгенерированных сертификатов.
- `copy configuration file` – копирование конфигурационного файла `config.yaml` на сервер.
- `add configuration file updater script` – копирование скрипта `update_config_file.sh` на сервер.

1. Роль `ydbd_static`. Задачи:

- `check if required variables are defined` – проверка, что переменные `ydb_cores_static`, `ydb_disks`, `ydb_domain`, `ydb_user` определены. Если хотя бы одна из этих переменных не определена, задача завершится с ошибкой, и будет выведено соответствующее сообщение об ошибке для каждой переменной, которая не была определена.
- `check if required secrets are defined` – проверка определения секретной переменной `ydb_password`. Если эта переменная не определена, задача завершится с ошибкой, и будет выведено сообщение об ошибке.
- `create static node configuration file` – создание конфигурационного файла статической ноды путём запуска скопированного скрипта `update_config_file.sh` и передачи в него конфигураций `ydbd-config.yaml`, `ydbd-config-static.yaml`.
- `create static node systemd unit` – создание файла `ydbd-storage.service` для статического узла на основе шаблона. После выполнения задачи отправляется уведомление для перезапуска службы `systemd`.
- `flush handlers` – выполнение накопленных хендлеров. Выполняется перезапуск всех служб `systemd`.
- `format drives confirmation block` – блок задач по форматированию дисков и прерывания выполнения плейбука в случае отказа пользователя от подтверждения. В терминал будет выведен запрос на подтверждение форматирования подключенного к серверу диска. Варианты ответа: `yes` – продолжить выполнение плейбука с форматированием диска. Любое другое значение, будет восприниматься как отказ от форматирования. По умолчанию диски форматируется автоматически без запроса разрешения у пользователя, так как переменные `ydb_allow_format_drives` и `ydb_skip_data_loss_confirmation_prompt` равны `true`. Если нужно запросить подтверждение от пользователя, то нужно изменить значение переменной `ydb_skip_data_loss_confirmation_prompt` на `false` в инвентаризационном файле `50-inventory.yaml`.

- `prepare drives` – задача по форматированию подключенных дисков. Вызывается плагин `drive_prepare` – это специально разработанный Ansible-модуль для установки YDB, который входит в состав поставки коллекции YDB и располагается в директории `.../.ansible/collections/ansible_collections/ydb_platform/ydb/plugins/action/drive_prepare.py`. Модуль выполнит форматирование подключенного к серверу диска, указанного в переменной `ydb_disks`. Форматирование будет произведено если переменная `ydb_allow_format_drives` имеет значение `true`.
- `start storage node` – запуск процесса сторадж ноды с помощью `systemd`. В случае возникновения любых ошибок запуска сервиса исполнение плейбука будет прервано.
- `get ydb token` – запрос токена YDB для выполнения команды инициализации сторадж. Токен сохраняется в переменной `ydb_credentials`. В задаче вызывается модуль `get_token` из директории `.../.ansible/collections/ansible_collections/ydb_platform/ydb/plugins/modules`. В случае возникновение любых ошибок на данном шаге выполнение плейбука будет прервано.
- `wait for ydb discovery to start working locally` – вызывается модуль `wait_discovery`, который выполняет запрос `ListEndpoints` к YDB для проверки работоспособности базовых подсистем кластера. Если подсистемы работают исправно – можно выполнять команды инициализации стороджа и создания базы данных.
- `init YDB storage if not initialized` – инициализация хранилища в случае если оно еще не создано. В задаче вызывается плагин `init_storage`, который выполняет команду инициализации хранилища с помощью grpcs-запроса к статической ноде на порт 2135. Результат выполнения команды сохраняется в переменной `init_storage`.
- `wait for ydb healthcheck switch to "GOOD" status` – ожидание получения статуса `GOOD` от системы проверки состояния YDB. В задаче вызывается плагин `wait_healthcheck`, который выполняет команду проверке состояния YDB.
- `set cluster root password` – установка пароля для root пользователя YDB. В задаче выполняется плагин `set_user_password`, который выполняет grpcs запрос к YDB и устанавливает заранее заданный пароль для root пользователя YDB. Пароль задаётся переменной `ydb_password` в инвентаризационном файле `/examples/9-nodes-mirror-3-dc/inventory/99-inventory-vault.yaml` в зашифрованном виде.

1. Роль `ydbd_dynamic`. Задачи:

- `check if required variables are defined` – проверка наличия установленных переменных (`ydb_domain`, `ydb_pool_kind`, `ydb_cores_dynamic`, `ydb_brokers`, `ydb_dbname`, `ydb_dynnodes`) и вывод ошибки в случае отсутствия любой из переменных.
- `create dynamic node configuration file` – создание конфигурационного файла для динамических нод.
- `create dynamic node systemd unit` – создание сервиса для systemd динамических нод. После выполнения задачи отправляется уведомление для перезапуска службы `systemd`.
- `flush handlers` – выполнение накопившихся хендлеров. Будет выполнен рестарт `systemd`.
- `start dynamic nodes` – запуск процесса динамических нод с помощью `systemd`.
- `get ydb token` – получение токена для создания базы данных.
- `create YDB database` – создание базы данных. В задаче выполняется плагин `create_database`, который выполняет запрос создания БД к YDB.
- `wait for ydb discovery to start working locally` – повторно вызывается модуль `wait_discovery` для проверки работоспособности базовых подсистем кластера.

Подключение к кластеру

После развёртывания кластера YDB, каждый сервер по умолчанию будет принимать соединения на следующих портах:

Порты статического узла

На каждом сервере кластера запускается по одному статическому узлу. В systemd он называется `ydbd-storage`. Статический узел принимает входящие соединения на следующих портах:

Порт	Описание
2135	Порт для основного API YDB, основанного на защищённом gRPC соединении поверх TLS.
19001	Порт для интерконнекта акторной системы YDB.
8765	HTTPS-порт для встроенного интерфейса, отображения метрик и других вспомогательных команд.
5432	Порт для режима совместимости с PostgreSQL.

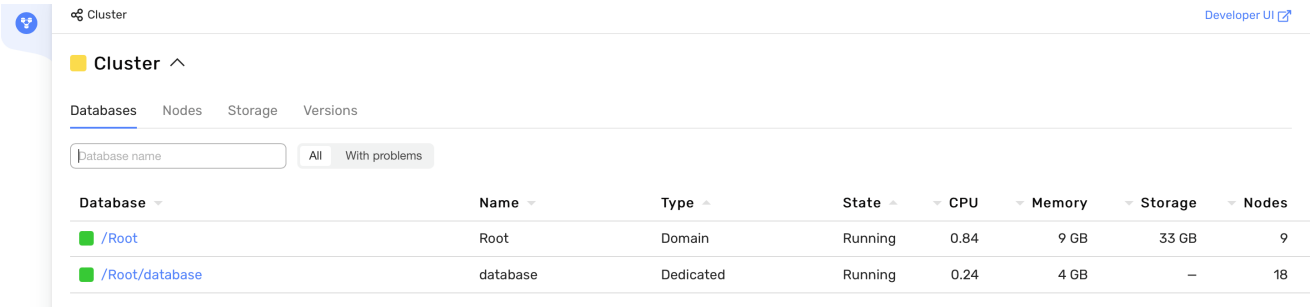
Порты динамических узлов

По умолчанию при развёртывании кластера через Ansible на каждом сервере запускается по два динамических узла. В systemd они называются `ydbd-database-a` и `ydbd-database-b`. Они принимают входящие соединения на следующих портах:

Название динамического узла	Порт	Описание

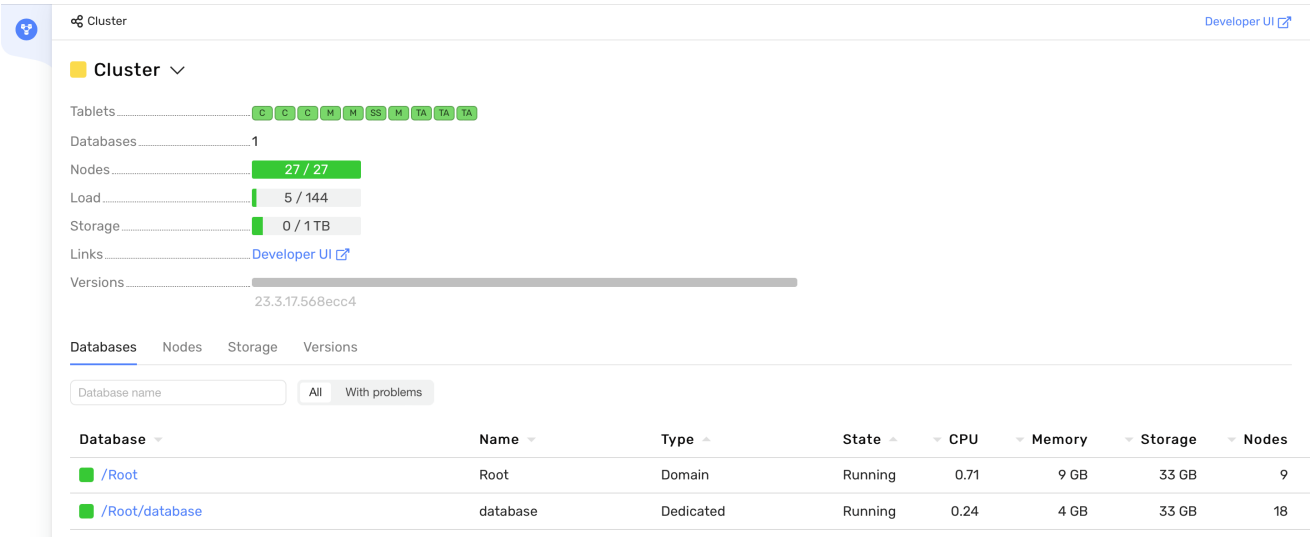
`ydbd-database-a`	2136	Порты для основного API YDB, основанного на защищённом gRPC соединении поверх TLS.
`ydbd-database-b`	2137	
`ydbd-database-a`	19002	Порты для интерконнекта акторной системы YDB.
`ydbd-database-b`	19003	
`ydbd-database-a`	8766	HTTPS-порты для встроенного интерфейса, отображения метрик и других вспомогательных команд.
`ydbd-database-b`	8767	

В результате выполнения плейбука будет создан кластер YDB, на котором развёрнута база данных под названием `database`, а также пользователь с именем `root` и максимальными правами доступа. Работать с кластером можно с помощью YDB CLI, посылая API-вызовы на порт 2135, через Embedded UI, который работает на порту 8765, а также в режиме PostgreSQL-совместимости через порт 5432. Также для подключения к Embedded UI можно настроить SSH-туннелирование. Для этого на локальной машине выполните команду `ssh -L 8765:localhost:8765 -i <ssh private key> <user>@<first ydb static node ip>`. После успешного установления соединения можно перейти по URL localhost:8765:



Мониторинг состояния кластера

После успешного создания кластера YDB проверить его состояние можно с помощью Embedded UI – http://localhost:8765/monitoring/cluster/tenants:

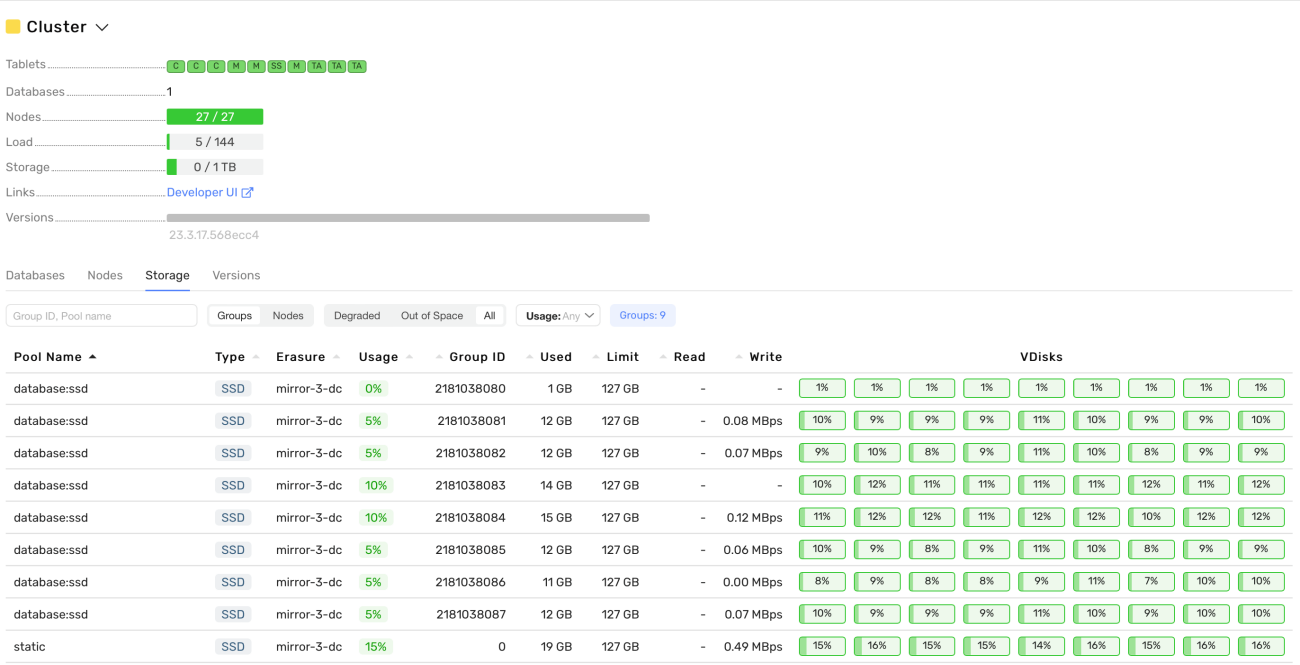


В разделе отражены следующие параметры кластера YDB, которые показывают состояние кластера:

- `Tablets` – список запущенных таблеток. Все индикаторы состояния таблеток должны быть зелёного цвета.

- `Nodes` – количество и состояние запущенных статических и динамических нод в кластере. Индикатор состояния нод должен быть зелёным, а пропорция созданных и запущенных нод должна быть равной. Например, 27/27 для кластера из девяти нод.
- Индикаторы показателей `Load` (количество используемой RAM) и `Storage` (количество используемого дискового пространства) должны быть зелёными.

Проверить состояние сторадж группы можно в разделе `storage` – http://localhost:8765/monitoring/cluster/storage:



Индикаторы `VDisks` должны быть зелёными, а статус `state` (находится в всплывающей подсказке при наведении на индикатор VDisk) должен быть `Ok`. Подробнее о показателях состояния кластера и мониторинге можно прочитать в статье YDB Monitoring.

Тестирование кластера

Протестировать кластер можно с помощью встроенных в YDB CLI нагрузочных тестов. Для этого скачайте на машину, на которой установлен Ansible, YDB CLI версии 2.5.0. Например, с помощью wget: `wget https://storage.yandexcloud.net/yandexcloud-ydb/release/2.5.0/linux/amd64/ydb`.

Сделайте скачанный бинарный файл исполняемым – `chmod +x ydb` и создайте профиль подключения к YDB:

```
./ydb \
  config profile create <profile name> \
  -d /Root/database \
  -e grpcs://<FQDN node>:2135 \
  --ca-file <path to generated certs>/CA/certs/ca.crt \
  --user root \
  --password-file <path to vault password file>/ansible_vault_password_file
```

Параметры команды и их значения:

- `config profile create` – команда создания профиля подключения. Задаётся имя профиля. Более детальную информацию о том, как создавать и изменять профили, можно найти в статье Создание и изменение профиля.
- `-e` – эндпоинт (endpoint), строка в формате `protocol://host:port`. Можно указать FQDN любой ноды кластера и не указывать порт. По умолчанию будет использован 2135 порт.
- `--ca-file` – путь к корневому сертификату для подключения к базе по `grpcs`. Сертификат создаётся скриптом `ydb-ca-update.sh` в директории `TLS` и располагается по пути `TLS/CA/certs/` относительно корня репозитория `ydb-ansible-examples`.
- `--user` – пользователь для подключения к БД. По умолчанию при выполнении плейбука `ydb_platform.ydb.initial_setup` создаётся пользователь root.
- `--password-file` – путь к файлу с паролем. В каждой папке с шаблоном развёртывания YDB кластера находится файл `ansible_vault_password_file`, который содержит пароль пользователя root.

Проверить, создался ли профиль, можно командой `./ydb config profile list` – будет выведен список профилей. После создания профиля его нужно активировать командой `./ydb config profile activate <profile name>`. Проверить, что профиль был активирован, можно повторным выполнением команды `./ydb config profile list` – активный профиль будет иметь отметку (active).

Теперь можно выполнить YQL запрос `./ydb yql -s 'select 1;'`, который вернёт в терминал результат выполнения команды `select 1` в табличной форме. После проверки соединения можно создать тестовую таблицу командой: `./ydb workload kv init --init-upserts 1000 --cols 4`. Будет создана тестовая таблица `kv_test`, состоящая из 4 столбцов и 1000 строк. Проверить, что таблица `kv_test` создалась и заполнилась тестовыми данными, можно командой `./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 5 видов нагрузок (`upsert` , `insert` , `select` , `read-rows` , `mixed`) и параметры их выполнения. Пример выполнения тестовой нагрузки `upsert` с параметром вывода времени выполнения запроса `--print-timestamp` и стандартными параметрами исполнения: `./ydb workload kv run upsert --print-timestamp`.

В терминал будет выведен отчёт следующего вида:

```
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
...
```

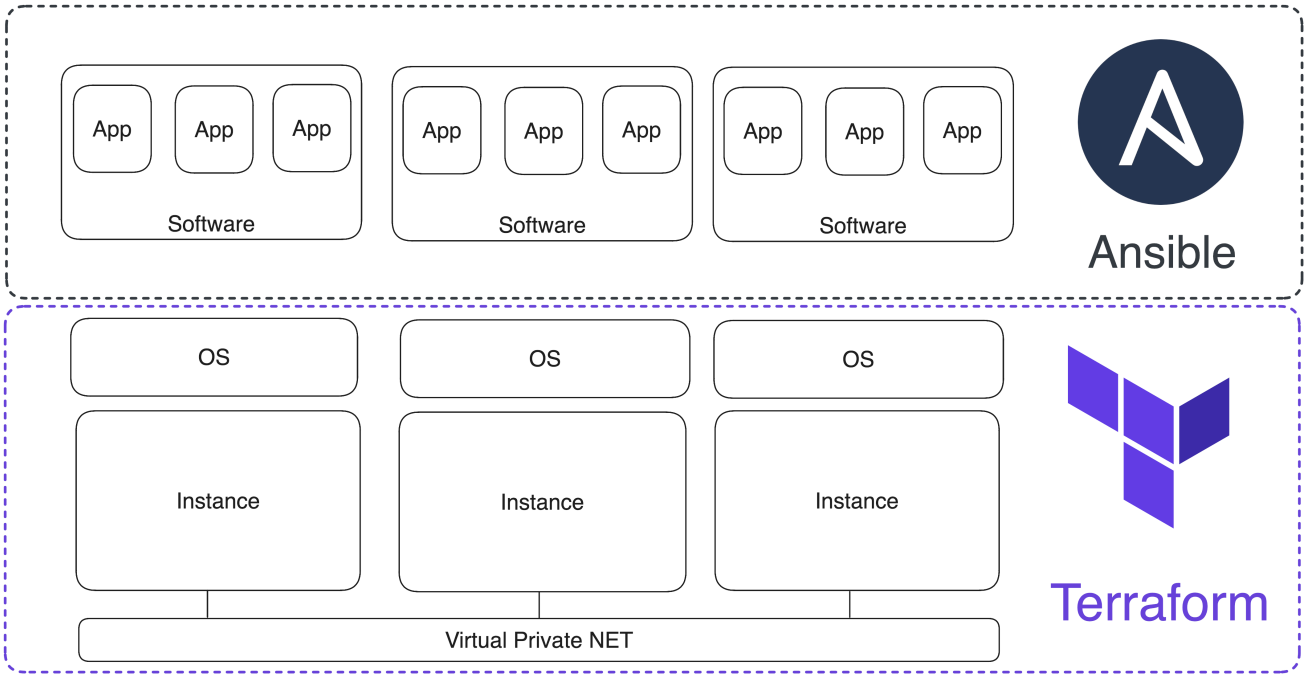
После завершения тестов удалить таблицу `kv_test` можно командой: `./ydb workload kv clean`. Подробнее об опциях создания тестовой таблицы и тестах можно прочесть в статье Key-Value нагрузка.

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 BM:



Конфигурация настройки окружения BM 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 (BM, 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:

   ```
   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.
5. Настроить Yandex Cloud Terraform провайдера.
6. Скачать данный репозиторий командой `git clone https://github.com/ydb-platform/ydb-terraform.git`.
7. 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
```

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     exited
   Configure Transparent Huge Pages (THP)
   ```

```
ydbd-database-a.service                                                  loaded active     running   YDB
dynamic node / database / a
ydbd-database-b.service                                                  loaded active     running   YDB
dynamic node / database / b
ydbd-storage.service                                                     loaded active     running   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.

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 or 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 даннух 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
...
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:22.4.44 --rm ydb-cli bash
   ```

3. Выполните запрос к базе данных YDB:

   ```
   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
   ```

Развёртывание YDB кластера вручную

Этот документ описывает способ развернуть мультитенантный кластер YDB на нескольких физических или виртуальных серверах.

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 Embedded UI.

При размещении нескольких динамических узлов на одном сервере потребуются отдельные порты для gRPC, Interconnect и HTTP интерфейса 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 требования к оборудованию описаны 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 ydb -g 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
```

Установите программное обеспечение YDB на каждом сервере

1. Скачайте и распакуйте архив с исполняемым файлом `ydbd` и необходимыми для работы YDB библиотеками:

   ```
   mkdir ydbd-stable-linux-amd64
   curl -L https://binaries.ydb.tech/ydbd-stable-linux-amd64.tar.gz | tar -xz --strip-component=1 -C ydbd-stable-linux-amd64
   ```

2. 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/
   ```

3. 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 или префикс, указывающий на тип устройства
- Последовательный идентификатор устройства (может быть буквой или числом)
- Последовательный идентификатор раздела на данном устройстве (обычно число)

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/nvme0n1
   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.

Подготовьте конфигурационные файлы

Подготовьте конфигурационный файл YDB:

1. Скачайте пример конфига для соответствующей модели отказа вашего кластера:
   - block-4-2 - для однодатацентрового кластера.
   - mirror-3dc - для cross-DC кластера из 9 нод.
   - mirror-3dc-3nodes - для cross-DC кластера из 3 нод.
2. В секции `host_configs` укажите все диски и их тип на каждой из нод кластера. Возможные варианты типов дисков:
   - ROT: rotational, HDD диски.
   - SSD: SSD или NVMe диски.

   ```
   host_configs:
   - drive:
     - path: /dev/disk/by-partlabel/ydb_disk_ssd_01
       type: SSD
     host_config_id: 1
   ```
3. В секции `hosts` укажите FQDN всех нод, их конфигурацию и расположение по датацентрам (`data_center`) и стойкам (`rack`):

   ```
   hosts:
   - host: node1.ydb.tech
     host_config_id: 1
     walle_location:
       body: 1
       data_center: 'zone-a'
       rack: '1'
   - host: node2.ydb.tech
     host_config_id: 1
     walle_location:
       body: 2
   ```

```
    data_center: 'zone-b'
    rack: '1'
- host: node3.ydb.tech
  host_config_id: 1
  walle_location:
    body: 3
    data_center: 'zone-c'
    rack: '1'
```

4. Включите аутентификацию пользователей (опционально).

   Если вы планируете использовать в кластере YDB возможности аутентификации и разграничения доступа пользователей, добавьте секцию `security_config` со следующими параметрами:

```
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"
```

При использовании режима шифрования трафика убедитесь в наличии в конфигурационном файле YDB установленных путей к файлам ключей и сертификатов в секциях `interconnect_config` и `grpc_config`:

```
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
```

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 защищенный каталог на каждом из узлов кластера YDB. Ниже приведен пример команд для создания защищенного каталога и копирования файлов с ключами и сертификатами.

```
sudo mkdir -p /opt/ydb/certs
sudo cp -v ca.crt /opt/ydb/certs/
sudo cp -v node.crt /opt/ydb/certs/
sudo cp -v node.key /opt/ydb/certs/
sudo cp -v web.pem /opt/ydb/certs/
sudo chown -R ydb:ydb /opt/ydb/certs
sudo chmod 700 /opt/ydb/certs
```

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. Выполнив специальную команду на каждой машине, инициализируйте эту директорию файлом 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 --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:

```
sudo su - ydb
cd /opt/ydb
export LD_LIBRARY_PATH=/opt/ydb/lib
/opt/ydb/bin/ydbd server --log-level 3 --syslog --tcp --config-dir /opt/ydb/cfg \
    --grpcs-port 2135 --ic-port 19001 --mon-port 8765 --mon-cert /opt/ydb/certs/web.pem --node static
```

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 \
    --config-dir /opt/ydb/cfg \
    --grpcs-port 2135 --ic-port 19001 --mon-port 8765 \
    --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
```

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`, путь к которому должен быть указан при выполнении соответствующих команд. Перед выполнением соответствующих команд скопируйте файл `ca.crt` на сервер, на котором эти команды будут выполняться.

Порядок действий по инициализации кластера зависят от того, включен ли в конфигурационном файле YDB режим аутентификации пользователей.

Аутентификация включена

Для выполнения административных команд (включая инициализацию кластера, создание баз данных, управление дисками и другие) в кластере со включённым режимом аутентификации пользователей необходимо предварительно получить аутентификационный токен с использованием клиента YDB CLI версии 2.0.0 или выше. Клиент YDB CLI следует установить на любом компьютере, имеющем сетевой доступ к узлам кластера (например, на одном из узлов кластера), в соответствии с инструкцией по установке.

При первоначальной установке кластера в нём существует единственная учётная запись `root` с пустым паролем, поэтому команда получения токена выглядит следующим образом:

```
ydb -e grpcs://<node1.ydb.tech>:2135 -d /Root --ca-file ca.crt \
     --user root --no-password auth get-token --force > token-file
```

В качестве сервера для подключения (параметр `-e` или `--endpoint`) может быть указан любой из серверов хранения в составе кластера.

При успешном выполнении указанной выше команды аутентификационный токен будет записан в файл `token-file`. Файл токена необходимо скопировать на один из серверов хранения в составе кластера, а затем на выбранном сервере выполнить команды:

```
export LD_LIBRARY_PATH=/opt/ydb/lib
ydb --token-file token-file --ca-file ca.crt -e grpcs://<node.ydb.tech>:2135 \
    admin cluster bootstrap --uuid <строка>
echo $?
```

Аутентификация выключена

На одном из серверов хранения в составе кластера выполните команды:

```
export LD_LIBRARY_PATH=/opt/ydb/lib
ydb --ca-file ca.crt -e grpcs://<node.ydb.tech>:2135 \
    admin cluster bootstrap --uuid <строка>
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 сертификата центра регистрации `ca.crt`, аналогично описанному выше порядку выполнения действий по инициализации кластера.

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.

Порядок действий по созданию базы данных зависит от того, включен ли в конфигурационном файле YDB режим аутентификации пользователей.

Аутентификация включена

Необходимо получить аутентификационный токен. Может использоваться файл с токеном аутентификации, полученный при выполнении инициализации кластера, либо подготовлен новый токен.

Файл токена необходимо скопировать на один из серверов хранения в составе кластера, а затем на выбранном сервере выполнить команды:

```
export LD_LIBRARY_PATH=/opt/ydb/lib
/opt/ydb/bin/ydbd -f token-file --ca-file ca.crt -s grpcs://`hostname -f`:2135 \
    admin database /Root/testdb create ssd:1
echo $?
```

Аутентификация выключена

На одном из серверов хранения в составе кластера выполните команды:

```
export LD_LIBRARY_PATH=/opt/ydb/lib
/opt/ydb/bin/ydbd --ca-file ca.crt -s grpcs://`hostname -s`:2135 \
    admin database /Root/testdb create ssd:1
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, должно соответствовать настройке `domains_config`.`domain`.`name` в файле конфигурации кластера;
- `testdb` - имя создаваемой базы данных;
- `ssd:1` - имя пула хранения и количество выделяемых групп хранения. Имя пула обычно означает тип устройств хранения данных и должно соответствовать настройке `storage_pool_types`.`kind` внутри элемента `domains_config`.`domain` файла конфигурации.

Создайте на каждом динамическом узле директорию `/opt/ydb/cfg` для работы кластера с конфигурацией. В случае поднятия нескольких узлов на одной машине, используйте одну и ту же директорию. Выполнив специальную команду на каждой машине, инициализируйте 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>
```

В примере команды выше `<node.ydb.tech>` - FQDN статического узла кластера, с которого будет загружен файл конфигурации.

Запустите динамические узлы

Вручную

Запустите динамический узел YDB для базы `/Root/testdb`:

```
sudo su - ydb
cd /opt/ydb
export LD_LIBRARY_PATH=/opt/ydb/lib
/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 \
    --config-dir /opt/ydb/cfg \
    --tenant /Root/testdb \
    --node-broker grpcs://<ydb1>:2135 \
    --node-broker grpcs://<ydb2>:2135 \
    --node-broker grpcs://<ydb3>:2135
```

В примере команды выше `<ydbN>` - FQDN трех любых серверов, на которых запущены статические узлы кластера.

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 \
    --config-dir /opt/ydb/cfg \
    --tenant /Root/testdb \
    --node-broker grpcs://<ydb1>:2135 \
    --node-broker grpcs://<ydb2>:2135 \
    --node-broker grpcs://<ydb3>:2135
LimitNOFILE=65536
LimitCORE=0
LimitMEMLOCK=32212254720

[Install]
WantedBy=multi-user.target
```

В примере файла выше `<ydbN>` - FQDN трех любых серверов, на которых запущены статические узлы кластера.

Запустите динамический узел YDB для базы `/Root/testdb`:

```
sudo systemctl start ydbd-testdb
```

Запустите дополнительные динамические узлы на других серверах для масштабирования и обеспечения отказоустойчивости базы данных.

Первоначальная настройка учетных записей

Если в файле настроек кластера включен режим аутентификации, то перед началом работы с кластером YDB необходимо выполнить первоначальную настройку учетных записей.

При первоначальной установке кластера YDB автоматически создается учетная запись `root` с пустым паролем, а также стандартный набор групп пользователей, описанный в разделе Начальная настройка безопасности кластера.

Для выполнения первоначальной настройки учетных записей в созданном кластере YDB выполните следующие операции:

1. Установите YDB CLI, как описано в документации.
2. Выполните установку пароля учетной записи `root`:

   ```
   ydb --ca-file ca.crt -e grpcs://<node.ydb.tech>:2136 -d /Root/testdb --user root --no-password \
       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.
3. Создайте дополнительные учетные записи:

   ```
   ydb --ca-file ca.crt -e grpcs://<node.ydb.tech>:2136 -d /Root/testdb --user root --password-file
   <path_to_root_pass_file> \
       yql -s 'CREATE USER user1 PASSWORD "passw0rd"'
   ```

4. Установите права учетных записей, включив их во встроенные группы:

   ```
   ydb --ca-file ca.crt -e grpcs://<node.ydb.tech>:2136 -d /Root/testdb --user root --password-file
   <path_to_root_pass_file> \
       yql -s 'ALTER GROUP `ADMINS` ADD USER user1'
   ```

В перечисленных выше примерах команд `<node.ydb.tech>` — FQDN сервера, на котором запущен любой динамический узел, обслуживающий базу `/Root/testdb`. При подключении по SSH к динамическому узлу YDB удобно использовать конструкцию `grpcs://$(hostname -f):2136` для получения FQDN.

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.
2. Создайте тестовую строковую (`test_row_table`) или колоночную таблицу (`test_column_table`):

Создание строковой таблицы

```
ydb --ca-file 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.crt -e grpcs://<node.ydb.tech>:2136 -d /Root/testdb --user root \
    yql -s 'CREATE TABLE `testdir/test_column_table` (id Uint64, title Utf8, PRIMARY KEY (id)) WITH (STORE = COLUMN);'
```

Где `<node.ydb.tech>` - FQDN сервера, на котором запущен динамический узел, обслуживающий базу `/Root/testdb`.

Проверка доступа ко встроенному web-интерфейсу

Для проверки доступа ко встроенному web-интерфейсу YDB достаточно открыть в Web-браузере страницу с адресом `https://<node.ydb.tech>:8765`, где `<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 встроенного web-интерфейса. Описание доступных функций и пользовательского интерфейса приведено в разделе Использование встроенного web-интерфейса.

Примечание

Обычно для обеспечения доступа ко встроенному web-интерфейсу YDB 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 исключается подсекция `security_config` в секции `domains_config`, а также целиком исключаются секции `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

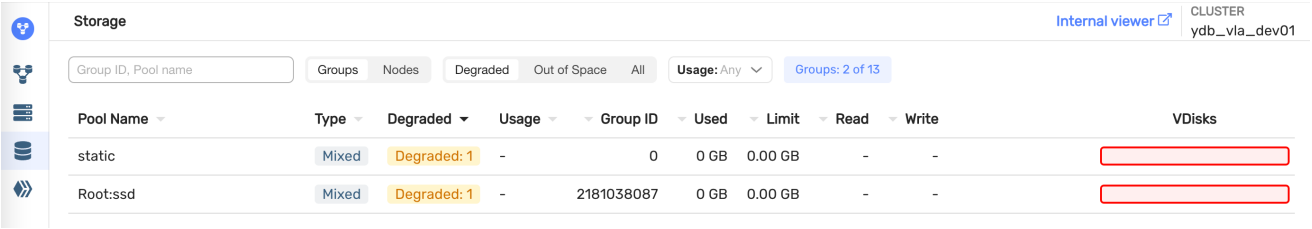
YDB распределенная система, поддерживающая плавное обновление (rolling restart) без даунтайма и деградации производительности.

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.

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 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.

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 передает 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 развернутом 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 --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

Для включения обязательной авторизации узлов баз данных в файл статической 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. По техническим причинам в этом списке также должен присутствовать субъект доступа `root@builtin`. Пример:

   ```
   domains_config:
       ...
       security_config:
           enforce_user_token_requirement: true
           ...
           register_dynamic_node_allowed_sids:
           - "root@builtin" # требуется по техническим причинам
           - "registerNode@cert"
   ```

   Более подробная информация по настройке параметров аутентификации кластера приведена в соответствующем разделе документации.

3. Обновите файлы статической конфигурации на всех узлах кластера вручную либо с помощью плейбука Ansible.
4. Выполните поэтапный перезапуск узлов хранения кластера с помощью ydbops либо с использованием плейбука Ansible.
5. Выполните поэтапный перезапуск узлов баз данных кластера с помощью ydbops либо с использованием плейбука Ansible.

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.

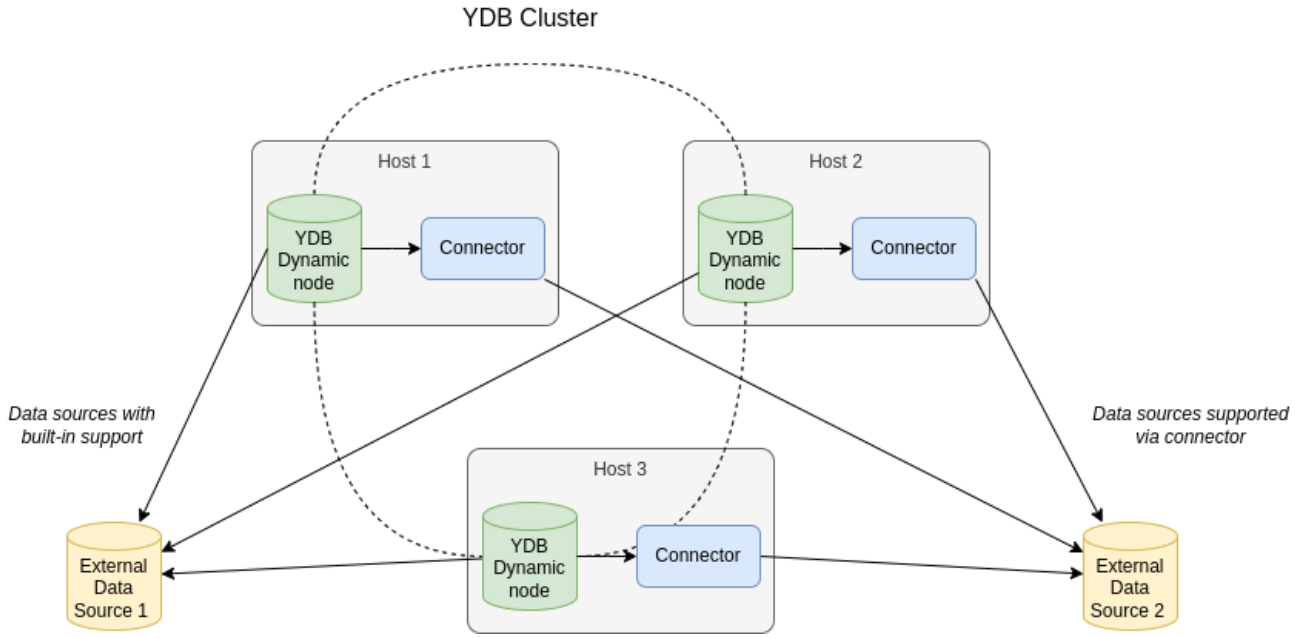
Примечание

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. Если для доступа к нужному вам источнику трубуется развернуть коннектор, в конфигурационном файле YDB в секции `query_service_config` добавьте подсекцию `generic` по приведённому ниже образцу. В полях `connector.endpoint.host` и `connector.endpoint.port` укажите сетевой адрес коннектора (по умолчанию `localhost` и `2130`). При совместном размещении коннектора и динамического узла YDB на одном сервере установка шифрованных соединений между ними *не требуется*, но в случае необходимости вы можете включить шифрование, передав значение `true` в поле `connector.use_ssl` и указав путь до сертификата CA, использованного для подписи TLS-ключей коннектора, в `connector.ssl_ca_crt`:

```
query_service_config:
    generic:
        connector:
            endpoint:
                host: localhost                # имя хоста, где развернут коннектор
                port: 2130                     # номер порта для слушающего сокета коннектора
            use_ssl: false                     # флаг, включающий шифрование соединений
            ssl_ca_crt: "/opt/ydb/certs/ca.crt" # (опционально) путь к сертификату CA
        default_settings:
            - name: DateTimeFormat
              value: string
            - name: UsePredicatePushdown
              value: "true"
```

4. В конфигурационном файле YDB добавьте секцию `feature_flags` следующего содержания:

```
feature_flags:
    enable_external_data_sources: true
    enable_script_execution_operations: true
```

5. Продолжайте развёртывание динамического узла YDB по инструкции.

Обзор управления дисковой подсистемой кластера

Управление дисковой подсистемой кластера включает следующие действия:

- Изменение конфигурации:
  - Расширение кластера.
  - Добавление групп хранения.
- Обслуживание:
  - Замена FQDN узла.
  - Включение и выключение Scrubbing.
  - Работа с SelfHeal.
  - Декомиссия части кластера.
  - Перевоз VDisk'ов.
- 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
```

Работа с SelfHeal

В процессе работы кластеров могут выходить из строя узлы целиком или отдельные блочные устройства, на которых работает YDB.

SelfHeal используется для сохранения работоспособности и отказоустойчивости кластера, если невозможно быстро восстановить вышедшие из строя узлы или устройства.

SelfHeal позволяет:

- обнаружить неисправные элементы системы;
- перенести неисправные элементы в щадящем режиме без потери данных и расформирования групп хранения.

SelfHeal включен по умолчанию.

Компонент YDB, отвечающий за SelfHeal, называется Sentinel.

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
```

Настройки SelfHeal

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 попытками на изменение статуса PDisk в BSC. CMS наблюдает состояние диска с интервалом **State update interval**. Если диск пребывает в одном состоянии несколько циклов **Status update interval**, то CMS меняет его статус в BSC. Далее идут настройки количества циклов обновления, через которое CMS будет изменять статус диска. Если состояние диска `Normal` , то диск переводится в статус `ACTIVE` , в остальных состояниях диск переводится в статус `FAULTY` . Значение `0` выключает изменение статуса для состояния (так реализовано для `Unknown` по умолчанию). Например, при настройках по умолчанию, если CMS наблюдает состояние диска `Initial` на протяжении 5 циклов `Status update interval` по 60 с каждый, статус диска будет изменен на `FAULTY` .
Default state limit	Для состояний, для которых нет 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
```

Декомиссия части кластера

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 переводятся в состояние `DECOMMIT_PENDING`, чтобы исключить перемещение данных внутри ЦОД-1.
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]"
```

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> GroupGeneration:
<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 вьюере.

   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 вьюере.
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.

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-linux-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-certificates.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-certificates.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.

Настройка мониторинга кластера YDB

На этой странице рассказано, как настроить мониторинг кластера YDB.

YDB предоставляет множество метрик состояния системы. Мгновенные значения метрик можно посмотреть в веб-интерфейсе:

```
http://<ydb-server-address>:<ydb-port>/counters/
```

где:

- `<ydb-server-address>` – адрес сервера YDB.

  Для локального однонодового кластера YDB, запущенного с помощью инструкции по Быстрому началу работы, используйте адрес `localhost`.
- `<ydb-port>` – порт YDB. Значение по умолчанию: 8765.

Связанные метрики объединены в подгруппы (например `counters auth`). Чтобы посмотреть значения метрик только определенной подгруппы, перейдите по URL следующего вида:

```
http://<ydb-server-address>:<ydb-port>/counters/counters=<servicename>/
```

- `<servicename>` — имя подгруппы метрик.

Например, данные об утилизации аппаратных ресурсов сервера доступны по следующему URL:

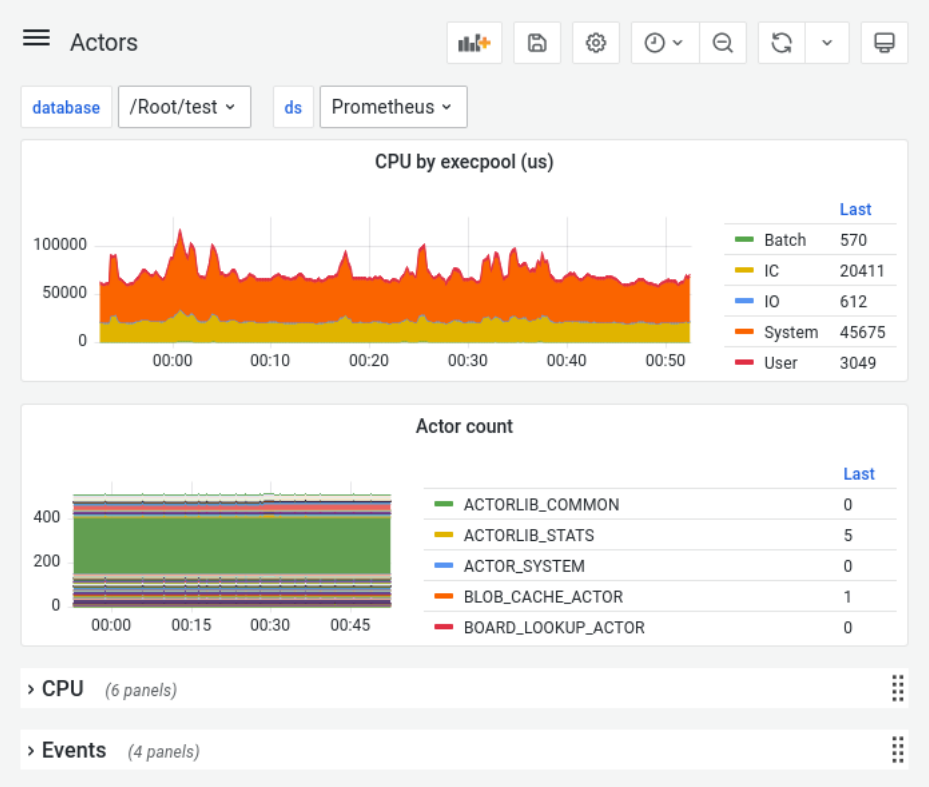
```
http://<ydb-server-address>:<ydb-port>/counters/counters=utils
```

Для сбора значений метрик вы можете использовать популярный инструмент с открытым исходным кодом Prometheus или любую другую систему с поддержкой этого формата. Значения метрик YDB в формате Prometheus доступны по URL следующего вида:

```
http://<ydb-server-address>:<ydb-port>/counters/counters=<servicename>/prometheus
```

- `<servicename>` — имя подгруппы метрик.

Визуализировать данные можно с помощью любой системы, которая поддерживает формат Prometheus, например Grafana, Zabbix или Amazon CloudWatch:



Настройка мониторинга с помощью Prometheus и Grafana

Чтобы настроить мониторинг кластера YDB с помощью Prometheus и Grafana:

1. Установите Prometheus.

2. Отредактируйте файл конфигурации Prometheus:
   2.1. В секции `targets` укажите адреса всех серверов кластера YDB и порты каждого из узлов хранения и узлов базы данных, работающих на сервере.

   Например, для кластера из трёх серверов, где на каждом сервере работает один узел хранения на порте 8765 и два узла базы данных на портах 8766 и 8767, необходимо прописать девять адресов для всех подгрупп метрик, кроме дисковых (для дисковых подгрупп метрик необходимо указать только адреса узлов хранения):

   ```
   static_configs:
   - targets:
     - ydb-s1.example.com:8765
     - ydb-s1.example.com:8766
     - ydb-s1.example.com:8767
     - ydb-s2.example.com:8765
     - ydb-s2.example.com:8766
     - ydb-s2.example.com:8767
     - ydb-s3.example.com:8765
     - ydb-s3.example.com:8766
     - ydb-s3.example.com:8767
   ```

   Для локального однонодового кластера YDB, в секции `targets` прописывается один адрес:

   ```
   - targets: ["localhost:8765"]
   ```

   2.2. При необходимости, в секции `tls_config` укажите сертификат центра регистрации (Certification Authority, CA), которым подписаны остальные сертификаты TLS кластера YDB:

   ```
   scheme: https
   tls_config:
       ca_file: '<ydb-ca-file>'
   ```

3. Запустите Prometheus, используя отредактированный файл конфигурации.
4. Установите и запустите Grafana.
5. Создайте источник данных с типом `prometheus` в Grafana и подсоедините его к запущенному экземпляру Prometheus.
6. Загрузите дашборды YDB в Grafana.

   Вы можете загрузить дашборды с помощью инструмента Import Grafana UI или выполнить скрипт. Обратите внимание, что скрипт использует базовую аутентификацию в Grafana. Для других случаев модифицируйте скрипт.

   Ознакомьтесь со справочником по дашбордам Grafana.

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 во встроенном интерфейсе YDB.

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.

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 DBA.

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 слотов (VSlot), которое может быть создано на этом PDisk
NumActiveSlots	Uint32		Количество работающих слотов в настоящий момент

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) формируют 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

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.

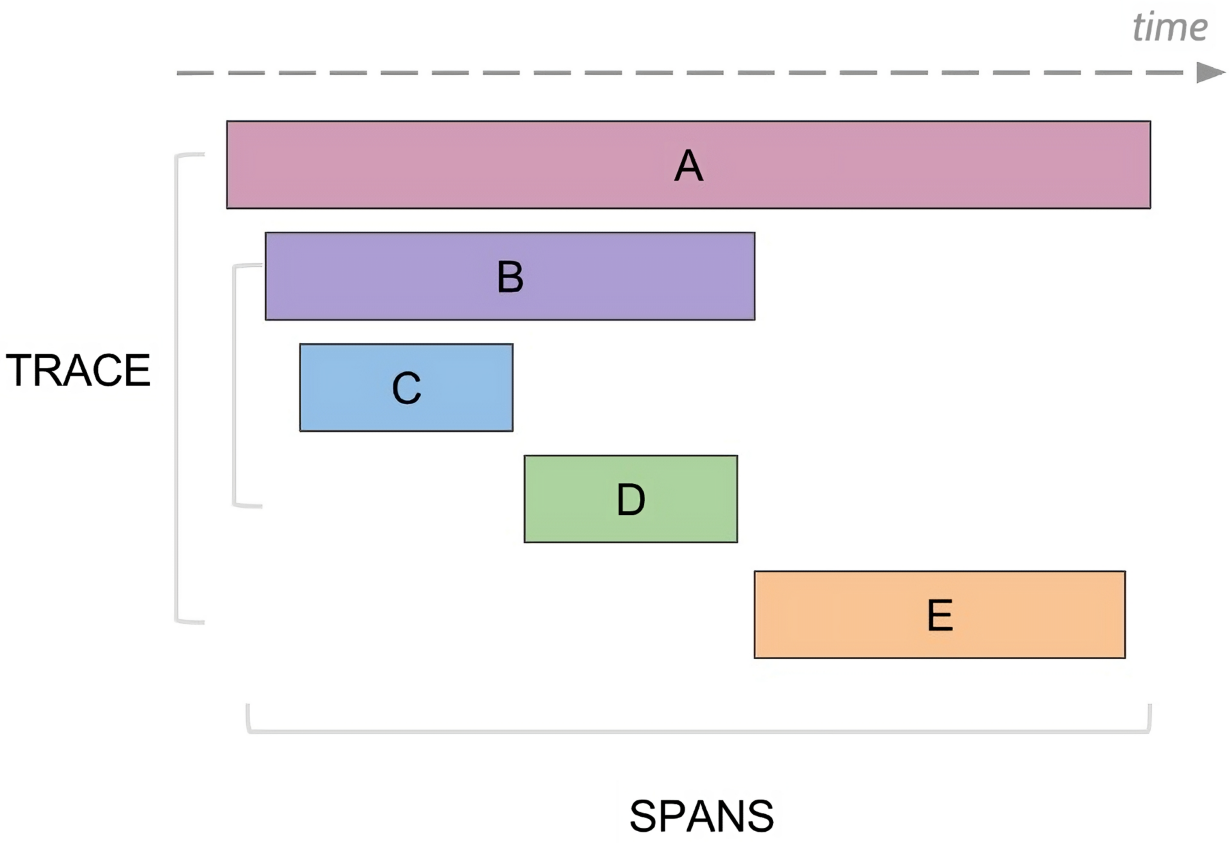
Примечание

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 случе 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

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.
- Альтернативно, YDB предоставляет PostgreSQL-совместимый API. Он предназначен для упрощения миграции существующих приложений, которые переросли PostgreSQL. Однако это также полезно для экзотических языков программирования, для которых есть клиентская библиотека PostgreSQL, но нет YDB SDK. Обратитесь к документации по совместимости PostgreSQL, чтобы проверить, соответствует ли ее полнота вашим потребностям.
- 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 или драйвер PostgreSQL в зависимости от выбранного выше пути.

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
- Встроенный пользовательский интерфейс
- Любая среда разработки SQL, поддерживающая JDBC.
- psql или pgAdmin для пути совместимости с PostgreSQL.

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.

Для пути совместимости с PostgreSQL

- Используйте свой опыт или обратитесь к любым ресурсам по PostgreSQL. Однако сверяйтесь со списком поддерживаемых функций и операторов, чтобы скорректировать свои ожидания.

Тестирование

Как написать тесты на приложения, работающие с YDB:

- Для функциональных тестов вы можно имитировать (mock) ответы YDB, используя подходящуй 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.

Тестовое приложение

В данном разделе разбирается код однотипного тестового приложения, реализованного с использованием YDB SDK на разных языках программирования:

- C++
- C# (.NET)
- Go
- Java
- Node.js
- Python

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# (.NET) | Go | Java | Node.js | PHP | Python

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# (.NET) | Go | Java | Node.js | PHP | Python

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# (.NET) | Go | Java | Node.js | PHP | Python

Получение выборки данных

Выполняется запрос на получение выборки данных с использованием команды `SELECT` языка запросов YQL. Демонстрируется обработка полученной выборки в приложении.

C++ | C# (.NET) | Go | Java | Node.js | PHP | Python

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# (.NET) | Go | Java | Node.js | PHP | Python

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# (.NET) | Go | Java | Node.js | 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# (.NET) | Go | Java | Node.js | PHP | Python

Обработка ошибок

Подробно об обработке ошибок написано в разделе Обработка ошибок в API.

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

Выбор первичного ключа

Рекомендации по выбору правильного первичного ключа для таблицы зависят от ее типа:

- Строковые таблицы
- Колоночные таблицы

Вторичные индексы

Внимание

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.

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.

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`.

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

Для обращения к строковой таблице по вторичному индексу его имя должно быть явно указано 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`.

Без указания секции `VIEW` для выполнения такого запроса будет полностью просканирована строковая таблица `orders`.

В транзакционных приложениях подобные информационные запросы выполняются с использованием постраничного вывода данных, что исключает рост стоимости и времени исполнения при увеличении количества записей, подходящих под условия фильтрации. Описанный на примере первичного ключа подход к написанию постраничных запросов применим также и к колонкам, включенным во вторичный индекс.

Также реализована экспериментальная возможность выбора вторичного индекса для использования в запросе в автоматическом режиме. Алгоритм выбора на данный момент является rule-based и использует только текст запроса для автоматического выбора вторичного индекса.

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, $param5)
```

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.

`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

Внимание

Функционал векторных индексов доступен в тестовом режиме в main. В стабильной версии планируется в 25.1.

В настоящее время не поддерживается:

- изменение строк в таблицах с векторными индексами;
- построение индекса для векторов с битовым квантованием.

Эти ограничения могут быть устранены в будущих версиях.

Векторные индексы — это специализированные структуры данных, которые позволяют эффективно выполнять векторный поиск в многомерных пространствах. В отличие от вторичных индексов, которые оптимизируют поиск 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.

Характеристики векторных индексов

Векторные индексы в YDB решают задачу поиска ближайших соседей с использованием функций схожести или расстояния. Поддерживается несколько функций расстояния/схожести: "inner_product", "cosine" (схожесть) и "cosine", "euclidean", "manhattan" (расстояние).

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).

Внутри векторный индекс состоит из скрытых индексных таблиц вида `indexImpl*Table`. В запросах на выборку с использованием векторного индекса, индексные таблицы будут отображены в статистике запросов.

Виды векторных индексов

Векторный индекс может быть **покрывающим**, что означает включение дополнительных колонок для возможности чтения из индекса без обращения к основной таблице.

Или же он может быть **префиксным**, что позволяет учитывать дополнительные колонки для быстрой фильтрации при выполнении чтения.

Ниже приведены примеры создания векторного индекса различных типов.

Базовый векторный индекс

Глобальный векторный индекс по колонке `embedding`:

```
ALTER TABLE my_table
  ADD INDEX my_index
  GLOBAL USING vector_kmeans_tree
  ON (embedding)
  WITH (distance=cosine, vector_type="uint8", vector_dimension=512, levels=2, clusters=128);
```

Векторный индекс с покрывающими колонками

Покрывающий векторный индекс, включающий дополнительную колонку `data`, чтобы избежать чтения из основной таблицы при поиске:

```
ALTER TABLE my_table
  ADD INDEX my_index
  GLOBAL USING vector_kmeans_tree
  ON (embedding) COVER (data)
  WITH (distance=cosine, vector_type="uint8", vector_dimension=512, levels=2, clusters=128);
```

Векторный индекс с префиксом

Векторный индекс с префиксом, позволяющий фильтровать по префиксной колонке `user` в момент выполнения векторного поиска:

```
ALTER TABLE my_table
  ADD INDEX my_index
  GLOBAL USING vector_kmeans_tree
  ON (user, embedding)
  WITH (distance=cosine, vector_type="uint8", vector_dimension=512, levels=2, clusters=128);
```

Векторный индекс с префиксом и покрывающими колонками

Векторный индекс с префиксом и покрывающими колонками:

```
ALTER TABLE my_table
  ADD INDEX my_index
  GLOBAL USING vector_kmeans_tree
  ON (user, embedding) COVER (data)
  WITH (distance=cosine, vector_type="uint8", vector_dimension=512, levels=2, clusters=128);
```

Создание векторных индексов

Векторные индексы можно создавать:

- при создании таблицы с помощью YQL-оператора CREATE TABLE;
- добавлять к существующей таблице с помощью YQL-оператора ALTER TABLE.

Использование векторных индексов

Запросы к векторным индексам выполняются с использованием синтаксиса `VIEW` в YQL. Для индексов с префиксом укажите префиксные колонки в условии `WHERE`:

```
DECLARE $query_vector AS List<Uint8>;

SELECT user, data
FROM my_table VIEW my_index
ORDER BY Knn::CosineSimilarity(embedding, $query_vector) DESC
LIMIT 10;
```

Подробнее о выполнении запросов `SELECT` с использованием векторных индексов можно прочитать в разделе VIEW VECTOR INDEX.

Примечание

Если не использовать выражение `VIEW`, запрос выполнит полное сканирование таблицы с попарным сравнением векторов.

Рекомендуется проверять оптимальность написанного запроса, используя статистику запросов. В частности, следует следить за отсутствием полного сканирования (full scan) основной таблицы.

Использование планов при оптимизации запросов

При написании запроса важно проанализировать план его выполнения для своевременного обнаружения и устранения причин возможного чрезмерного потребления ресурсов кластера или аномально высокого времени выполнения. В данной статье будет рассмотрен конкретный пример анализа плана запроса, а со справочной информацией о структуре планов YDB можно ознакомиться здесь.

Рассмотрим следующий запрос, выполняющий поиск серии по названию:

```
SELECT season_id, episode_id
  FROM episodes
  WHERE title = 'The Work Outing'
```

Схема строковой таблицы `episodes`:

Id	Key	Name	Type	NotNull
1	🔑	series_id	Uint64	
2	🔑	season_id	Uint64	
3	🔑	episode_id	Uint64	
5		air_date	Uint64	
4		title	Utf8	

Построим план для данного запроса, в YDB это можно сделать двумя способами:

YDB CLI

Получить план запроса через YDB CLI можно с помощью следующей команды:

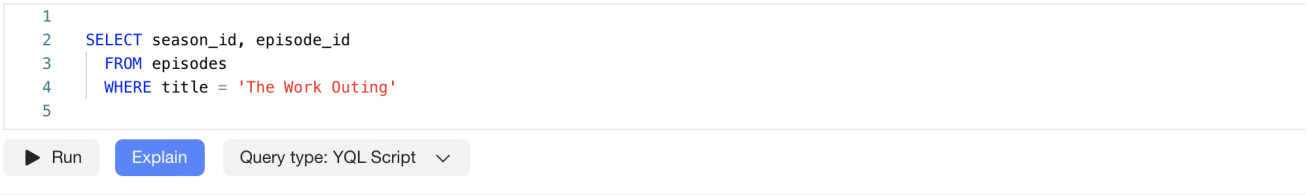
```
ydb -p <profile_name> table query explain \
  -q "SELECT season_id, episode_id
  FROM episodes
  WHERE title = 'The Work Outing'"
```

Результат:

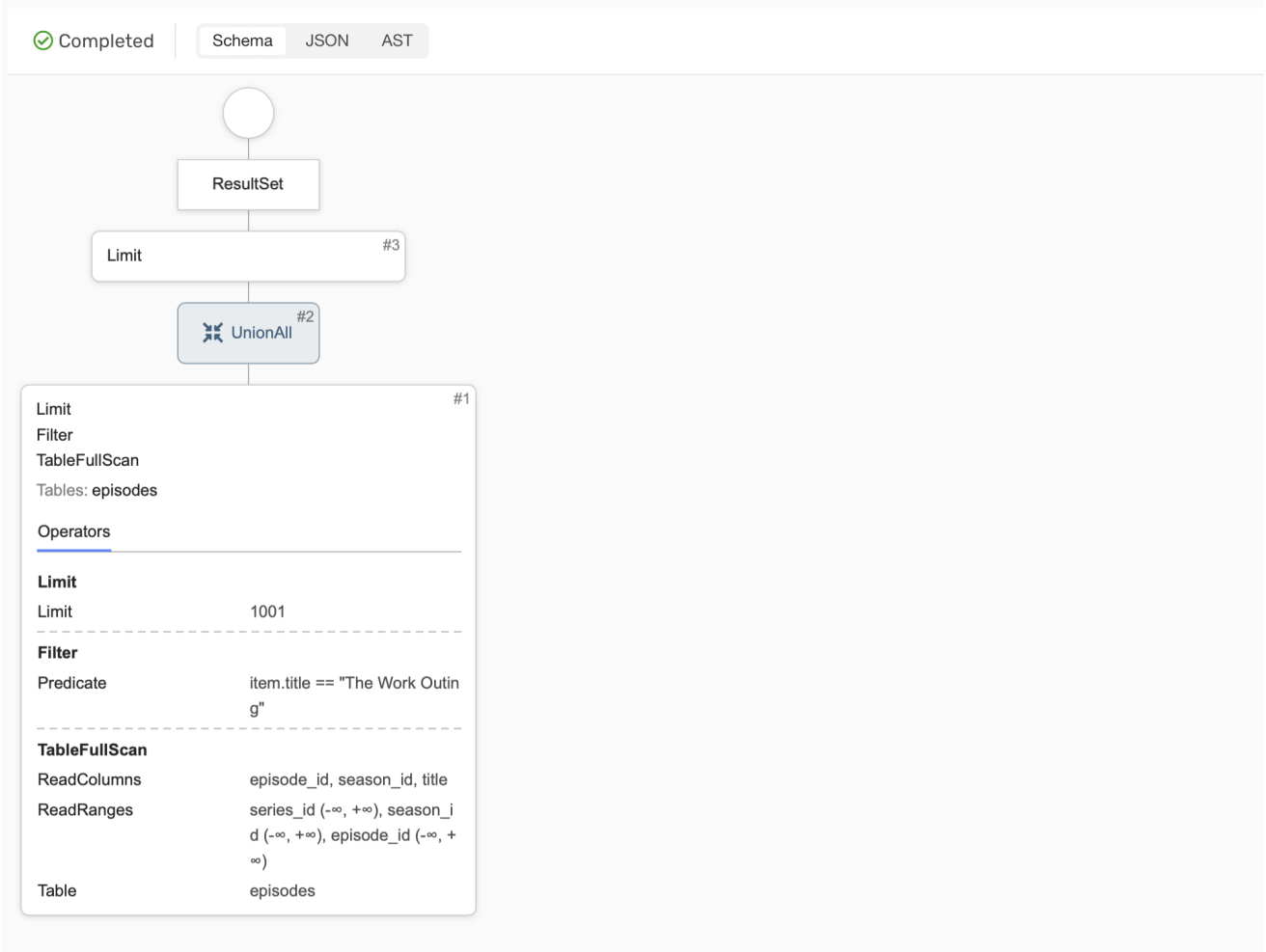
```
Query Plan:
ResultSet
└──Limit (Limit: 1001)
   └──<UnionAll>
      └──Limit (Limit: 1001)
      └──Filter (Predicate: item.title == "The Work Outing")
      └──TableFullScan (ReadRanges: ["series_id (-∞, +∞)","season_id (-∞, +∞)","episode_id (-∞, +∞)"], ReadColumns:
["episode_id","season_id","title"], Table: episodes)
         Tables: ["episodes"]
```

Embedded UI

План запроса также можно получить через Embedded UI. Для этого необходимо перейти на страницу базы данных в раздел `Query` и нажать `Explain`:



Результат:



И в визуальном и в текстовом представлении видно, что в корне этого плана возвращение данных на клиент, в листьях работа со строковыми таблицами, а на промежуточных узлах — преобразования данных. Важно обратить внимание на узел, показывающий обращение к строковой таблице `episodes`. В данном случае это `TableFullScan`, который означает выполнение полного сканирования строковой таблицы. А полное сканирование таблицы потребляет времени и ресурсов пропорционально её размеру, из-за чего по возможности их стараются избегать в таблицах, которые имеют тенденцию расти с течением времени или просто большие.

Одним из типовых способов избежать полного сканирования строковой таблицы является добавление вторичного индекса. В данном случае имеет смысл добавить вторичный индекс для колонки `title`, для этого воспользуемся запросом:

```
ALTER TABLE episodes
  ADD INDEX title_index GLOBAL ON (title)
```

Стоит отметить, что в данном примере мы используем синхронный вторичный индекс. Создание индекса в YDB — асинхронная операция, поэтому даже если запрос на построение завершился успехом, стоит подождать какое-то время, так как фактически индекс может быть еще не готов к использованию. Управлять асинхронной операцией можно через CLI.

Построим план того же запроса с использованием вторичного индекса `title_index`. Обратите внимание, что вторичный индекс надо явно указать в запросе через конструкцию `VIEW`.

YDB CLI

Команда:

```
ydb -p <profile_name> table query explain \
  -q "SELECT season_id, episode_id
  FROM episodes VIEW title_index
  WHERE title = 'The Work Outing'"
```

Результат:

```
Query Plan:
ResultSet
└──Limit (Limit: 1001)
   └──<UnionAll>
      └──Limit (Limit: 1001)
      └──Filter (Predicate: Exist(item.title))
      └──TablePointLookup (ReadRange: ["title (The Work Outing)","series_id (-∞, +∞)","season_id (-∞, +∞)","episode_id
```

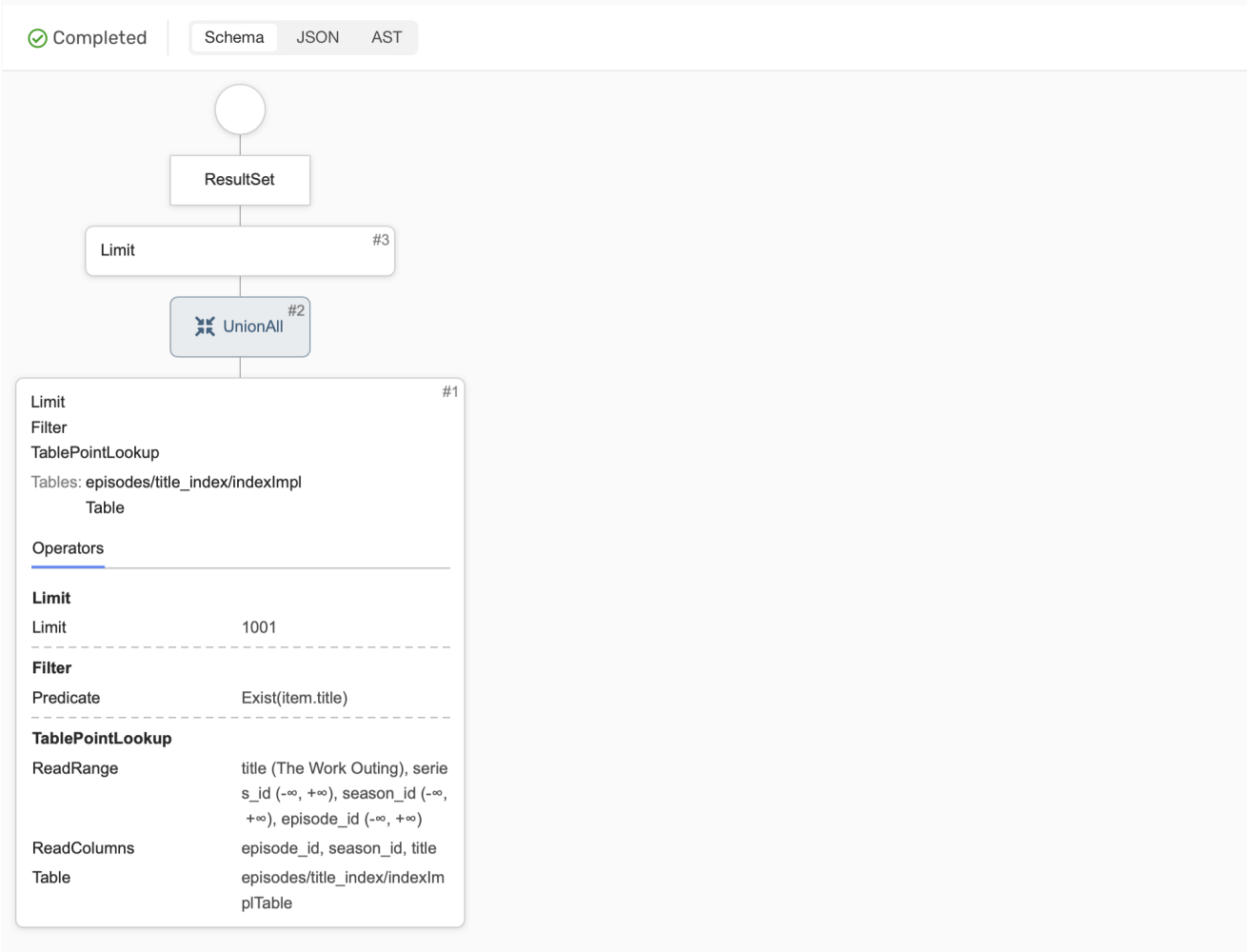
```
(-∞, +∞)"], ReadLimit: 1001, ReadColumns: ["episode_id","season_id","title"], Table:
episodes/title_index/indexImplTable)
         Tables: ["episodes/title_index/indexImplTable"]
```

Embedded UI

```
1
2   SELECT season_id, episode_id
3     FROM episodes VIEW title_index
4     WHERE title = 'The Work Outing'
5
```

Run | Explain | Query type: YQL Script

Результат:



С использованием вторичного индекса запрос выполняется без полного сканирования основной таблицы. Вместо `TableFullScan` появился `TablePointLookup` - чтение индексной таблицы по ключу, а основную таблицу мы теперь совсем не читаем, так как все нужные нам колонки содержатся в индексной таблице.

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.

Применение таймаутов

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"

  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*300) // client and by default operation timeout
defer cancel()
ctx = ydb.WithOperationTimeout(ctx, time.Millisecond*400)     // operation timeout override
ctx = ydb.WithOperationCancelAfter(ctx, time.Millisecond*300) // cancel after timeout
tx := table.TxControl(table.BeginTx(table.WithSerializableReadWrite()), table.CommitTx())
_, res, err := s.Execute(ctx, tx, query, table.NewQueryParameters())
}
```

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 подписчика 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
`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 процесорное 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`.

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
```

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 подписчика 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 подписчика 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`.
`ResourceWeight`	Веса для распределения ресурсов между пулами. Тип: `Double`.

`TotalCpuLimitPercentPerNode`	Процент доступного CPU, который могут использовать все запросы на узле в данном пуле ресурсов. Тип: `Double`.
`QueryCpuLimitPercentPerNode`	Процент доступного CPU на узле для одного запроса в пуле ресурсов. Тип: `Double`.
`QueryMemoryLimitPercentPerNode`	Процент доступной памяти на узле, который может использовать запрос в данном пуле ресурсов. Тип: `Double`.

Пример

Следующий запрос выводит информацию о настройках пула ресурсов с именем `default`:

```
SELECT
    Name,
    ConcurrentQueryLimit,
    QueueSize,
    DatabaseLoadCpuThreshold,
    ResourceWeight,
    TotalCpuLimitPercentPerNode,
    QueryCpuLimitPercentPerNode,
    QueryMemoryLimitPercentPerNode
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`.

Change Data Capture

Внимание

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":[45220497351143909]}
{"update":{"created":"2022-06-24T11:35:00.000000Z","customer":"Name7773"},"key":[23779788941838502 58]}
...
```

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

> **Внимание**
>
> 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

> **Внимание**
>
> 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

> **Внимание**
>
> 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.

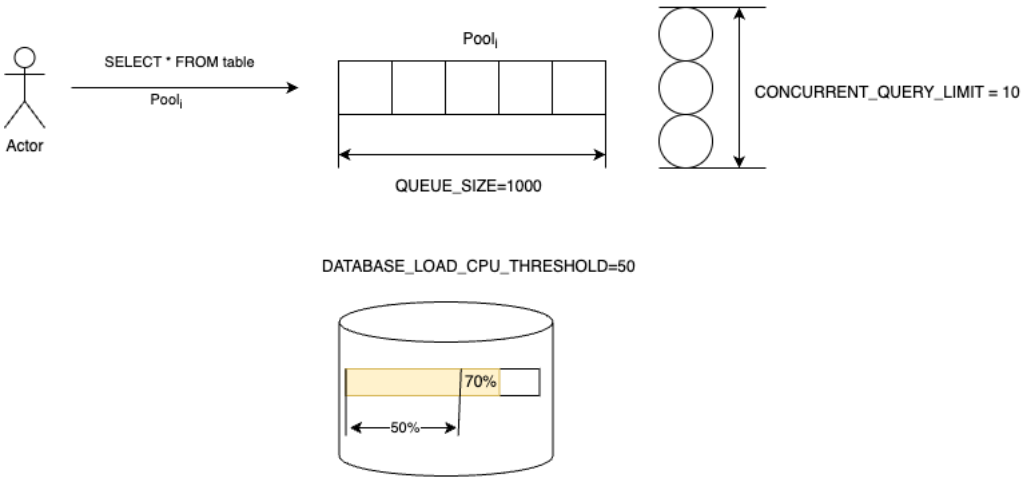
> **Важно**
>
> Представленная функциональность для управления потреблением ресурсами находится в стадии Preview.

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,
    RESOURCES_WEIGHT=100,
    QUERY_CPU_LIMIT_PERCENT_PER_NODE=50,
    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, `QUERY_CPU_LIMIT_PERCENT_PER_NODE`, `TOTAL_CPU_LIMIT_PERCENT_PER_NODE`, `QUERY_MEMORY_LIMIT_PERCENT_PER_NODE`). Между всеми пулами может разделяться CPU в случае переподписки на одном вычислительном узле с помощью `RESOURCES_WEIGHT`.



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:

На один запрос в этом пуле 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
```

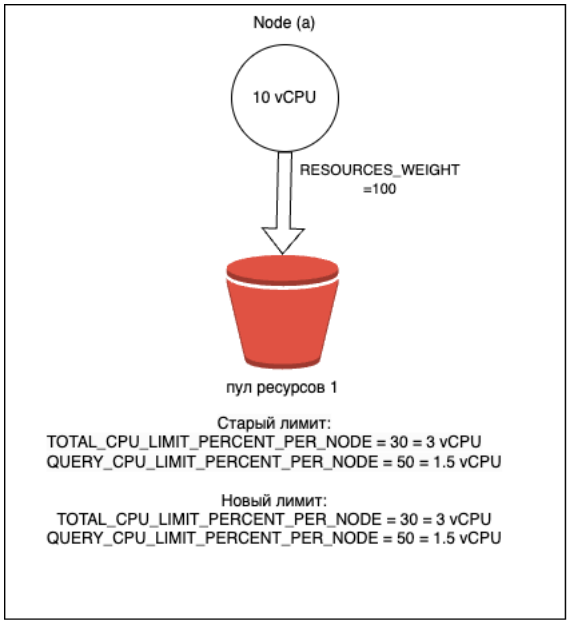
На число параллельно выполняемых запросов влияет не только `CONCURRENT_QUERY_LIMIT`, но и `DATABASE_LOAD_CPU_THRESHOLD`.

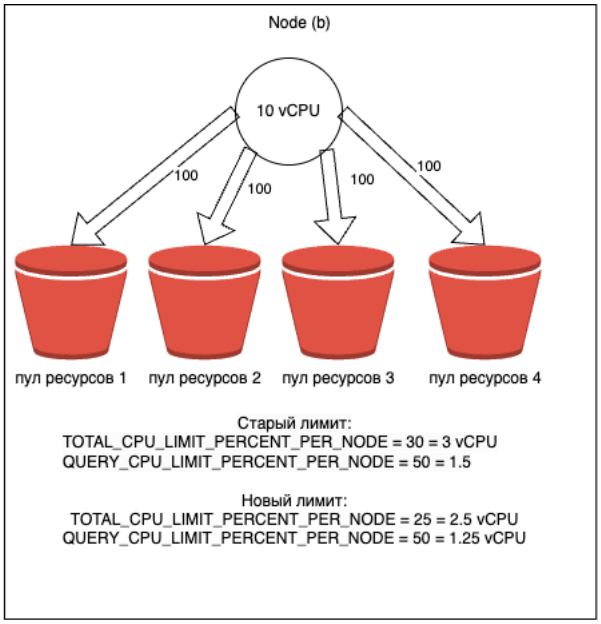
Как работает DATABASE_LOAD_CPU_THRESHOLD

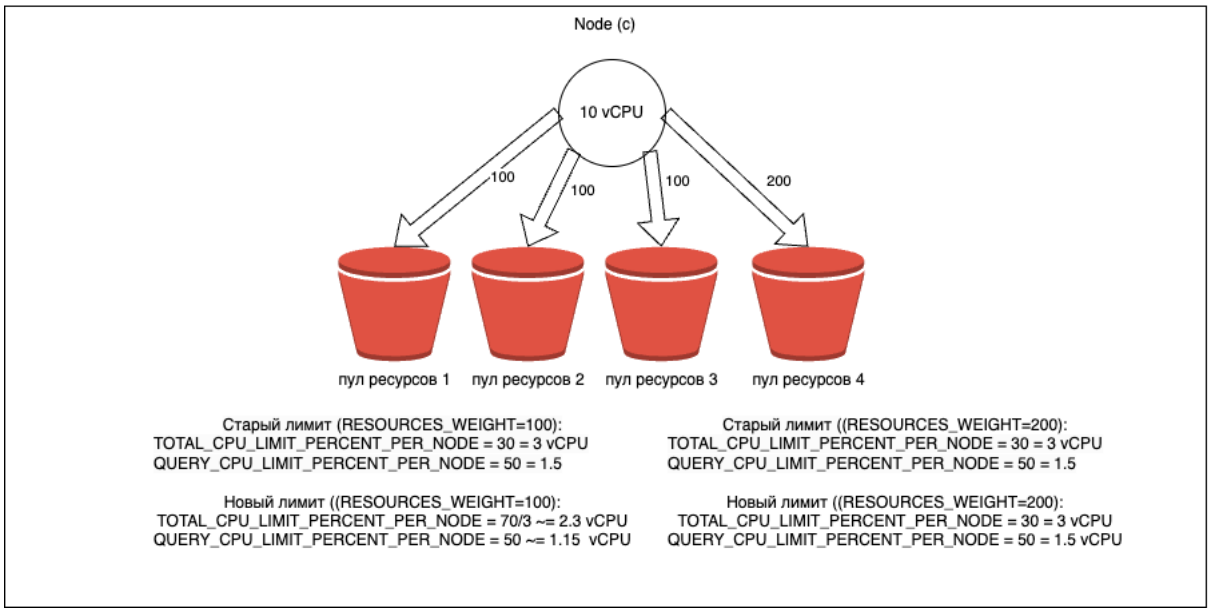
Когда запрос поступает в пул ресурсов, для которого установлен `DATABASE_LOAD_CPU_THRESHOLD`, сразу резервируется 10% от доступного CPU на узле, исходя из предположения, что запрос как минимум потребует такой объём ресурсов. Затем каждые 10 секунд происходит перерасчёт потребляемых ресурсов по всей базе 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.

Распределение ресурсов в соответствии с RESOURCES_WEIGHT







Параметр `RESOURCES_WEIGHT` начинает работать только в случае переподписки и при наличии более одного пула ресурсов в системе. В текущей реализации `RESOURCES_WEIGHT` влияет только на распределение ресурсов `vCPU`. Когда в пуле ресурсов появляются запросы, он начинает участвовать в распределении ресурсов. Для этого в пулах происходит перерасчёт лимитов согласно алгоритму Max-min fairness. Само перераспределение ресурсов выполняется на каждом вычислительном узле индивидуально, как показано на рисунке выше.

Допустим, у нас есть узел в системе с доступными . Установлены ограничения:

- ,
- .

В этом случае у пула ресурсов будет ограничение на узел и на один запрос в этом пуле (рисунок *a*). Если в системе существует 4 таких пула, и все они пытаются использовать максимальные ресурсы, это составит , что превышает предел доступных ресурсов на узле (). В этом случае начинают действовать `RESOURCES_WEIGHT`, и каждому пулу будет выделено по (рисунок *b*).

Если необходимо увеличить выделяемые ресурсы для конкретного пула, можно изменить его вес, например, на 200. Тогда этот пул получит , а остальные пулы поделят поровну оставшиеся , что составит на каждый пул (рисунок *c*).

> **Важно**
>
> Текущий алгоритм распределения ресурсов может быть изменен в будущем без поддержки обратной совместимости.

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,
    RESOURCES_WEIGHT=-1,
    QUERY_MEMORY_LIMIT_PERCENT_PER_NODE=-1,
    QUERY_CPU_LIMIT_PERCENT_PER_NODE=-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

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;
```

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.

Управление ACL классификатора пула ресурсов

Ограничений на использование классификатора пула ресурсов нет — они глобальны для всей базы данных и доступны всем пользователям. Для создания, удаления или изменения классификатора пула ресурсов необходимо иметь разрешение `ALL` на всю базу данных, которое можно выдать запросом вида:

```
GRANT ALL ON `/my_db` TO user1;
```

Для использования классификатора пула ресурсов у пользователя должен быть доступ к пулу ресурсов, на который ссылается данный классификатор.

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 задачи распределения ресурсов между командой аналитиков и условным директором (CEO). Для CEO важно иметь приоритет над вычислительными ресурсами, которые используются для аналитических задач, однако полезно обеспечить возможность команде аналитиков утилизировать больше ресурсов кластера в те периоды времени, когда CEO не использует ресурсы. 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,
    RESOURCES_WEIGHT=20,
    QUERY_CPU_LIMIT_PERCENT_PER_NODE=80,
    TOTAL_CPU_LIMIT_PERCENT_PER_NODE=100
);

CREATE RESOURCE POOL the_ceo WITH (
    CONCURRENT_QUERY_LIMIT=20,
    QUEUE_SIZE=100,
    RESOURCES_WEIGHT=100,
    QUERY_CPU_LIMIT_PERCENT_PER_NODE=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`**:
  - Имеет вес 20.
  - Ограничение на запускаемые запросы при перегрузке базы данных составляет 80% доступных ресурсов.
- **Пул ресурсов `the_ceo`**:
  - Имеет больший вес — 80.
  - Не имеет ограничений на запускаемые запросы при перегрузке.

Вес 80 для `the_ceo` фактически означает, что при конкуренции за ресурсы пул `the_ceo` будет получать приоритет в 4 раза больше, чем пул `olap`. Если в оба пула поступают запросы, система пересчитает лимиты, и для `olap` лимит `TOTAL_CPU_LIMIT_PERCENT_PER_NODE` будет сокращён до 20%, а для `the_ceo` — увеличен до 80%. Это перераспределение ресурсов основано на весах, как описано выше.

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.

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

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()-
>FormatLocalTime("%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 active
        // 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 instead,
    // 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# (.NET)

На этой странице подробно разбирается код тестового приложения, использующего C# (.NET) 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:

```
public static async Task Run(
    string endpoint,
    string database,
    ICredentialsProvider credentialsProvider)
{
    var config = new DriverConfig(
        endpoint: endpoint,
        database: database,
        credentials: credentialsProvider
    );

    using var driver = new Driver(
        config: config
    );

    await driver.Initialize();
}
```

Фрагмент кода приложения для создания сессии:

```
using var queryClient = new QueryService(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.

Для создания таблиц используется метод `queryClient.Exec` с DDL (Data Definition Language) YQL-запросом.

```
await queryClient.Exec(@"
    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)
    );
");
```

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 queryClient.Exec(@"
    UPSERT INTO series (series_id, title, release_date) VALUES
        ($id, $title, $release_date);
    ",
    new Dictionary<string, YdbValue>
    {
        { "$id", YdbValue.MakeUint64(1) },
        { "$title", YdbValue.MakeUtf8("NewTitle") },
        { "$release_date", YdbValue.MakeDate(DateTime.UtcNow) }
    }
);
```

Получение выборки данных

Выполняется запрос на получение выборки данных с использованием команды `SELECT` языка запросов YQL. Демонстрируется обработка полученной выборки в приложении.

Для чтения YQL-запросов используется методы `queryClient.ReadRow` или `queryClient.ReadAllRows`. SDK позволяет в явном виде контролировать выполнение транзакций и настраивать необходимый режим выполнения транзакций с помощью класса `TxMode`. Во фрагменте кода, приведенном ниже, используется транзакция с режимом `NoTx` и автоматическим коммитом после выполнения запроса. Значения параметров запроса передаются в виде словаря имя-значение в аргументе `parameters`.

```
var row = await queryClient.ReadRow(@"
        SELECT
            series_id,
            title,
            release_date
        FROM series
        WHERE series_id = $id;
    ",
    new Dictionary<string, YdbValue>
    {
        { "$id", YdbValue.MakeUint64(id) }
    }
);
```

Обработка результатов выполнения

Результат выполнения запроса (ResultSet) состоит из упорядоченного набора строк (Rows). Пример обработки результата выполнения запроса:

```
foreach (var row in resultSet.Rows)
{
    Console.WriteLine($"> Series, " +
        $"series_id: {(ulong)row["series_id"]}, " +
        $"title: {(string?)row["title"]}, " +
        $"release_date: {(DateTime?)row["release_date"]}");
}
```

Скан запросы

Выполняется скан запрос данных, результатом исполнения которого является стрим. Стрим позволяет считать неограниченное количество строк и объем данных.

```
await queryClient.Stream(
    $"SELECT title FROM seasons ORDER BY series_id, season_id;",
    async stream =>
    {
        await foreach (var part in stream)
        {
```

```
            foreach (var row in part.ResultSet!.Rows)
            {
                Console.WriteLine(row[0].GetOptionalUtf8());
            }
        }
    });
```

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_date",
&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
}
```

Скан запросы

Выполняется скан запрос данных, результатом исполнения которого является стрим. Стрим позволяет считать неограниченное количество строк и объем данных.

> **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://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:

```
<auth_mode_var>="<auth_mode_value>" java -jar ydb-java-examples/query-example/target/ydb-query-example.jar
grpcs://<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
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.

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, params))
            .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 дальшейшей 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 Node.js

На этой странице представлено подробное описание кода тестового приложения,
доступного в составе YDB Node.js 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:

Используя connectionString

```
const authService = getCredentialsFromEnv();
logger.debug('Driver initializing...');
const driver = new Driver({connectionString, authService});
const timeout = 10000;
if (!await driver.ready(timeout)) {
    logger.fatal(`Driver did not become ready within ${timeout}ms!`);
    process.exit(1);
}
```

Используя endpoint и database

```
const authService = getCredentialsFromEnv();
logger.debug('Driver initializing...');
const driver = new Driver({endpoint, database, authService});
const timeout = 10000;
if (!await driver.ready(timeout)) {
    logger.fatal(`Driver did not become ready within ${timeout}ms!`);
    process.exit(1);
}
```

Фрагмент кода приложения для создания сессии:

```
const result = await driver.queryClient.do({
    ...
    fn: async (session) => {
        ...
    }
});
```

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.

```
async function createTables(driver: Driver, logger: Logger) {
    logger.info('Dropping old tables and creating new ones...');
    await driver.queryClient.do({
        fn: async (session) => {

          try {
              await session.execute({
                  text: `
                      DROP TABLE ${SERIES_TABLE};
                      DROP TABLE ${EPISODES_TABLE};
                      DROP TABLE ${SEASONS_TABLE};`,
              });
          } catch (err) { // Ignore if tables are missing
              if (err instanceof SchemeError) throw err;
```

```
        }

        await session.execute({
            text: `
                CREATE TABLE ${SERIES_TABLE}
                (
                    series_id    UInt64,
                    title        Utf8,
                    series_info  Utf8,
                    release_date DATE,
                    PRIMARY KEY (series_id)
                );

                CREATE TABLE ${SEASONS_TABLE}
                (
                    series_id   UInt64,
                    season_id   UInt64,
                    title UTF8,
                    first_aired DATE,
                    last_aired DATE,
                    PRIMARY KEY (series_id, season_id)
                );

                CREATE TABLE ${EPISODES_TABLE}
                (
                    series_id  UInt64,
                    season_id  UInt64,
                    episode_id UInt64,
                    title      UTf8,
                    air_date   DATE,
                    PRIMARY KEY (series_id, season_id, episode_id),
                    INDEX      episodes_index GLOBAL ASYNC ON (air_date)
                );`,
        });
      },
    });
}
```

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:

```
async function upsertSimple(driver: Driver, logger: Logger): Promise<void> {
    logger.info('Making an upsert...');
    await driver.queryClient.do({
        fn: async (session) => {
            await session.execute({
                text: `
                   UPSERT INTO ${EPISODES_TABLE} (series_id, season_id, episode_id, title)
                   VALUES (2, 6, 1, "TBD");`,
           })
        }
    });
    logger.info('Upsert completed.')
}
```

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 `QuerySession.execute()`.

В зависимости оп параметра `rowMode` данные можно получить в JavaScript форме или как YDB структуры.

rowMode: RowType.Native

```
async function selectNativeSimple(driver: Driver, logger: Logger): Promise<void> {
    logger.info('Making a simple native select...');
    const result = await driver.queryClient.do({
        fn: async (session) => {
            const {resultSets} =
                await session.execute({
```

```
                    // rowMode: RowType.Native, // Result set columns and rows are returned as native JavaScript values.
This is the default behavior.
                    text: `
                        SELECT series_id,
                               title,
                               release_date
                        FROM ${SERIES_TABLE}
                        WHERE series_id = 1;`,
                });
            const {value: resultSet1} = await resultSets.next();
            const rows: any[][] = []
            for await (const row of resultSet1.rows) rows.push(row);
            return {cols: resultSet1.columns, rows};
        }
    });
    logger.info(`selectNativeSimple cols: ${JSON.stringify(result.cols, null, 2)}`);
    logger.info(`selectNativeSimple rows: ${JSON.stringify(result.rows, null, 2)}`);
}
```

rowMode: RowType.Ydb

```
async function selectTypedSimple(driver: Driver, logger: Logger): Promise<void> {
    logger.info('Making a simple typed select...');
    const result = await driver.queryClient.do({
        fn: async (session) => {
            const {resultSets} =
                await session.execute({
                    rowMode: RowType.Ydb, // enables typedRows() on result sets
                    text: `
                        SELECT series_id,
                               title,
                               release_date
                        FROM ${SERIES_TABLE}
                        WHERE series_id = 1;`,
                });
            const {value: resultSet1} = await resultSets.next();
            const rows: Series[] = [];
            // Note: resultSet1.rows will iterate YDB IValue structures
            for await (const row of resultSet1.typedRows(Series)) rows.push(row);
            return {cols: resultSet1.columns, rows};
        }
    });
    logger.info(`selectTypedSimple cols: ${JSON.stringify(result.cols, null, 2)}`);
    logger.info(`selectTypedSimple rows: ${JSON.stringify(result.rows, null, 2)}`);
}
```

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.

```
async function selectWithParameters(driver: Driver, data: ThreeIds[], logger: Logger): Promise<void> {

  await driver.queryClient.do({
      fn: async (session) => {
          for (const [seriesId, seasonId, episodeId] of data) {
              const episode = new Episode({seriesId, seasonId, episodeId, title: '', airDate: new Date()});
              const {resultSets, opFinished} = await session.execute({
                  parameters: {
                      '$seriesId': episode.getTypedValue('seriesId'),
                      '$seasonId': episode.getTypedValue('seasonId'),
                      '$episodeId': episode.getTypedValue('episodeId')
                  },
                  text: `
                      DECLARE $seriesId AS Uint64;
                      DECLARE $seasonId AS Uint64;
                      DECLARE $episodeId AS Uint64;

                      SELECT title,
                             air_date
                      FROM episodes
                      WHERE series_id = $seriesId
                        AND season_id = $seasonId
                        AND episode_id = $episodeId;`
```

```
            });
            const {value: resultSet} = await resultSets.next();
            const {value: row} = await resultSet.rows.next();
            await opFinished;
            logger.info(`Parametrized select query ${JSON.stringify(row, null, 2)}`);
        }
    }
  });
}
```

Скан запросы

Выполняется скан запрос данных, результатом исполнения которого является стрим. Стрим позволяет считать неограниченное количество строк и объем данных.

Для получения данных потоком используется метод `QuerySession.execute()`.

```
async function selectWithParametrs(driver: Driver, data: ThreeIds[], logger: Logger): Promise<void> {
  logger.info('Selecting prepared query...');
  await driver.queryClient.do({
      fn: async (session) => {
          for (const [seriesId, seasonId, episodeId] of data) {

              const episode = new Episode({seriesId, seasonId, episodeId, title: '', airDate: new Date()});

              const {resultSets, opFinished} = await session.execute({
                  parameters: {
                      '$seriesId': episode.getTypedValue('seriesId'),
                      '$seasonId': episode.getTypedValue('seasonId'),
                      '$episodeId': episode.getTypedValue('episodeId')
                  },
                  text: `
                      DECLARE $seriesId AS Uint64;
                      DECLARE $seasonId AS Uint64;
                      DECLARE $episodeId AS Uint64;

                      SELECT title,
                             air_date
                      FROM episodes
                      WHERE series_id = $seriesId
                        AND season_id = $seasonId
                        AND episode_id = $episodeId;`
              });
              const {value: resultSet} = await resultSets.next();
              const {value: row} = await resultSet.rows.next();
              await opFinished;
              logger.info(`Parametrized select query ${JSON.stringify(row, null, 2)}`);
          }
      }
  });
}
```

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 `Session.beginTransaction()` и `Session.commitTransaction()` для выполнения транзакции:

do()

```
async function explicitTcl(driver: Driver, ids: ThreeIds, logger: Logger) {
    logger.info('Running prepared query with explicit transaction control...');
    await driver.queryClient.do({
        fn: async (session) => {
            await session.beginTransaction({serializableReadWrite: {}});
            const [seriesId, seasonId, episodeId] = ids;
            const episode = new Episode({seriesId, seasonId, episodeId, title: '', airDate: new Date()});
            await session.execute({
                parameters: {
                    '$seriesId': episode.getTypedValue('seriesId'),
                    '$seasonId': episode.getTypedValue('seasonId'),
                    '$episodeId': episode.getTypedValue('episodeId')
```

```
                },
                text: `
                    DECLARE $seriesId AS Uint64;
                    DECLARE $seasonId AS Uint64;
                    DECLARE $episodeId AS Uint64;

                    UPDATE episodes
                    SET air_date = CurrentUtcDate()
                    WHERE series_id = $seriesId
                      AND season_id = $seasonId
                      AND episode_id = $episodeId;`
            })
            const txId = session.txId;
            await session.commitTransaction();
            logger.info(`TxId ${txId} committed.`);
        }
    });
}
```

doTx()

```
async function transactionPerWholeDo(driver: Driver, ids: ThreeIds, logger: Logger) {
    logger.info('Running query with one transaction per whole doTx()...');
    await driver.queryClient.doTx({
        txSettings: {serializableReadWrite: {}},
        fn: async (session) => {
            const [seriesId, seasonId, episodeId] = ids;
            const episode = new Episode({seriesId, seasonId, episodeId, title: '', airDate: new Date()});
            await session.execute({
                parameters: {
                    '$seriesId': episode.getTypedValue('seriesId'),
                    '$seasonId': episode.getTypedValue('seasonId'),
                    '$episodeId': episode.getTypedValue('episodeId')
                },
                text: `
                    DECLARE $seriesId AS Uint64;
                    DECLARE $seasonId AS Uint64;
                    DECLARE $episodeId AS Uint64;

                    UPDATE episodes
                    SET air_date = CurrentUtcDate()
                    WHERE series_id = $seriesId
                      AND season_id = $seasonId
                      AND episode_id = $episodeId;`
            })
            logger.info(`TxId ${session.txId} will be committed by doTx().`);
        }
    });
}
```

Обработка ошибок

Подробно об обработке ошибок написано в разделе Обработка ошибок в API.

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 reference/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, $episode_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 special 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 special 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)
```

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

Важно

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Доступные способы загрузки данных в колоночные таблицы:

- YDB CLI
- Пакетная вставка данных
- Yandex Data Transfer

Наполните данными созданные ранее строковые 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 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 Dave 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

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Доступные способы загрузки данных в колоночные таблицы:

- YDB CLI
- Пакетная вставка данных
- Yandex Data Transfer

Добавьте данные в строковые таблицы 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

Важно

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Доступные способы загрузки данных в колоночные таблицы:

- YDB CLI
- Пакетная вставка данных
- Yandex Data Transfer

Добавьте данные в строковую 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

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Доступные способы загрузки данных в колоночные таблицы:

- YDB CLI
- Пакетная вставка данных
- Yandex Data Transfer

Добавьте данные в строковую 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

Важно

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Доступные способы загрузки данных в колоночные таблицы:

- YDB CLI
- Пакетная вставка данных
- Yandex Data Transfer

Обновите данные в строковой 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

Важно

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Вместо `DELETE FROM` для удаления данных из колоночных таблиц можно воспользоваться механизмом удаления строк по времени — TTL. TTL можно задать при создании строковой или колоночной таблицы (`CREATE TABLE`) или при их изменении (`ALTER TABLE`).

Удалите данные из строковой 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 строковую или колоночную 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 строковую или колоночную таблицу:

```
ALTER TABLE episodes ADD COLUMN viewers Uint64;
```

Удалить колонку

Удалите добавленную колонку из строковой или колоночной таблицы:

```
ALTER TABLE episodes DROP COLUMN viewers;
```

Удаление таблиц

Удалите созданные строковые и/или колоночные таблицы с помощью конструкции DROP TABLE.

```
DROP TABLE episodes;
DROP TABLE seasons;
DROP TABLE series;
```

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 данные 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.
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.

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. Оценить необходимое количество партиций можно по предполагаемому объему добавляемых в таблицу данных. Средняя пропускная способность партиции на добавление данных — 1 МБ/с. При этом пропускная способность в первую очередь определяется выбранным первичным ключом (необходимостью сортировки данных внутри партиции при добавлении новых данных). Для небольших потоков данных не рекомендуется задавать больше 128 партиций.

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

При потоке данных в 1 ГБ/с оптимально использовать аналитическую таблицу с 1000 партиций. При этом создавать таблицы со значительным запасом партиций не стоит, т.к. это приведет к росту потребляемых ресурсов в кластере и итоговому замедлению скорости выполнения запросов.

Аутентификация

После успешной установки сетевого соединения сервер принимает к обработке запросы от клиента, в которых передается аутентификационная информация в виде аутентификационного токена. На ее 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 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, так называемый *search+bind* метод, состоящий из нескольких шагов. После получения логина и пароля пользователя, которого нужно аутентифицировать, выполняется операция *bind* с заранее определенными в секции ldap_authentication учетными данными специального сервисного аккаунта. Учетные данные такого сервисного аккаунта задаются через параметры конфигурации `bind_dn` и `bind_password`. После успешной аутентификации сервисного пользователя от его имени выполняется поиск в LDAP каталоге пользователя, который пытается аутентифицироваться в системе. Операция *search* выполняется для всего поддерева, корень которого берется из параметра конфигурации `base_dn`. Поиск записи в поддереве осуществляется в соответствии с фильтром, задаваемым в параметре конфигурации `search_filter`. После того как запись пользователя была найдена, YDB выполняет повторную операцию *bind* от лица этого найденного пользователя, используя ранее запрошенный пароль. Успех аутентификации пользователя определяется результатом повторной операции *bind*.

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.

Проверка токена

После аутентификации пользователя в системе, формируется токен, который проверяется перед выполнением запрошенной операции. В процессе проверки токена определяется, от имени какого пользователя запрашивается действие в системе и в каких группах он состоит. Для пользователей из LDAP каталога токен не содержит информацию о группах, поэтому, после проверки токена, выполняется еще один запрос к LDAP серверу для получения списка групп, в которых состоит пользователь.

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

В секции конфигурационного файла, описывающего аутентификационную информацию, можно настроить частоту обновления информации о пользователе и его группах. Этот показатель регулируется параметром `refresh_time`. Подробную информацию о файлах конфигурации можно узнать в разделе о конфигурировании кластера.

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 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`.

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`.

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, который содержит информацию обо всех операциях (успешных или не успешных) над объектами YDB:

- база данных — создание, изменение и удаление;
- директория — создание и удаление;
- таблица — создание, изменение схемы, изменение количества партиций, резервное копирование и восстановление, копирование и переименование, удаление;
- топик — создание, изменение и удаление;
- ACL — изменение.

Данные потока аудитного лога могут быть направлены:

- в файл на каждом узле кластера YDB;
- в агент для поставки метрик Unified Agent;
- в стандартный вывод ошибок `stderr`.

Можно использовать любое из перечисленных назначений или их комбинацию.

В случае направления в файл доступ к аудитному логу задается правами на уровне файловой системы. Сохранение аудитного лога в файл рекомендуется для использования в промышленных инсталляциях.

Направление аудитного лога в стандартный вывод ошибок `stderr` рекомендуется для тестовых инсталляций. Дальнейшая обработка данных потока определяется настройками логирования кластера YDB.

События аудитного лога

Информация о каждой операции записывается в аудитный лог в виде отдельного события. Каждое событие содержит набор атрибутов. Одни атрибуты являются общими для любых событий, другие определяются компонентом YDB, в котором произошло событие.

Атрибут	Описание
Общие атрибуты	
`subject`	SID источника события (формат `<login>@<subsystem>`). Если обязательная аутентификация не включена, атрибут будет иметь значение `{none}`. Обязательный.
`operation`	Название операции или действия, сходны с синтаксисом YQL (например, `ALTER DATABASE`, `CREATE TABLE`). Обязательный.
`status`	Статус завершения операции. Возможные значения: • `SUCCESS` — операция завершена успешно; • `ERROR` — операция завершена с ошибкой; • `IN-PROCESS` — операция выполняется. Обязательный.
`reason`	Сообщение об ошибке. Необязательный.
`component`	Имя компонента YDB — источника события (например, `schemeshard`). Необязательный.
`request_id`	Уникальный идентификатор запроса, вызвавшего операцию. По `request_id` можно отличать события разных операций и связывать события в единый аудитный контекст операции. Необязательный.
`remote_address`	IP-адрес клиента, приславшего запрос. Необязательный.
`detailed_status`	Статус, который передает компонент YDB (например `StatusAccepted`, `StatusInvalidParameter`, `StatusNameConflict`). Необязательный.
Атрибуты владения и разрешений	
`new_owner`	SID нового владельца объекта при передаче владения. Необязательный.

`acl_add`	Список добавленных разрешений при создании объектов или изменении разрешений в краткой записи (например, `[+R:someuser]`). Необязательный.
`acl_remove`	Список удаленных разрешений при изменении разрешений в краткой записи (например, `[-R:somegroup]`). Необязательный.
Пользовательские атрибуты	
`user_attrs_add`	Список добавленных пользовательских атрибутов при создании объектов или изменении атрибутов (например, `[attr_name1: A, attr_name2: B]`). Необязательный.
`user_attrs_remove`	Список удаленных пользовательских атрибутов при изменении атрибутов (например, `[attr_name1, attr_name2]`). Необязательный.
Атрибуты компонента SchemeShard	
`tx_id`	Уникальный идентификатор транзакции. Как и `request_id`, может быть использован для различения событий разных операций. Обязательный.
`database`	Путь базы данных (например, `/my_dir/db`). Обязательный.
`paths`	Список путей внутри БД, которые изменяет операция (например, `[/my_dir/db/table-a, /my_dir/db/table-b]`). Обязательный.

Включение аудитного лога

Отправка событий в поток аудитного лога включается целиком для кластера YDB. Для включения необходимо в конфигурацию кластера добавить секцию `audit_config`, указать в ней одно из назначений для потока (`file_backend`, `unified_agent_backend`, `stderr_backend`) или их комбинацию:

```
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
```

Ключ	Описание
`file_backend`	Сохранять аудитный лог в файл на каждом узле кластера.Необязательный.
`format`	Формат аудитного лога. Значение по умолчанию: `JSON`. Возможные значения: • `JSON` — JSON в сериализованном виде; • `TXT` — текст. Необязательный.
`file_path`	Путь к файлу, в который будет направлен аудитный лог. Путь и файл в случае их отсутствия будут созданы на каждом узле при старте кластера. Если файл существует, запись в него будет продолжена. Обязательный при использовании `file_backend`.
`unified_agent_backend`	Направить аудитный лог в Unified Agent. Необходимо также описать секцию `uaclient_config` в конфигурации кластера.Необязательный.

`log_name`	Метаданные сессии, которые передаются вместе с сообщением. Позволяют направить логирующий поток в один или несколько дочерних каналов по условию `_log_name: "session_meta_log_name"`. Необязательный.
`stderr_backend`	Направить аудитный лог в стандартный вывод ошибок `stderr`.Необязательный.

Пример конфигурации для сохранения аудитного лога в текстовом формате в файл `/var/log/ydb-audit.log`:

```
audit_config:
  file_backend:
    format: TXT
    file_path: "/var/log/ydb-audit.log"
```

Пример конфигурации для отправки аудитного лога в текстовом формате в Yandex Unified Agent с меткой `audit`, а также вывода его в `stderr` в формате JSON:

```
audit_config:
  unified_agent_backend:
    format: TXT
    log_name: audit
  stderr_backend:
    format: JSON
```

Примеры

Фрагмент файла аудитного лога в формате `JSON`:

```
2023-03-13T20:05:19.776132Z: {"paths":"
[/my_dir/db1/some_dir]","tx_id":"562949953476313","database":"/my_dir/db1","remote_address":"ipv6:
[xxxx:xxx:xxx:xxx:x:xxxx:xxx:xxxx]:xxxxx","status":"SUCCESS","subject":"
{none}","detailed_status":"StatusAccepted","operation":"CREATE DIRECTORY","component":"schemeshard"}
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"}
2023-03-13T19:59:27.614731Z: {"paths":"
[/my_dir/db1/some_table]","tx_id":"562949953426315","database":"/my_dir/db1","remote_address":"
{none}","status":"SUCCESS","subject":"{none}","detailed_status":"StatusAccepted","operation":"CREATE
TABLE","component":"schemeshard"}
2023-03-13T20:10:44.345767Z: {"paths":"[/my_dir/db1/some_table,
/my_dir/db1/another_table]","tx_id":"562949953506313","database":"{none}","remote_address":"ipv6:
[xxxx:xxx:xxx:xxx:x:xxxx:xxx:xxxx]:xxxxx","status":"SUCCESS","subject":"
{none}","detailed_status":"StatusAccepted","operation":"ALTER TABLE RENAME","component":"schemeshard"}
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:-]"}
```

Событие от `2023-03-13T20:05:19.776132Z` в JSON-pretty:

```
{
  "paths": "[/my_dir/db1/some_dir]",
  "tx_id": "562949953476313",
  "database": "/my_dir/db1",
  "remote_address": "ipv6:[xxxx:xxx:xxx:xxx:x:xxxx:xxx:xxxx]:xxxxx",
  "status": "SUCCESS",
  "subject": "{none}",
  "detailed_status": "StatusAccepted",
  "operation": "CREATE DIRECTORY",
  "component": "schemeshard"
}
```

Те же события в формате `TXT`:

```
2023-03-13T20:05:19.776132Z: component=schemeshard, tx_id=844424930186969, 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=StatusAccepted
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)
2023-03-13T19:59:27.614731Z: component=schemeshard, tx_id=562949953426315, remote_address={none}, subject={none},
database=/my_dir/db1, operation=CREATE TABLE, paths=[/my_dir/db1/some_table], status=SUCCESS,
detailed_status=StatusAccepted
2023-03-13T20:10:44.345767Z: component=schemeshard, tx_id=562949953506313, remote_address=ipv6:
[xxxx:xxx:xxx:xxx:x:xxxx:xxx:xxxx]:xxxxx, subject={none}, database={none}, operation=ALTER TABLE RENAME, paths=
[/my_dir/db1/some_table, /my_dir/db1/another_table], status=SUCCESS, detailed_status=StatusAccepted
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:-]
```

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.

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

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 PostgreSQL — для внешнего взаимодействия с клиентскими приложениями, изначально разработанными для работы с PostgreSQL.
  - Протокол Kafka — для внешнего взаимодействия с клиентскими приложениями, изначально разработанными для работы с Apache Kafka.
  - HTTP — для работы с встроенным 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 для 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 Настройка TLS.

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/token/ssh из yubikey/password и т.д. Рекомендуемый метод - ssh-ключи.
- Если у вас еще нет созданных ключей (или yubikey), то просто создайте новые ключи. Полные инструкции находятся на этой странице GitHub.
- Если у вас есть свои личные ключи, и вы используете skotty как ssh-agent:
  - Добавьте ключи в skotty командой ssh-add
  - Отредактируйте файл `~/.skotty/config.yaml`, добавив секцию:

```
keys_order:
    - added
    - insecure
    - legacy
    - secure
```

- Если у вас есть yubikey, вы можете использовать legacy ключ из yubikey:
  - Предположим, что у вас уже есть настроенный yubikey (или вы настроили yubikey локально)
  - На вашем ноутбуке: `skotty ssh keys`
  - Загрузите ssh-ключ `legacy@yubikey` на GitHub (через пользовательский интерфейс)
  - Проверьте подключение на ноутбуке: `ssh -T git@github.com`

Удаленная разработка

Если вы разрабатываете на удаленном компьютере, вы можете использовать ключ со своего ноутбука (сгенерированный или ключ от yubikey). Вам необходимо настроить переадресацию ключей. (Полные инструкции находятся на этой странице GitHub).

Предположим, что ваш удаленный компьютер dev123456.search.yandex.net.

- на вашем ноутбуке добавьте переадресацию по ssh (`~/.ssh/config`):

```
Host dev123456.search.yandex.net
    ForwardAgent yes
```

- на удаленном компьютере добавьте в `~/.bashrc`:

```
if [[ -S "$SSH_AUTH_SOCK" && ! -h "$SSH_AUTH_SOCK" ]]; then
    ln -sf "$SSH_AUTH_SOCK" ~/.ssh/ssh_auth_sock;
fi
export SSH_AUTH_SOCK=~/.ssh/ssh_auth_sock;
```

- проверьте подключение: `ssh -T git@github.com`

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'y

При создании Pull Request'a описание будет заполнено текстом из шаблона, который нужно отредактировать:

1. **Changelog Category.** В этом блоке нужно оставить одну категорию из списка, которой соответствует изменение (см. как выбрать категорию). Есть категории, для которых сообщения из Changelog Entry можно не заполнять, они не будут опубликованы в списке изменений. Остальные категории определяют раздел списка изменений, в который попадёт сообщение.
2. **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.
3. **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;
- содержать не менее 20 символов, если выбранная категория имеет раздел в списке изменений;
- опираться на термины, которые используются в глоссарии;
- описывать, что в работе системы изменилось для конечного пользователя;
- соответствовать дополнительным требованиям, определяемым выбранной категорией.

Как выбрать Changelog Category

Категория	Раздел в списке изменений	Описание	Требования к сообщению
Feature	Функциональность	Новая функциональность. Иногда выпускается под флагом для сохранения совместимости. Если вы делаете более одного PR'а в рамках задачи, эту категорию нужно выбрать только для финального коммита, а для промежуточных коммитов — Not for changelog.	Должно включать: • ссылку на документацию (если описание превышает два предложения); • описание того, как включить функциональность, если она выпускается под флагом (если описание превышает одно предложение, его также стоит вынести в документацию).

Experimental feature	Функциональность с пометкой «экспериментально»	Новая функциональность. Всегда выпускается под флагом для сохранения совместимости. Отличие от предыдущей категории в том, что даже после тестирования мы не можем гарантировать конечному пользователю корректную работу системы с этим изменением.	Должно включать: • ссылку на документацию (если описание превышает два предложения); • описание того, как включить функциональность (если описание превышает одно предложение, его также стоит вынести в документацию).
Improvement	Функциональность	Изменения существующей функциональности.	Должно включать ссылку на документацию (если описание превышает два предложения).
Performance improvement	Производительность	Изменения в производительности системы — ускорения, оптимизации, улучшения, которые привели к измеряемым изменениям в производительности.	Должно отвечать на вопросы: производительность какой части системы изменилась и насколько.
Bugfix	Исправления ошибок	Исправления ошибок, с которыми пользователь сталкивался или мог столкнуться в предыдущих релизных версиях.	Должно содержать: • ссылку на задачу в GitHub; • описание условий, при которых пользователь мог столкнуться с проблемой; • описание проявления проблемы; • описание способа исправления (необязательно, если логика работы системы принципиально не изменилась).
User Interface	YDB UI	Любые изменения в YDB UI	
Backward incompatible changes	Изменения с потерей обратной совместимости	Изменения с потерей обратной совместимости с предыдущими релизными версиями.	Должно содержать: • инструкцию по обновлению кластера без downtime приложений, которые используют изменяемую функциональность; • инструкцию по откату обновлений с кластера без downtime приложений.

Documentation	-	Изменения в документации.	Не будет опубликовано в истории изменений. Сообщение заполнять не нужно.
Not for Changelog	-	Изменения, которые не видны конечному пользователю — рефакторинг кода, тесты, CI/CD, изменения CHANGELOG.md, исправления ошибок в коде новой функциональности (которая не попала в предыдущие релизные версии).	Не будет опубликовано в истории изменений. Сообщение заполнять не нужно.

Примечание

Важно выбрать только одну категорию, поэтому не рекомендуется включать в один PR изменения из разных категорий, например, новую функциональность и исправление ошибки. Однако если так случилось, следует выбрать категорию более важного для пользователя изменения, а в блок Changelog Entry добавить описание всех изменений.

Сообщение из Changelog Entry будет опубликовано в CHANGELOG.md

После создания Pull Request будет запущена неблокирующая предварительная проверка, которая убеждается, что:

- описание к Pull Request написано;
- поле Changelog Category содержит одну категорию;
- поле Changelog Entry содержит не менее 20 символов, если у категории есть раздел в списке изменений.

Регулярно в `main` и `stable` ветках будет запущен процесс, который:

1. Собирает Changelog Entry из Pull Request, которые:

```
* замержены в ветку;
* относятся к категории, у которой есть [раздел в списке изменений](#choose_category);
* успешно прошли предварительную проверку.
```

2. Формирует Pull Request на добавление их в CHANGELOG.md.

На ревью этого Pull Request проверят, что выбрана верная категория и описание соответствует требованиям. Если описание не будет отвечать требованиям, то автора Pull Request попросят исправить сообщение. После ревью сообщение будет опубликовано в CHANGELOG.md. Перед публикацией новой версии YDB будет собрана история изменений на основании CHANGELOG.md и опубликована в документацию.

Где посмотреть CHANGELOG.md

Файл CHANGELOG.md нужен, чтобы:

- ознакомиться с изменениями, которые были сделаны в предыдущих релизах или планируются к выпуску в следующих;
- обновить описание вашего изменения (например, если вы хотите дополнить его ссылкой на документацию);
- после мержа изменения в текущий релиз перенести описание из блока Unreleased в блок нужной версии.

В `main` файл CHANGELOG.md хранится в корне. Чтобы открыть CHANGELOG.md в stable-ветке, выберите её из списка веток.
В `main` CHANGELOG.md содержит изменения для всех выпущенных версий и изменения, которые попадут в следующий релиз. В `stable-*` — изменения, которые были в `main` на момент ответвления, а также изменения, которые были домержены.

Как изменить сообщение в CHANGELOG.md

Если Pull Request ещё не замержен в ветку, можно изменить блок Changelog Entry в его описании.
Если Pull Request уже замержен, следует создать новый Pull Request, в котором изменить соответствующее сообщение в файле CHANGELOG.md этой ветки. Если от этой ветки уже были созданы другие stable-ветки, то в них тоже нужно внести соответствующее изменение — найдите последнюю мажорную ветку `stable-XX-Y`, а также все её минорные ветки `stable-XX-Y-Z`, и поправьте CHANGELOG.md в них.

Предварительные проверки

Перед мержем изменений выполняются прекоммитные проверки Pull Request'a.

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'a. 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
```

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/библиотека/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.

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**
>
> Раньше версия сервера YDB состояла из 3 чисел (например, v24.3.3), начиная с мажорной версии 25.1, добавлена четвертое число, которое обозначает номер патча (например, v25.1.1.3). Изменилась именование релизных веток и общая схема работы с ними - в мажорную релизную ветку можно мерджить новые фичи по согласованию, релизные теги создаются в минорных ветках, в которые можно мерджить только исправления ошибок. Подробнее про все изменения тут.

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, `23.1.2.1`).

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 feature-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

major release acceptance (nemesis, compatibil...

bug fix

feature

unsuccessful tag

unsuccessful tag

successful tag

bug fix needed ASAP in production (hotfix)

stable-25-1 **stable-25-1**

25-1-1-1 25-1-1-1

stable-25-1-2 **stable-25-1-2**

25-1-1-2

25-1-1-2

25-1-1-3 25-1-1-3

25-1-1-2-hotfix-1 25-1-1-2-hotfix-1

stable-25-1-1-2-hotfix **stable-25-1-1-2-hotfix**

stable-25-2 **stable-25-2**

stable-25-2-1 **stable-25-2-1**

stable-25-3 **stab**

stable-25-4Text is not SVG - cannot display

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.

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
- Документация GitHub
- Документация Git

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.

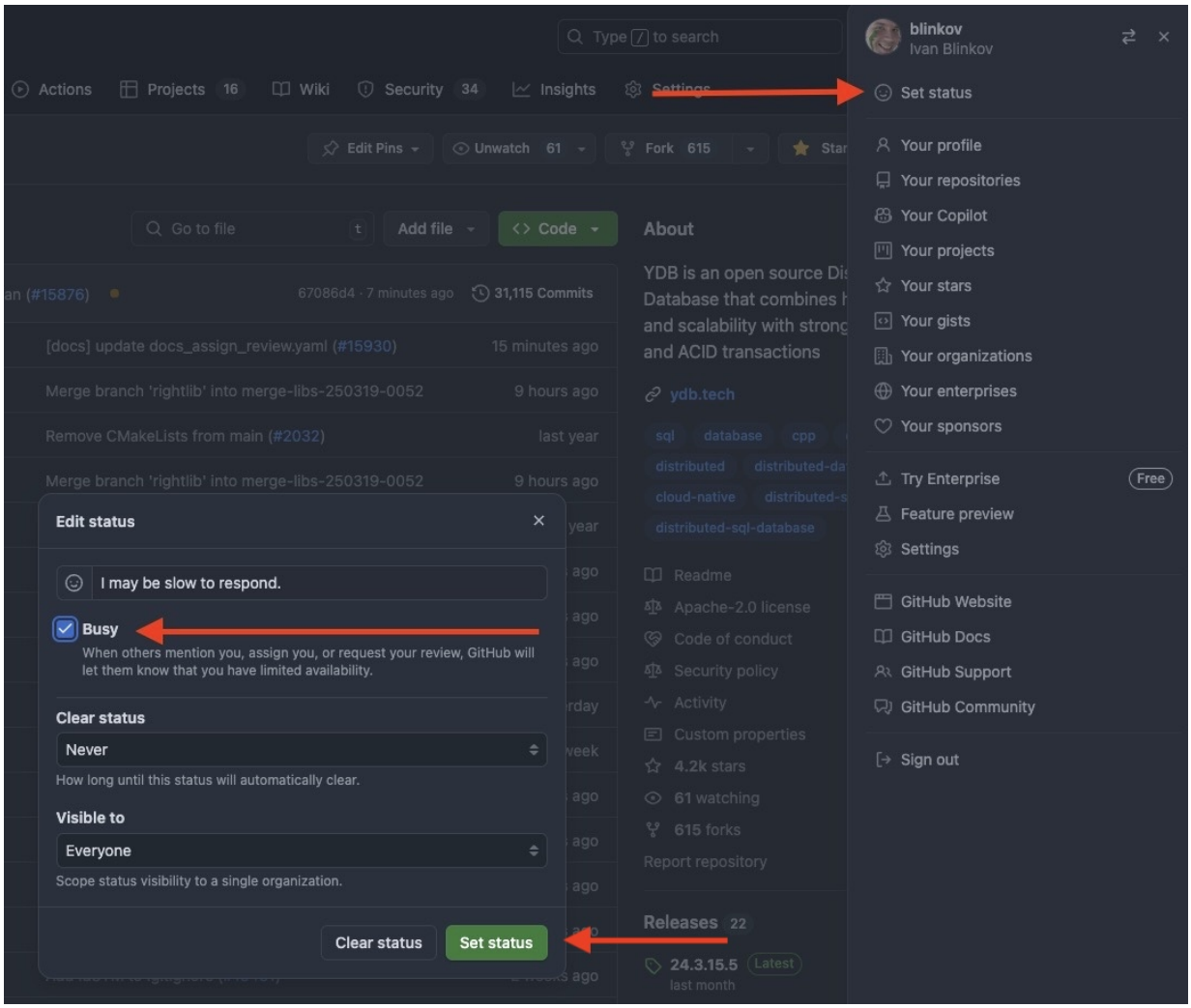
Роли

- **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

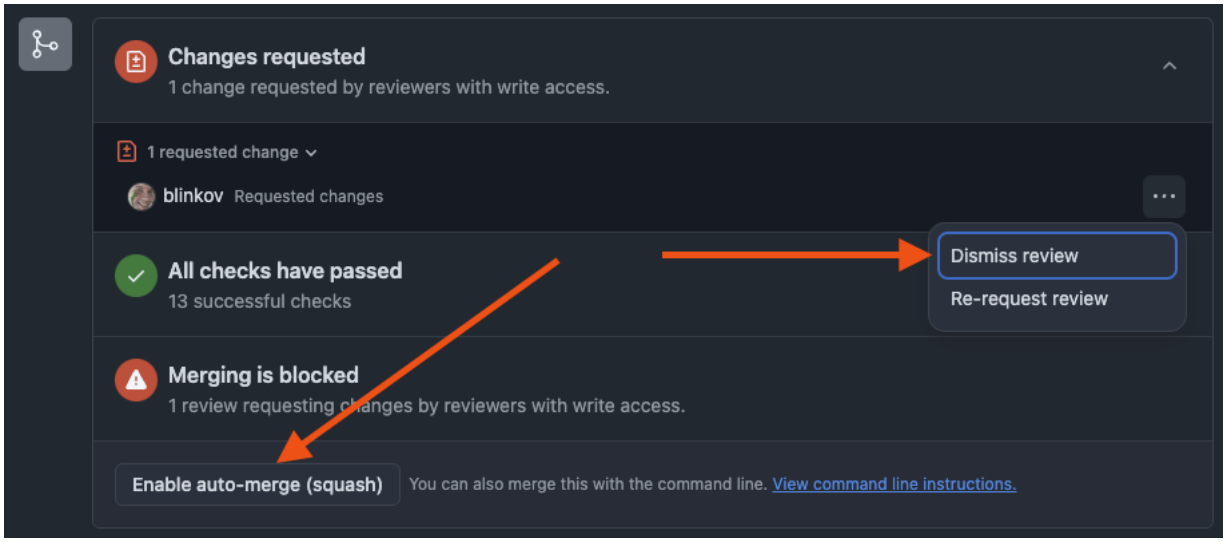
1. **Автор** открывает pull-запрос на GitHub с предложенными изменениями в папке ydb/docs. Следование руководству по стилю с самого начала делает процесс ревью более гладким.
2. **Автор** обеспечивает, чтобы pull-запрос был в состоянии, готовом к ревью, соответствуя всем следующим критериям:
   2.1. Pull-запрос имеет `* Documentation` как единственную категорию в changelog. Если всё сделано правильно, автоматизация пометит pull-запрос меткой «documentation».
   2.2. Pull-запрос *не* помечен как черновик.
   2.3. 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

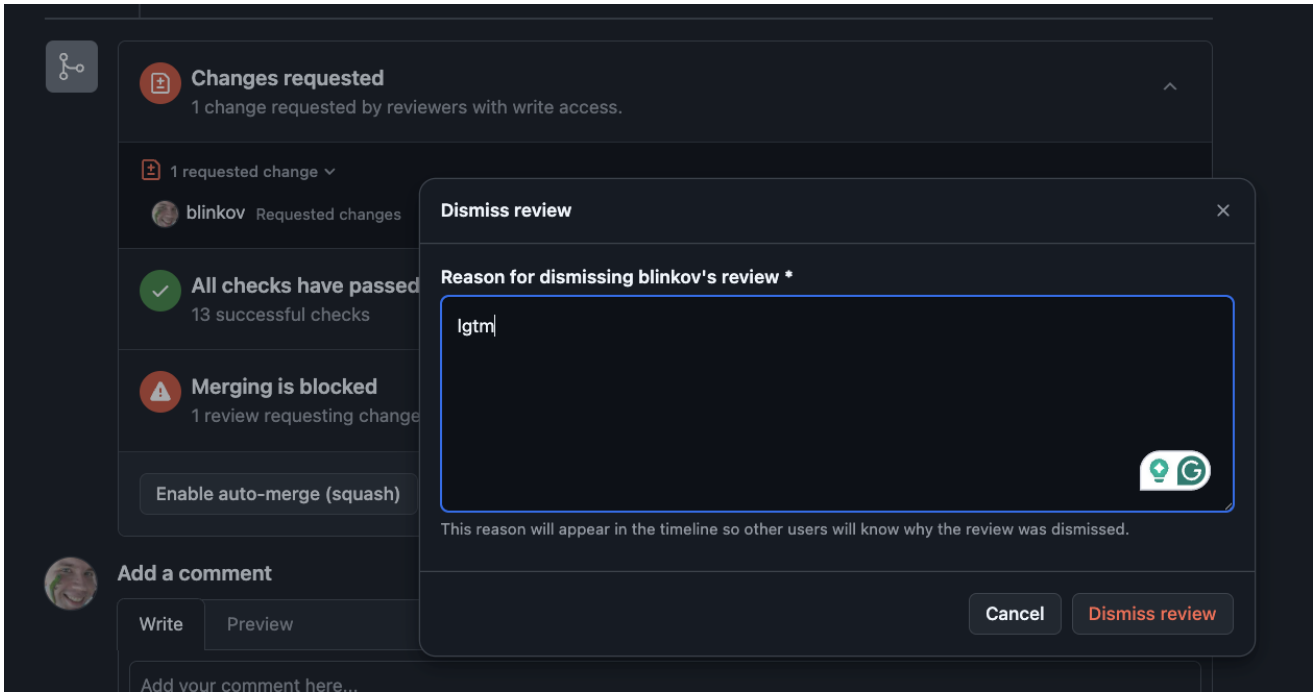


5. **Автор** и **основной ревьюер** совместно итеративно работают, пока предложенные изменения не пройдут контрольный список. **Основной ревьюер** предоставляет отзывы через комментарии к pull-запросу, а **автор** их учитывает. Ожидаемое время ответа для каждой итерации ревью составляет два рабочих дня, 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

- [ ] 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.

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, DevOps-инженеры, 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.
  - 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.
- **Для DevOps.** Папка для DevOps-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.** 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 Обзор 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 YDB для DevOps-инженеров, 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.

Общая схема YDB

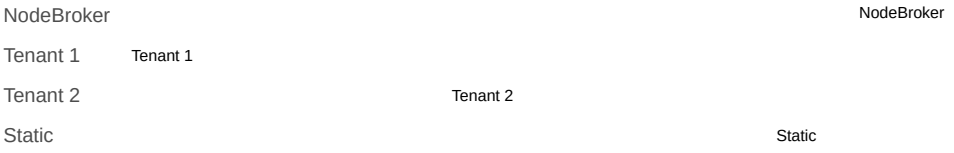
Примерная общая схема YDB представлена ниже.

gRPC proxy/KQP

DataShard

DataShard
DataShard
Hive
gRPC proxy/KQP
gRPC proxy/KQP
DataShard
DataShard
Dynamic
Node 1
Dynami...
Dynamic
Node 2
gRPC proxy/KQP
gRPC proxy/KQP

DataShard
DataShard
Hive
Hive
Dynami...
Dynamic
Node 3
gRPC proxy/KQP
gRPC proxy/KQP
BSC
BSC
Static...
Static
Node 1
gRPC proxy/KQP
gRPC proxy/KQP
CMS
CMS
Static...
Static
Node 2


NodeBroker
Tenant 1
Tenant 2
Static
NodeBroker
Tenant 1
Tenant 2
Static

BlobStorage
BlobStorage

NodeWarden
NodeWarden

NodeWarden

NodeWarden

PDisk

PDisk

PDisk

PDisk

VDisk

VDisk

VDisk

VDisk

DS proxy

DS proxy

DS proxy

DS proxy

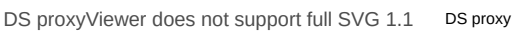
NodeWarden
NodeWarden

NodeWarden NodeWarden

NodeWarden

DS proxy

DS proxy



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 Дисковая подсистема кластера aka YDB distributed storage.

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).

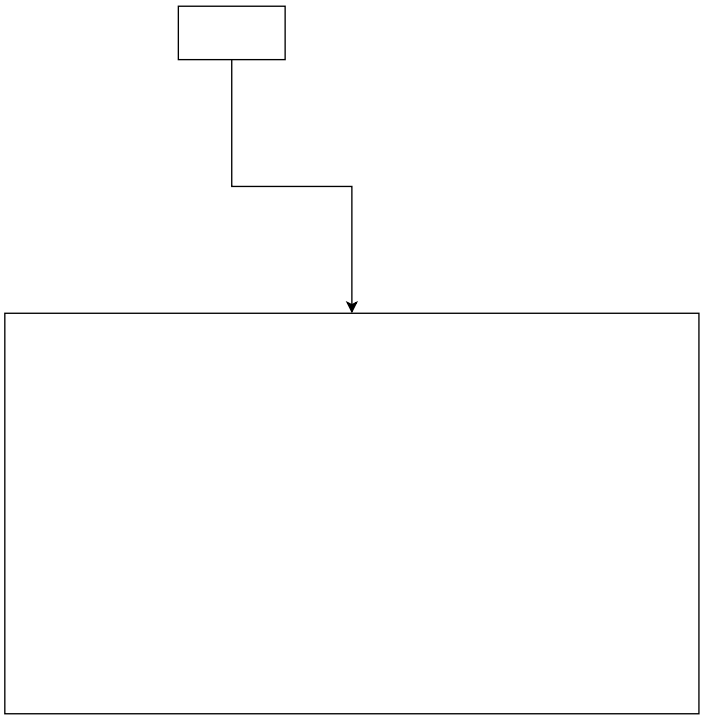
Блобы разных типов таблетка записывает в разные *каналы*. Канал указывает ветвь хранилища, в которой следует хранить блобы, и выполняет несколько функций, а именно:

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:



Step

Cookie Cookie

BlobSize BlobSize

PartId PartId

BlobId BlobId

64 64

8 8

32 32

32 32

24 24

28 (26) 28 (26)

4 4

TTableStorageInfo TableStorageInfo

Channel 0 Channel 0

Channel 1

Channel 1

Channel 2

Channel N

TTabletChannelInfo

TTabletChannelInfo

0≤gen<10

0≤gen<10

Group 12345

Group 12345

10≤gen<125

10≤gen<125

Group 54321

Group 54321

125≤gen

125≤gen

Group 12345

ТаблеткаViewer does not support full SVG 1.1

Таблетка

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.

Дисковая подсистема кластера aka YDB distributed storage

YDB distributed storage — это подсистема YDB, которая отвечает за надёжное хранение данных.

Она позволяет хранить *блобы* (бинарные фрагменты размером от 1 байта до 10 мегабайт) с уникальным идентификатором.

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, о котором будет рассказано ниже.

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.

PDisk 1:1000 PDisk 1:1000

PDisk 1:1001 PDisk 1:1001

PDisk 2:1000 PDisk 2:1000

PDisk 2:1001 PDisk 2:1001

PDisk 3:1000 PDisk 3:1000

PDisk 3:1001

PDisk 3:1001

PDisk 4:1000 PDisk 4:1000

PDisk 4:1001 PDisk 4:1001

Node 1 Node 1

Node 2 Node 2

Node 3

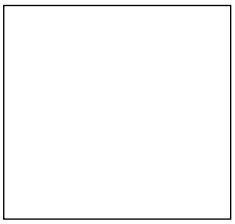
Node 4

DS proxyViewer does not support full SVG 1.1

DS proxy

Разноцветными квадратиками выделены VDisk'и разных групп; один цвет означает одну группу.

Группу можно рассматривать как совокупность VDisk'ов:



0

1

2

3

4

5

5

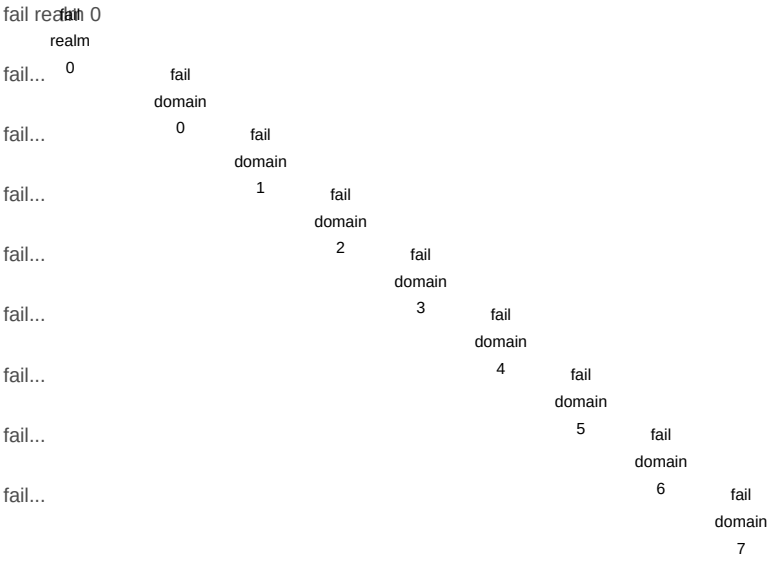
6

6

7

7

fail realm 0



VDISK[GroupId:Generation:0:6:0]

VDISK[GroupId:Generation:0:6:0]

0

0

1

2

2

3

3

4

4

5

5

6 6

7 7

8 8

fail
realm
0

fail realm 0

fail realm 1

fail realm 1

fail realm 2

fail realm 2

fail...

fail domain 0

fail...

fail domain 1

fail domain 2

fail...

VDISK[GroupId:Generation:1...Viewer does not support full VDISK[GroupId:Generation:1:2:0] 1_1

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*.

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 соответствуют 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.

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 `Hive/MAX(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 `Hive/MAX(BalanceScatter)` 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.

Максимальное из значений `ObjectImbalance` по разным объектам доступно в виде сенсора `Hive/MAX(BalanceObjectImbalance)` в подгруппе `tablets`.

Смотрите также

- Настройка Hive
- Hive web-viewer
- Запуск таблеток

Устройство механизма конфигурации V2

Конфигурация V2 в YDB реализует единый подход к управлению настройками кластера. Как пользоваться этим механизмом, описано в разделе для DevOps, а 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.

Внутренние процессы Distconf

- **Binding**
  Узлы хранения формируют ациклический граф. Каждый узел подключается к случайному соседнему таким образом, чтобы не было циклов. В итоге получается дерево из узлов с явно выделенной вершиной-корнем — узлом, который не может никуда подключиться, т.к. все уже подключены к нему. Корневой узел становится **лидером**, через который проходят все команды и который может принимать решения, в том числе рассылать команды через описанный далее механизм Scatter. Привязка узлов происходит через события `TEvInterconnect::TEvNodesInfo` и `TEvInterconnect::TEvNodeConnected` в `distconf_binding.cpp`.
- **Scatter/Gather**
  Лидер рассылает команды другим узлам по сформированному ранее графу (Scatter) и собирает результаты (Gather). Реализовано в `distconf_scatter_gather.cpp`, задачи рассылки и сбора ответов хранятся в структуре `TScatterTasks` и исполняются посредством методов `IssueScatterTask`, `CompleteScatterTask` и `AbortScatterTask`.
- **Finite State Machine (FSM)**
  В `distconf_fsm.cpp` реализован конечный автомат, непосредственно управляющий конфигурацией, её изменениями и их распространение по кластеру. Он отвечает за propose и commit изменений, а также смену лидера.
- **Persistent Storage**
  Локальная запись и чтение метаданных (`TPDiskMetadataRecord`) на PDisk'ах с обработкой событий `TEvStorageConfigLoaded` / `TEvStorageConfigStored` через `TDistributedConfigKeeper`.
- **InvokeOnRoot**
  Входная точка для команд администратора кластера: запросы `TEvNodeConfigInvokeOnRoot` обрабатываются в `distconf_invoke_common.cpp`, ответы возвращаются в виде `TEvNodeConfigInvokeOnRootResult`.
- **Dynamic**
  Узлы баз данных подписываются на события `TEvNodeWardenDynamicConfigPush` через `ConnectToStaticNode`, чтобы получать обновления конфигурации в реальном времени.
- **Self-Heal**
  При использовании Distconf для статической группы работает Self-Heal по аналогии с динамическими группами.
- **Контроллер распределённого хранилища**
  DS-controller получает от Distconf изменения в конфигурации и использует их для работы распределённого хранилища.
- **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

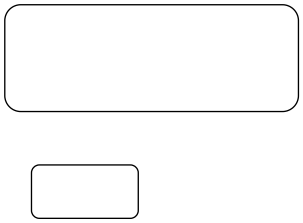
Обзор тестирования с помощью нагружающих акторов

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:



PDisk

PDisk

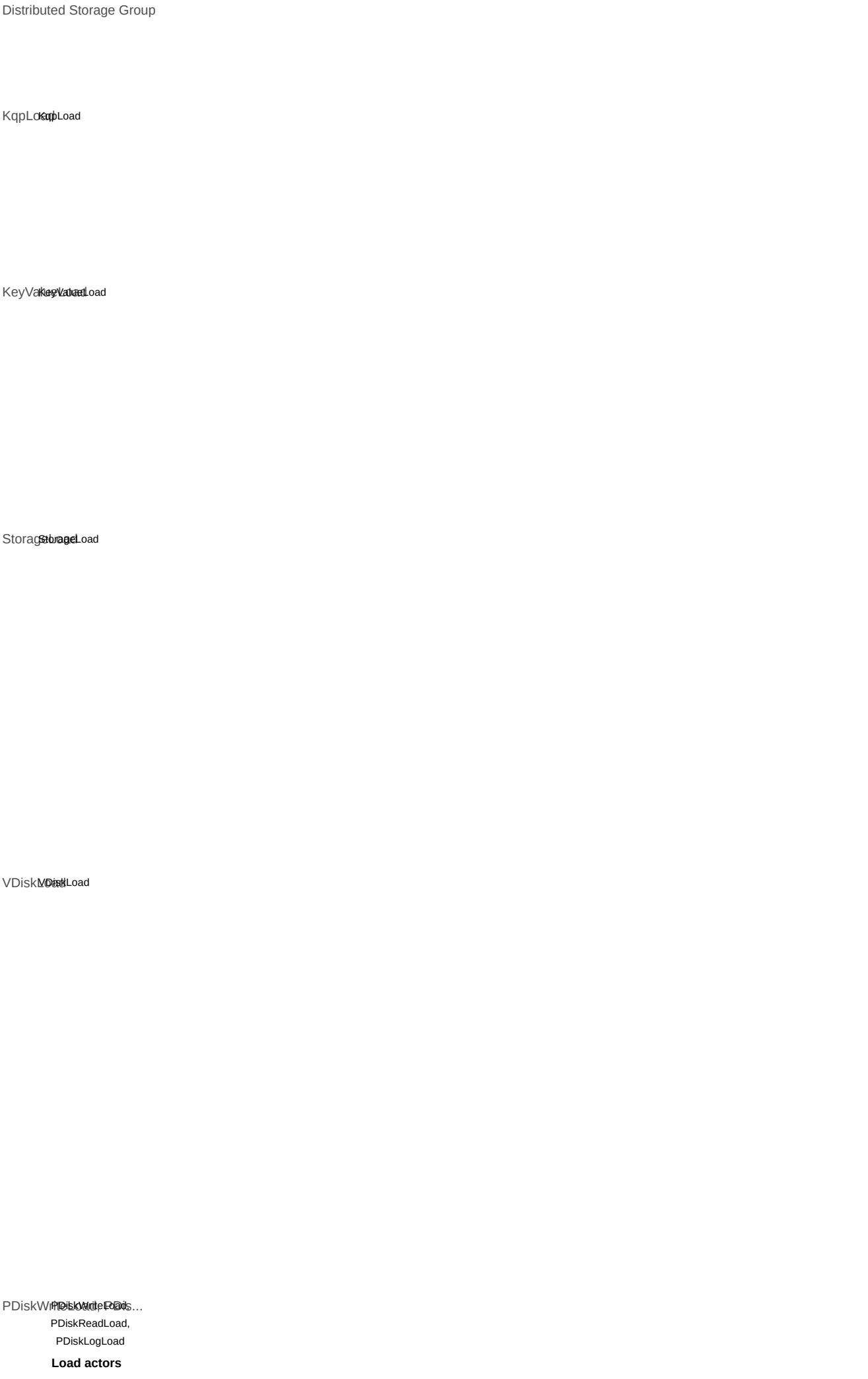
VDisk VDisk

Device Device

Key-value... Key-value
Tablet

Distributed Storage Proxy

Distributed Storage Proxy
Data Shard...
Data Shard
Tablet
Query Processor
Query Processor
VDisk
VDisk
PDisk
PDisk
Distributed Storage Group


Distributed Storage Group
KqpLoad
KqpLoad
KeyValueLoad
KeyValueLoad
StorageLoad
StorageLoad
VDiskLoad
VDiskLoad
PDiskWriteLoad, PDi...
PDiskWriteLoad,
PDiskReadLoad,
PDiskLogLoad
Load actors

Load actors

YDB components **YDBcomponents**

DeviceText is not SVG - cannot display Device

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.
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:

- Embedded 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.

Embedded 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 Embedded UI. Описание выводимых параметров смотрите в документации соответствующего актора.

Embedded 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

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 ключами `(RowId, 2000, 11)` и `(RowId, 1000, 10)`, соответственно. Об их наличии которых 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.

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.

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 KQP, 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 KQP 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 KQP 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. KQP 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 KQP. 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 KQP 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 KQP 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, KQP 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 KQP старается аккумулировать изменения в памяти либо до момента их коммита, либо до необходимости чтения этих изменений в той же транзакции. Далее KQP 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).

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 KQP 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 KQP 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) KQP 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 KQP 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 KQP 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. Учатники, 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.

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.

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;
}
```

StorageLoad

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 `InFlightWrites`. 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 `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
`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.
`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.

`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` – взаимосключающие.
`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.

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]`.

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`.

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 }
```

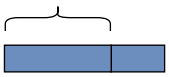
```
    Chunks: { Slots: 4096 Weight: 1 }
    Chunks: { Slots: 4096 Weight: 1 }
    Chunks: { Slots: 4096 Weight: 1 }
    Chunks: { Slots: 4096 Weight: 1 }
    DurationSeconds: 120
    IntervalMsMin: 0
    IntervalMsMax: 0
    InFlightReads: 64
    Sequential: false
    IsWardenlessTest: false
}
```

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`.

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.



SizeInterv...

SizeInterval
VDisk1

BurstSize VDisk1

BurstSize VDisk1

BurstInter...

BurstInterval
VDisk1

StorageDuration VDisk1

StorageDuration VDisk1

SizeInterval
VDisk2

SizeInterv...

BurstInter...

BurstInterval
VDisk2

StorageDuration VDisk2

StorageDuration VDisk2

BurstSize VDisk2

Data that is being deleted

Data that is being dele...

Timeline,...Text is not SVG - cannot display

Timeline,
bytes

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 Embedded UI кластера.

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
}
```

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 Embedded 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
}
```

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.

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 сетевым протоколам PostgreSQL или Apache Kafka. Благодаря слою совместимости, множество инструментов, разработанных для работы с этими системами, могут также взаимодействовать с YDB. Уровень совместимости каждого конкретного приложения необходимо уточнять отдельно.

Графические пользовательские интерфейсы

Среда	Инструкция	Уровень поддержки
Встроенный 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
Jupyter Notebook	Инструкция	С помощью YDB-SQLAlchemy

Визуализация данных (Business Intelligence, BI)

Среда	Уровень поддержки	Инструкция
Apache Superset	Через PostgreSQL-совместимость	Инструкция
DataLens	Полный	Инструкция
FineBI	Через PostgreSQL-совместимость	Инструкция
Grafana	Полный	Инструкция

Оркестрация

Среда	Инструкция
Apache Airflow™	Инструкция

Поставка данных

Система поставки	Инструкция
FluentBit	Инструкция
LogStash	Инструкция
Kafka Connect Sink	Инструкция
Произвольные JDBC-источники данных	Инструкция

Потоковая поставка данных

Система поставки	Инструкция
Apache Kafka API	Инструкция
Apache Kafka Connect	Инструкция

Миграции данных

Среда	Инструкция
goose	Инструкция
Liquibase	Инструкция
Flyway	Инструкция

Объектно-реляционное отображение (ORM)

Система	Инструкция
Hibernate	Инструкция
Spring Data JDBC	Инструкция
JOOQ	Инструкция
Dapper	Инструкция
Entity Framework	Инструкция

Векторный поиск

Инструмент	Инструкция
LangChain	Инструкция

Смотрите также

- Справка по YDB SDK
- Совместимость YDB с PostgreSQL
- Kafka API

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.

Справка по YDB SDK

Для работы с YDB доступны OpenSource SDK для следующих языков программирования:

Язык	Репозиторий на GitHub	Справка по API
C++	ydb-platform/ydb/tree/main/ydb/public/sdk/cpp	N/A
C# (.NET)	ydb-platform/ydb-dotnet-sdk	N/A
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
Node.js	ydb-platform/ydb-nodejs-sdk	N/A
PHP	ydb-platform/ydb-php-sdk	N/A
Python	ydb-platform/ydb-python-sdk	https://ydb-platform.github.io/ydb-python-sdk
Rust	ydb-platform/ydb-rs-sdk	N/A

Документация по SDK содержит следующие разделы:

- Установка
- Аутентификация
- Обработка ошибок
- Сравнение возможностей SDK
- Параметризованные запросы
- Работа с топиками
- Работа с узлами координации

Смотрите также:

- Документация для разработчиков приложений
- Тестовые приложения
- Рецепты кода

Языки и API

- ADO.NET — доступ к YDB из .NET
- JDBC-драйвер для YDB
- YDB Model Context Protocol Server

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

Параметры конфигурации кластера

В данной статье приведено описание основных групп конфигурируемых параметров в файле конфигурации кластера.

host_configs — типовые конфигурации хостов

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>
  - 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 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)

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
```

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 — статические узлы кластера

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 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.

log_config — конфигурация логирования

Секция `log_config` управляет тем, как YDB обрабатывает и управляет своими логами. Она позволяет настраивать уровни логирования, форматы и места назначения для различных компонентов.

Подробную информацию о параметрах конфигурации логирования смотрите в Секция конфигурации `log_config`.

Топология кластера

Для конфигурации топологии кластера YDB доступны следующие режимы отказоустойчивости:

Режим	Описание
`none`	Избыточность отсутствует. Применяется для тестирования.
`block-4-2`	Избыточность с коэффициентом 1,5, применяется для однодатацентровых кластеров.
`mirror-3-dc`	Избыточность с коэффициентом 3, применяется для мультидатацентровых кластеров.

Топология кластера задается в качестве параметра `erasure` и принимает одно из значений, указанных в таблице выше.

Конфигурация Blob Storage

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

```
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

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 учетом 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 3 работающих узла (из 5) остается при сбое двух доменов.
Геораспределенный	9	Выберите три узла в разных доменах отказа внутри каждой из трех зон доступности пула хранения mirror-3-dc для гарантии, что большинство в 5 работающих узлов (из 9) остается при сбое зоны доступности + домена отказа.

При развертывании 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 нечетные числа, так как использование четных чисел не улучшает отказоустойчивость по сравнению с ближайшим меньшим нечетным числом.

domains_config — домен кластера

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.
Георасределенный	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.

Примеры

`block-4-2`

```
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
```

`mirror-3-dc`

```
domains_config:
  domain:
  - name: global
    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
```

`none` (без отказоустойчивости)

```
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 пулов

```
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
```

Настройка акторной системы

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 Embedded UI.

Автоматическое конфигурирование

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.

Контроллер памяти

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. 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.
- KQP: хранит промежуточные результаты обработки запросов.
- 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:

- KQP.

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.

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 KQP:

```
memory_controller_config:
  query_execution_limit_percent: 25
```

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 KQP.

blob_storage_config — статическая группа кластера

Укажите конфигурацию статической группы кластера. Статическая группа необходима для работы базовых таблеток кластера, в том числе `Hive`, `SchemeShard`, `BlobstorageContoller`.
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 нодах, 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 AZ необходимо указать 3 кольца. Для конфигураций, расположенных в одной AZ, указывается ровно одно кольцо.

Настройка стабильных имен узлов кластера

Присвоение имен узлам осуществляет Node Broker — системная таблетка, которая отвечает за регистрацию динамических узлов в кластере YDB.

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, необходимо добавить в конфигурацию кластера следующее:

```
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 — брокер ресурсов

Брокер ресурсов — это 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`	2	—	Операции восстановления из резервной копии.
`queue_build_index`	10	—	Операции онлайн-создания вторичного индекса.
`queue_cdc_initial_scan`	4	—	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
```

Настройка Health Check

В этом разделе настраиваются пороговые значения и таймауты, используемые сервисом Health Check YDB. Эти параметры помогают настраивать возможные проблемы, такие как чрезмерные перезапуски или расхождение по времени между динамическими узлами.

Синтаксис

```
healthcheck_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)

Примеры конфигураций кластеров

В репозитории можно найти модельные примеры конфигураций кластеров для самостоятельного развертывания. Ознакомьтесь с ними перед развертыванием кластера.

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

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.
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 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.

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

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.

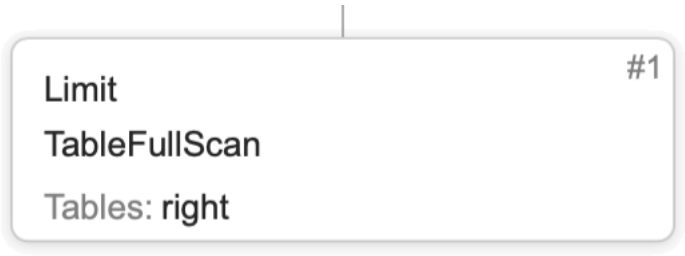
Ниже представлена справочная информация по типам узлов, а пример анализа конкретного плана запроса можно посмотреть здесь.

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

Внимание

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

Внимание

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

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.

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 быть использован в первичном ключе
`Double`	Вещественное число с переменной точностью размером 8 байт.	Не может быть использован в первичном ключе

`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.	

Строковые типы

Тип	Описание	Примечания
`String`	Строка, может содержать произвольные бинарные данные	
`Utf8`	Текст в кодировке UTF-8	
`Json`	JSON в текстовом представлении	Не поддерживает возможность сравнения, не может быть использован в первичном ключе
`JsonDocument`	JSON в бинарном индексированном представлении	Не поддерживает возможность сравнения, не может быть использован в первичном ключе
`Yson`	YSON в текстовом или бинарном представлении	Не поддерживает возможность сравнения, не может быть использован в первичном ключе
`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	Примечания
`Date`	Дата, точность до дней	Диапазон значений для всех временных типов кроме `Interval` - от нуля часов 01.01.1970 до нуля часов 01.01.2106. Внутреннее представление `Date` – беззнаковое целое 16 бит
`Datetime`	Дата/время, точность до секунд	Внутреннее представление – беззнаковое целое 32 бит
`Timestamp`	Дата/время, точность до микросекунд	Внутреннее представление – беззнаковое целое 64 бит
`Interval`	Интервал времени (знаковый), точность до микросекунд	Диапазон значений – от -136 лет до +136 лет. Внутреннее представление – знаковое целое 64 бит. Не может быть использован в первичном ключе
`TzDate`	Дата с меткой временной зоны, точность до дней	Не поддерживается в столбцах таблиц
`TzDateTime`	Дата/время с меткой временной зоны, точность до секунд	Не поддерживается в столбцах таблиц
`TzTimestamp`	Дата/время с меткой временной зоны, точность до микросекунд	Не поддерживается в столбцах таблиц

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 численным 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
Utf8	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

[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
Bool	Нет	Нет	Нет	Нет
Int8	Да	Да	Да	Да
Int16	Да	Да	Да	Да
Int32	Да	Да	Да	Да
Int64	Да	Да	Да	Да
Uint8	Да	Да	Да	Да
Uint16	Да	Да	Да	Да
Uint32	Да	Да	Да	Да
Uint64	Да	Да	Да	Да
Float	Нет	Нет	Нет	Нет

Double	Нет	Нет	Нет	Нет
Decimal	Нет	Нет	Нет	Нет
String	Да	Да	Да	Да
Utf8	Да	Да	Да	Да
Json	Нет	Нет	Нет	Нет
Yson	Нет	Нет	Нет	Нет
Uuid	Нет	Нет	Нет	Нет
Date	—	Да	Да	Нет
Datetime	Да	—	Да	Нет
Timestamp	Да	Да	—	Нет
Interval	Нет	Нет	Нет	—

Приведение к другим типам данных

Тип	String	Utf8	Json	Yson	Uuid
Bool	Да	Нет	Нет	Нет	Нет
Int8	Да	Нет	Нет	Нет	Нет
Int16	Да	Нет	Нет	Нет	Нет
Int32	Да	Нет	Нет	Нет	Нет
Int64	Да	Нет	Нет	Нет	Нет
Uint8	Да	Нет	Нет	Нет	Нет
Uint16	Да	Нет	Нет	Нет	Нет
Uint32	Да	Нет	Нет	Нет	Нет
Uint64	Да	Нет	Нет	Нет	Нет
Float	Да	Нет	Нет	Нет	Нет
Double	Да	Нет	Нет	Нет	Нет
Decimal	Да	Нет	Нет	Нет	Нет
String	—	Да	Да	Да	Да
Utf8	Да	—	Нет	Нет	Нет
Json	Да	Да	—	Нет	Нет
Yson	Да[1]	Нет	Нет	Нет	Нет
Uuid	Да	Да	Нет	Нет	—
Date	Да	Да	Нет	Нет	Нет
Datetime	Да	Да	Нет	Нет	Нет
Timestamp	Да	Да	Нет	Нет	Нет
Interval	Да	Да	Нет	Нет	Нет

[1] При помощи 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:

Численные типы

При несовпадении численных 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
Date	—	—	—	`Date`	—	—	—
Datetime	—	—	—	`Datetime`	—	—	—
Timestamp	—	—	—	`Timestamp`	—	—	—
Interval	`Date`	`Datetime`	`Timestamp`	—	`TzDate`	`TzDatetime`	`TzTimestamp`
TzDate	—	—	—	`TzDate`	—	—	—
TzDatetime	—	—	—	`TzDatetime`	—	—	—
TzTimestamp	—	—	—	`TzTimestamp`	—	—	—

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).

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, code: 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"`.

Yson

В данном разделе собрана информация про YSON — JSON-подобный формат данных, разработанный в Яндексе.

> **Примечание**
>
> SQL-функции для работы с YSON описаны здесь

Введение

К основным **отличиям YSON от JSON** относится:

1. Поддержка бинарного представления скалярных типов (чисел, строк и булевого типа);
2. Атрибуты: произвольный словарь, который можно установить дополнительно на литерал любого (даже скалярного) типа.

Кроме того существуют **синтаксические отличия**:

1. Вместо запятой в качестве разделителя используется точка с запятой;
2. В словарях ключ от значения отделяется не двоеточием, а знаком равенства: `=`;
3. Строковые литералы не обязательно всегда заключать в кавычки (только если иначе возникает неоднозначность при парсинге).

Имеется следующий набор **скалярных** типов:

1. Строки (`string`);
2. Знаковые и беззнаковые 64-битные целые числа (`int64` и `uint64`);
3. Числа с плавающей точкой двойной точности (`double`);
4. Булев (логический) тип (`boolean`);
5. Специальный тип entity, имеющий всего один литерал (`#`).

Скалярные типы обычно имеют как текстовое, так и бинарное представление.

Есть два **композитных** типа:

1. Список (`list`);
2. Словарь (`map`).

Скалярные типы

Строки

Токены строк бывают трех видов:

1. **Идентификаторы** задаются регулярным выражением `[A-Za-z_][A-Za-z0-9_.\-]*` (Первый символ - буква или нижнее подчеркивание, со второго символа могут быть дополнительно использованы цифры и символы `-`, `.`). Идентификатор задает строку с идентичным ему содержимым и используется в первую очередь для краткости (не нужно ставить кавычки).

   Примеры:

   - `abc123`;
   - `_`;
   - `a-b`.
2. **Текстовые строки** — C-escaped строки в двойных кавычках.

   Примеры:

   - `"abc123"`;
   - `""`;
   - `"quotation-mark: \", backslash: \\, tab: \t, unicode: \xEA"`.
3. **Бинарные строки**: `\x01 + length (protobuf sint32 wire format) + data (<length> bytes)`.

Знаковые 64-битные целые числа (`int64`)

Два способа записи:

1. **Текстовый**: (`0`, `123`, `-123`, `+123`);
2. **Бинарный**: `\x02 + value (protobuf sint64 wire format)`.

Беззнаковые 64-битные целые числа (`uint64`)

Два способа записи:

1. **Текстовый**: (`10000000000000`, `123u`);

2. **Бинарный**: `\x06 + value (protobuf uint64 wire format)`.

Числа с плавающей точкой (`double`)

Два способа записи:

1. **Текстовый**: (`0.0`, `-1.0`, `1e-9`, `1.5E+9`, `32E1`, `%inf`, `%-inf`, `%nan`);
2. **Бинарный**: `\x03 + protobuf double wire format`.

> **Важно**
>
> Текстовое представление чисел с плавающей точкой включает в себя округление, которое может привести к тому, что при обратном парсинге значение окажется иным. В случае, если вам важна точность, следует использовать бинарное представление.

> **Важно**
>
> Значения `%inf`, `%-inf`, `%nan` не существуют в JSON, поэтому вызов `Yson::SerializeJson` с содержащим эти значения YSON приведет к ошибке

Булевы литералы (`boolean`)

Два способа записи:

1. **Текстовый** (`%false`, `%true`);
2. **Бинарный** (`\x04`, `\x05`).

Entity (`entity`)

Entity представляет собой атомарное скалярное значение, не имеющее никакого собственного содержимого. Сценарии, в которых данный тип может быть полезен, разнообразны. Например, часто `entity` обозначает `null`. При этом `entity` может иметь аттрибуты

Лексически entity кодируется символом решетки: `#`.

Выделенные литералы

Специальные токены:

`;`, `=`, `#`, `[`, `]`, `{`, `}`, `<`, `>`, `)`, `/`, `@`, `!`, `+`, `^`, `:`, `,`, `~`.

Не все эти символы используются в YSON, некоторые используются в YPath.

Композитные типы

Список (`list`)

Задается следующим образом: `[value; ...; value]`, где `value` — литералы произвольных скалярных или композитных типов.

Пример: `[1; "hello"; {a=1; b=2}]`.

Словарь (`map`)

Задается следующим образом: `{key = value; ...; key = value}`. Здесь `*key*` — литералы строкового типа, а `value` — литералы произвольных скалярных или композитных типов.

Пример: `{a = "hello"; "38 parrots" = [38]}`.

Атрибуты

На любой литерал в YSON можно установить **атрибуты**. Записывается это так: `<key = value; ...; key = value> value`. Внутри угловых скобок синтаксис аналогичен словарю. Например, `<a = 10; b = [7,7,8]>"some-string"` или `<"44" = 44>44`. Но чаще всего атрибуты можно встретить на литералах типа `entity`, например, `<id="aaad6921-b5704588-17990259-7b88bad3">#`.

Грамматика

YSON-данные бывают трех типов:

1. **Node** (одно дерево, в примере — `<tree>`)
2. **ListFragment** (значения, разделенные `;`, в примере — `<list-fragment>`)
3. **MapFragment** (пары ключ-значение, разделенные `;`, в примере — `<map-fragment>`)

Грамматика (определяется с точностью до пробельных символов, которые могут быть в произвольном количестве добавлены и удалены между токенами):

```
         <tree> = [ <attributes> ], <object>;
       <object> = <scalar> | <map> | <list> | <entity>;

       <scalar> = <string> | <int64> | <uint64> | <double> | <boolean>;
         <list> = "[", <list-fragment>, "]";
          <map> = "{", <map-fragment>, "}";
       <entity> = "#";
   <attributes> = "<", <map-fragment>, ">";

<list-fragment> = { <list-item>, ";" }, [ <list-item> ];
    <list-item> = <tree>;

 <map-fragment> = { <key-value-pair>, ";" }, [ <key-value-pair> ];
<key-value-pair> = <string>, "=", <tree>;  % Key cannot be empty
```

Символ `;` после последнего элемента внутри `<list-fragment>` и `<map-fragment>` может быть опущен. Следующие конструкции следует считать валидными при чтении:

С `;` на конце	Сокращенная запись

`<a=b;>c`
`{a=b;}`
`1;2;3;`
	`<a=b>c`
	`{a=b}`
	`1;2;3`

Примеры

- Map (Node)

```
{ performance = 1 ; precision = 0.78 ; recall = 0.21 }
```

- Map (Node)

```
{ cv-precision = [ 0.85 ; 0.24 ; 0.71 ; 0.70 ] }
```

- List (Node)

```
[ 1; 2; 3; 4; 5 ]
```

- String (Node)

```
foobar
```

```
"hello world"
```

- Int64 (Node) `42`
- Double (Node) `3.1415926`
- ListFragment

```
{ key = a; value = 0 };
{ key = b; value = 1 };
```

```
{ key = c; value = 2; unknown_value = [] }
```

- MapFragment

```
do = create; type = table; scheme = {}
```

- HomeDirectory (Node)

```
{ home = { sandello = { mytable = <type = table> # ; anothertable = <type = table> # } ; monster = { } } }
```

YPATH

В данном разделе собрана информация про YPath — язык, описывающий пути к объектам в YSON.

YPath представляет собой язык описания путей, которые идентифицирует объекты в YSON. Язык позволяет обращаться к узлам и указывать аннотации, которые могут быть полезны при совершении операций над узлами, такими как запись и чтение свойств.

Например:

- `/0-25-3ec012f-406daf5c/@type` — путь к атрибуту `type` объекта с идентификатором `0-25-3ec012f-406daf5c`;

Существует несколько разновидностей YPath. В самом простом случае YPath представляет собой строку, кодирующую путь.

Лексика

Строка, кодирующая простой YPath, разбивается на следующие **токены**:

1. **Специальные символы**: прямой слеш (`/`), "собака" (`@`), амперсанд (`&`), звездочка (`*`);
2. **Литералы**: максимальная непустая последовательность неспециальных символов. В литералах разрешен escaping вида `\<escape-sequence>`, где в качестве `<escape-sequence>` может выступать один из символов `\`, `/`, `@`, `&`, `*`, `[`, `{`, а также выражение вида `x<hex1><hex2>`, где `<hex1>` и `<hex2>` — шестнадцатеричные цифры.

Синтаксис и семантика

Структурно YPath имеет вид `/<relative-path>`. `<relative-path>` разбирается последовательно слева направо, в результате чего возникают шаги перемещения по дереву следующих видов:

- **Переход к потомку**: последовательность из токена `/` и литерала.
  Данный тип шагов применим к словарям и спискам. В случае словаря литерал должен содержать имя потомка. Пример: `/child` — переход к потомку с именем `child`.
  В случае списка литерал должен содержать целое число в десятичной системе счисления — номер потомка. Потомки в списке нумеруются с нуля. Разрешены также отрицательные номера, которые нумеруют потомков с конца списка. Примеры: `/1` — переход ко второму потомку в списке, `/-1` — переход к последнему потомку в списке;
- **Переход к атрибуту**: последовательность из токенов `/@` и литерала.
  Данный тип шагов применим в любой точке пути и означает переход к атрибуту с данными именем. Пример: `/@attr` — переход к атрибуту с именем `attr`.

> **Примечание**
>
> В YPath относительные пути начинаются со слешей. Тем самым, слеш служит не разделителем (как в случае файловых систем), а полноправным членом команды перемещения по дереву. В частности, для склейки двух YPath достаточно обычной конкатенации строк. Это свойство может показаться необычным, но во многих местах оно удобно, и к нему достаточно легко привыкнуть.

Примеры

```
$data = Yson(@@{"0-25-3ec012f-406daf5c" = {a=<why="I can just do it">1;b=2}}@@);
SELECT Yson::SerializeJson($data), Yson::SerializeJson(Yson::YPath($data, "/0-25-3ec012f-406daf5c/a/@/why"));
```

Результат:

column0	column1

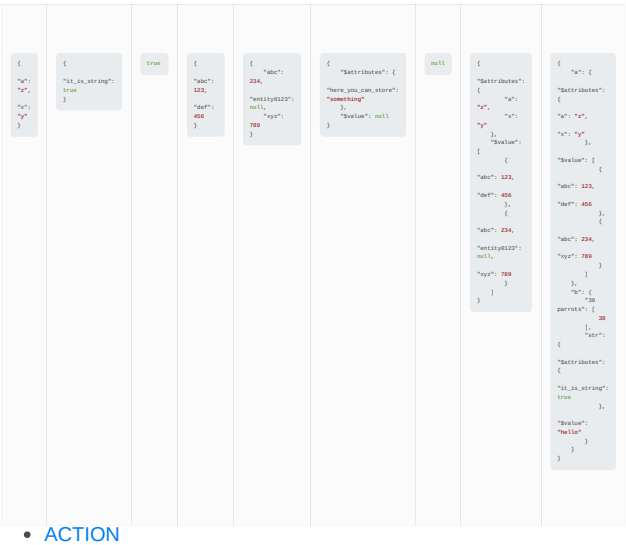
```
"I can just do it"
```

```
{
  "0-25-3ec012f-406daf5c": {
    "a": {
      "$attributes": {
        "why": "I can just do it"
      },
      "$value": 1
    },
    "b": 2
  }
}
```

```
  b = {str = <it_is_string=%true>"hello"; "38 parrots" = [38]};
  entity0 = <here_you_can_store=something>#;
  }
@@);

SELECT Yson::ConvertToStringDict(Yson::YPath($data, "/a/@")) AS attrs_root,
Yson::SerializeJson(Yson::YPath($data, "/b/str/@")) AS attrs_b_str,
Yson::SerializeJson(Yson::YPath($data, "/b/str/@/it_is_string")) AS attr_exact,
Yson::SerializeJson(Yson::YPath($data, "/a/0")) as array_index0,
Yson::SerializeJson(Yson::YPath($data, "/a/-1")) as array_last,
Yson::SerializeJson(Yson::YPath($data, "/entity0")) as entity,
Yson::SerializeJson(Yson::YPath($data, "/a/#entity0123/abc")) as entity1,
Yson::SerializeJson(Yson::YPath($data, "/a")) AS whole_a,
Yson::SerializeJson($data) AS whole_data;
```

Результат:

attrs_root	attrs_b_str	attr_exact	array_index0	array_last	entity	entity1	whole_a	whole_data



- ACTION
- INTO RESULT
- PRAGMA
- DECLARE
- CREATE TOPIC
- ALTER TOPIC
- DROP TOPIC
- CREATE ASYNC REPLICATION
- ALTER ASYNC REPLICATION
- DROP ASYNC REPLICATION

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
```

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 `||`.

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 $_
export $_; --- ошибка: Can not export 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;
```

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. Применимо только совместно с параметром `FAILOVER_MODE` (см. ниже). Возможные значения:
  - `DONE` — завершение процесса асинхронной репликации.
- `FAILOVER_MODE` — режим переключения состояния. Применимо только совместно с параметром `STATE`. Возможные значения:
  - `FORCE` — принудительное переключение состояния.

Примеры

Следующий запрос принудительно завершит процесс асинхронной репликации:

```
ALTER ASYNC REPLICATION my_replication SET (STATE = "DONE", FAILOVER_MODE = "FORCE");
```

См. также

- CREATE ASYNC REPLICATION
- DROP ASYNC REPLICATION

ALTER GROUP

Добавляет или удаляет группу указанному пользователю. Для одного оператора вы можете указать несколько пользователей.

Синтаксис

```
ALTER GROUP role_name ADD USER user_name [, ... ]
ALTER GROUP role_name DROP USER user_name [, ... ]
```

- `role_name` — имя группы.
- `user_name` — имя пользователя.

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.

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: .
- `QUERY_MEMORY_LIMIT_PERCENT_PER_NODE` (Double) — опциональное поле, определяющее процент доступной памяти на узле, который может использовать запрос в данном пуле ресурсов. Если значение `-1`, действует ограничение на общую доступную память между всеми запросами. 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: .
- `QUERY_CPU_LIMIT_PERCENT_PER_NODE` (Double) — опциональное поле, определяющее процент доступного CPU на узле для одного запроса в пуле ресурсов. Если значение `-1`, ограничений нет. Значение по умолчанию: `-1`. Допустимые значения: .
- `RESOURCE_WEIGHT` (Int32) — опциональное поле, задающее веса для распределения ресурсов между пулами. Если значение `-1`, веса не используются. 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. Если поле не указано, классификатор игнорирует `MEMBER_NAME`, и классификация осуществляется по другим признакам.

См. также

- Workload Manager — управление потреблением ресурсов
- CREATE RESOURCE POOL CLASSIFIER
- DROP RESOURCE POOL CLASSIFIER

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`, значение по умолчанию — `18h`.
- `retention_storage_mb` — ограничение на максимальное место на диске, занимаемое данными топика. При превышении этого значения старые данные будут удаляться, как по retention. 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`.

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:

- `important` — определяет важного читателя. Никакие данные из топика не будут удалены, пока все важные читатели их не прочитали. Тип значения — `boolean`, значение по умолчанию: `false`.
- `read_from` — определяет момент времени записи сообщений, начиная с которого читатель будет получать данные. Данные, записанные ранее этого момента, прочитаны не будут. Тип значения: `Datetime` ИЛИ `Timestamp` или `integer` (unix-timestamp в виде числа). Значение по умолчанию — `0` (чтение с самого раннего доступного в топике времени).

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);
```

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 (important = true)
    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 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` форсирует сбор табличной и колоночной статистики для оптимизатора запросов.

Синтаксис:

```
ANALYZE <path_to_table> [ (<column_name> [, ...]) ]
```

В результате выполнения команды будет собрана и актуализирована статистика по указанной таблицам в целом, а также по всем или указанным колонкам. Комадна ANALYZE синхронная, то есть эта команда завершится только тогда, когда будет собрана и актуализирована указанная статистика.

- `path_to_table` — путь к конкретной таблице, для которой требуется форсированный сбор статистики
- `column_name` — форсировать сбор статистики только по определённым колонкам таблицы

Текущий набор статистик указан в разделе Статистики для стоимостного оптимизатора.

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.
- Настройки для аутентификации в базе-источнике одним из способов (обязательно):
  - С помощью токена:
    - `TOKEN_SECRET_NAME` — имя секрета, содержащего токен.
  - С помощью логина и пароля:
    - `USER` — имя пользователя.
    - `PASSWORD_SECRET_NAME` — имя секрета, содержащего пароль.

Примеры

> **Совет**
>
> Перед созданием экземпляра асинхронной репликации 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_NAME = '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_NAME = '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_NAME = '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_NAME = 'my_secret'
);
```

См. также

- ALTER ASYNC REPLICATION
- DROP ASYNC REPLICATION

CREATE GROUP

Создает группу с указанным именем. Есть возможность указать список пользователей, входящих в эту группу.

Синтаксис

```
CREATE GROUP group_name [ WITH USER user_name [ , user_name [ ... ]] [ , ] ]
```

Параметры

- `group_name` — имя группы. Может содержать строчные буквы латинского алфавита и цифры.
- `user_name` — имя пользователя, который станет участником группы после её создания. Может содержать строчные буквы латинского алфавита и цифры.

Примеры

```
CREATE GROUP group1;
```

```
CREATE GROUP group2 WITH USER user1;
```

```
CREATE GROUP group3 WITH USER user1, user2,;
```

```
CREATE GROUP group4 WITH USER user1, user3, user2;
```

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.

CREATE EXTERNAL DATA SOURCE

Вызов `CREATE EXTERNAL DATA SOURCE` создает внешний источник данных.

```
CREATE EXTERNAL DATA SOURCE 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_NAME="password_secret_name"
)
```

Где:

- `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_name` - имя секрета, содержащего 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 именем секрета `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_NAME="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_NAME` имя секрета, содержащего 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 именем секрета `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_NAME="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_NAME` имя секрета, содержащего 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 `192.168.1.2` и портом `5432`, логином `admin` и именем секрета `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_NAME="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 EXTERNAL TABLE

Вызов `CREATE EXTERNAL TABLE` создает внешнюю таблицу с указанной схемой данных.

```
CREATE EXTERNAL TABLE 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:

- `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)

> **Важно**
>
> Текущий синтаксис работы с секретами является временным, в будущих релизах YDB он будет изменен.

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)

Важно

Текущий систаксис работы с секретами является временным, в будущих релизах YDB он будет изменен.

Для управления доступом к секретам используются специальные объекты `SECRET_ACCESS`.

```
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 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.

Правила наименования схемных объектов

У каждого схемного объекта базы данных в YDB есть имя. В YQL-выражениях имена схемных объектов указываются с помощью идентификаторов, заключённых в обратные кавычки (`` ` ``) или без этих символов. Для более подробной информации об идентификаторах, см. Ключевые слова и идентификаторы.

Имена схемных объектов в YDB должны соответствовать следующим требованиям:

- Имя объекта может состоять из следующих символов:
  - прописные латинские буквы;
  - строчные латинские буквы;
  - цифры;
  - специальные символы: `.`, `-` и `_`.
- Длина имени объекта не должна превышать 255 символов.
- Объекты не должны создаваться в папках, имена которых начинаются с точки, таких как `.sys`, `.medatata`, `.sys_health`.

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: .
- `QUERY_MEMORY_LIMIT_PERCENT_PER_NODE` (Double) — опциональное поле, определяющее процент доступной памяти на узле, который может использовать запрос в данном пуле ресурсов. Если значение `-1`, действует ограничение на общую доступную память между всеми запросами. 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: .
- `QUERY_CPU_LIMIT_PERCENT_PER_NODE` (Double) — опциональное поле, определяющее процент доступного CPU на узле для одного запроса в пуле ресурсов. Если значение `-1`, ограничений нет. Значение по умолчанию: `-1`. Допустимые значения: .
- `RESOURCE_WEIGHT` (Int32) — опциональное поле, задающее веса для распределения ресурсов между пулами. Если значение `-1`, веса не используются. 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`;
```

Примеры

```
CREATE RESOURCE POOL olap WITH (
    CONCURRENT_QUERY_LIMIT=20,
    QUEUE_SIZE=1000,
    DATABASE_LOAD_CPU_THRESHOLD=80,
    RESOURCE_WEIGHT=100,
    QUERY_MEMORY_LIMIT_PERCENT_PER_NODE=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.
- Каждый запрос в пуле может потребить не более 80% доступной памяти на узле. Если запрос превысит этот лимит, он будет завершён со статусом `OVERLOADED`.
- Общее ограничение на доступный CPU для всех запросов в пуле на узле составляет 70%.
- Пул ресурсов имеет вес 100, который начинает работать только в случае переподписки.

См. также

- Workload Manager — управление потреблением ресурсов
- ALTER RESOURCE POOL
- DROP RESOURCE POOL

CREATE RESOURCE POOL CLASSIFIER

`CREATE RESOURCE POOL CLASSIFIER` создаёт пул классификаторов 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.

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. Если поле не указано, классификатор игнорирует `MEMBER_NAME`, и классификация осуществляется по другим признакам.

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 такие классификаторы будут пропускаться.

С ограничениями на число классификаторов можно ознакомиться на странице ограничений.

Разрешения

Требуется разрешение `ALL` на базу данных

Пример выдачи такого разрешения:

```
GRANT 'ALL' ON `/my_db` TO `user1@domain`;
```

Примеры

```
CREATE RESOURCE POOL CLASSIFIER olap_classifier WITH (
    RANK=1000,
    RESOURCE_POOL="olap",
    MEMBER_NAME="user1@domain"
)
```

В примере выше создаётся классификатор пула ресурсов с именем `olap_classifier`, который направляет запросы от пользователя `user1@domain` в пул ресурсов с именем `olap`. Запросы от всех остальных пользователей будут отправляться в пул ресурсов `default`, при условии, что других классификаторов пулов ресурсов не существует.

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:

- `important` — определяет важного читателя. Никакие данные из топика не будут удалены, пока все важные читатели их не прочитали. Тип значения — `boolean`, значение по умолчанию: `false`.
- `read_from` — определяет момент времени записи сообщений, начиная с которого читатель будет получать данные. Данные, записанные ранее этого момента, прочитаны не будут. Тип значения: `Datetime` ИЛИ `Timestamp` или `integer` (unix-timestamp в виде числа). Значение по умолчанию — `0` (чтение с самого раннего доступного в топике времени).

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 — `18h`.
- `retention_storage_mb` — ограничение на максимальное место на диске, занимаемое данными топика. При превышении этого значения старые данные будут удаляться, как по retention. 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 — `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`.

Правила наименования схемных объектов

У каждого схемного объекта базы данных в YDB есть имя. В YQL-выражениях имена схемных объектов указываются с помощью идентификаторов, заключённых в обратные кавычки (`` ` ``) или без этих символов. Для более подробной информации об идентификаторах, см. Ключевые слова и идентификаторы.

Имена схемных объектов в YDB должны соответствовать следующим требованиям:

- Имя объекта может состоять из следующих символов:
  - прописные латинские буквы;
  - строчные латинские буквы;
  - цифры;
  - специальные символы: `.`, `-` и `_`.
- Длина имени объекта не должна превышать 255 символов.
- Объекты не должны создаваться в папках, имена которых начинаются с точки, таких как `.sys`, `.medatata`, `.sys_health`.

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')
);
```

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

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Вместо `DELETE FROM` для удаления данных из колоночных таблиц можно воспользоваться механизмом удаления строк по времени — TTL. TTL можно задать при создании таблицы через `CREATE TABLE` или изменить позже через `ALTER TABLE`.

Удаляет строки из строковой 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;
```

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_NAME = '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 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, которого нужно удалить.

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.

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)

Важно

Текущий систаксис работы с секретами является временным, в будущих релизах YDB он будет изменен.

Удаляет указанный секрет.

Если секрета с таким именем не существует, возвращается ошибка.

Пример

```
DROP OBJECT my_secret (TYPE SECRET);
```

DROP OBJECT (TYPE SECRET_ACCESS)

Важно

Текущий систаксис работы с секретами является временным, в будущих релизах YDB он будет изменен.

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 TABLE

Удаляет указанную строковую или колоночную таблицу.

Если таблицы с таким именем не существует, возвращается ошибка.

Примеры

```
DROP TABLE my_table;
```

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.

> **Примечание**
>
> Функциональность находится в разработке, исходный код доступен в ветке *main* репозитория YDB. О том, как собрать YDB из исходного кода, читайте в инструкции.

Общий вид команды:

```
DROP TOPIC topic_path;
```

Следующая команда удалит топик с именем `my_topic`:

```
DROP TOPIC `my_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 сообщения из топиков
`ydb.granular.update_row`	`UPDATE ROW`	Право обновлять строки в таблице (insert, update, upsert, replace), писать сообщения в топики
`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

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Доступные способы загрузки данных в колоночные таблицы:

- YDB CLI
- Пакетная вставка данных
- Yandex Data Transfer

Добавляет строки в строковую 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.

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`;
```

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 /).

UDF

Тип значения	По умолчанию	Статическая / динамическая

Строка	—	Статическая
Строка - имя префикса, добавляемого ко всем модулям	""	Статическая

Импорт всех UDF из указанной приложенной к запросу скомпилированной под Linux x64 разделяемой библиотеки (.so).
При указании префикса, он добавляется перед названием всех загруженных модулей, например, CustomPrefixIp::IsIPv4 вместо Ip::IsIPv4.
Указание префикса позволяет подгрузить одну и ту же UDF разных версий.

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

Тип значения	По умолчанию
Флаг	true

При использовании `SELECT ... 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.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

REPLACE INTO

Важно

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Доступные способы загрузки данных в колоночные таблицы:

- YDB CLI
- Пакетная вставка данных
- Yandex Data Transfer

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 строковую таблицу с перезаписью строк по первичному ключу. Если заданный первичный ключ отсутствует, в таблицу будет добавлена новая строка. Если задан существующий `PRIMARY_KEY`, строка будет перезаписана. При этом значения столбцов, не участвующих в операции, 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 [, ...]} FROM
{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 сообщения из топиков
`ydb.granular.update_row`	`UPDATE ROW`	Право обновлять строки в таблице (insert, update, upsert, replace), писать сообщения в топики
`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;
```

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,
    FAMILY `default` (COMPRESSION = 'zstd'),
    FAMILY `Family1` (COMPRESSION = 'off'),
    FAMILY `Family2` (COMPRESSION = 'lz4'),
    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
;
```

UPDATE

Важно

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Доступные способы загрузки данных в колоночные таблицы:

- YDB CLI
- Пакетная вставка данных
- Yandex Data Transfer

`UPDATE` не может менять значение колонок, входящих в состав первичного ключа.

Изменяет данные в строковой таблице. После ключевого слова `SET` указываются столбцы, значение которых необходимо заменить, и сами новые значения. Список строк задается с помощью условия `WHERE`. Если `WHERE` отсутствует, изменения будут применены ко всем строкам таблицы.

`UPDATE` не может менять значение `PRIMARY_KEY`.

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;
```

UPSERT INTO

Важно

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Доступные способы загрузки данных в колоночные таблицы:

- YDB CLI
- Пакетная вставка данных
- Yandex Data Transfer

UPSERT (расшифровывается как UPDATE or INSERT) обновляет или добавляет множество строк в строковой 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 OBJECT (TYPE SECRET)

> **Важно**
>
> Текущий систаксис работы с секретами является временным, в будущих релизах YDB он будет изменен.

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`.

Ещё не поддерживаемые конструкции из классического SQL

[NOT] [EXISTS|INTERSECT|EXCEPT]

Доступный альтернативный вариант — `EXISTS` синтаксически доступен, но из-за отсутствия поддержки коррелированных подзапросов не очень полезен. Также можно переписать через `JOIN`.

NATURAL JOIN

Доступный альтернативный вариант — явно перечислить совпадающие с обеих сторон колонки.

NOW() / CURRENT_TIME()

Доступный альтернативный вариант — воспользоваться функциями CurrentUtcDate, CurrentUtcDatetime и CurrentUtcTimestamp.

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.

Добавление, удаление и переименование индекса

Внимание

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Внимание

Функционал векторных индексов доступен в тестовом режиме в main. В стабильной версии планируется в 25.1.

В настоящее время не поддерживается:

- изменение строк в таблицах с векторными индексами;
- построение индекса для векторов с битовым квантованием.

Эти ограничения могут быть устранены в будущих версиях.

Добавление индекса

`ADD INDEX` — добавляет индекс с указанным именем и типом для заданного набора колонок в строковых таблицах. Приведенный ниже код добавит глобальный индекс с именем `title_index` для колонки `title`.

```
ALTER TABLE `series` ADD INDEX `title_index` GLOBAL ON (`title`);
```

Могут быть указаны все параметры вторичного индекса, описанные в команде `CREATE TABLE`.
Могут быть указаны все параметры векторного индекса, описанные в команде `CREATE TABLE`.

Также добавить вторичный индекс можно с помощью команды table index YDB CLI.

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, который должен быть одним из следующих:
  - 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

Примечание

Эти настройки нельзя вернуть к исходным.

- `<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`

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"
);
```

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.

Пример переименования индекса:

```
ALTER TABLE `series` RENAME INDEX `title_index` TO `title_index_new`;
```

Изменение состава колонок

YDB поддерживает возможность добавлять столбцы в строковые и колоночные таблицы, а также удалять неключевые колонки из таблиц.

`ADD COLUMN` — добавляет столбец с указанными именем и типом. Приведенный ниже код добавит к таблице `episodes` столбец `views` с типом данных `Uint64`.

```
ALTER TABLE episodes ADD COLUMN views Uint64;
```

`DROP COLUMN` — удаляет столбец с указанным именем. Приведенный ниже код удалит столбец `views` из таблицы `episodes`.

```
ALTER TABLE episodes DROP COLUMN views;
```

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

> **Внимание**
>
> 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.
- `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 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

Внимание

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

```
ALTER TABLE old_table_name RENAME TO new_table_name;
```

См. правила наименования таблиц и колонок

Правила наименования схемных объектов

У каждого схемного объекта базы данных в YDB есть имя. В YQL-выражениях имена схемных объектов указываются с помощью идентификаторов, заключённых в обратные кавычки (`` ` ``) или без этих символов. Для более подробной информации об идентификаторах, см. Ключевые слова и идентификаторы.

Имена схемных объектов в YDB должны соответствовать следующим требованиям:

- Имя объекта может состоять из следующих символов:
  - прописные латинские буквы;
  - строчные латинские буквы;
  - цифры;
  - специальные символы: `.`, `-` и `_`.
- Длина имени объекта не должна превышать 255 символов.
- Объекты не должны создаваться в папках, имена которых начинаются с точки, таких как `.sys`, `.medatata`, `.sys_health`.

Правила наименования колонок

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_`.

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`;
```

Создание и изменение групп колонок

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

Внимание

Поддерживается только для строковых таблиц.

Приведённый ниже код для группы колонок `default` в таблице `series_with_families` сменит тип хранилища на `hdd`:

```
ALTER TABLE series_with_families ALTER FAMILY default SET DATA "hdd";
```

Примечание

Доступные типы устройств хранения зависят от конфигурации кластера YDB.

Изменение кодека сжатия

Внимание

Кодек `"zstd"` поддерживается только для колоночных таблиц.

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 уровня кодека сжатия

Внимание

Поддерживается только для колоночных таблиц.

Приведённый ниже код для группы колонок `default` в таблице `series_with_families` сменит уровень кодека сжатия, если он поддерживает различные уровни сжатия:

```
ALTER TABLE series_with_families ALTER FAMILY default SET COMPRESSION_LEVEL 5;
```

Могут быть указаны все параметры группы колонок, описанные в команде `CREATE TABLE`

CREATE TABLE

Синтаксис CREATE TABLE

Вызов `CREATE TABLE` создает таблицу с указанной схемой данных и ключевыми колонками (`PRIMARY KEY`). Позволяет определить вторичные индексы на создаваемой таблице.

```
CREATE [TEMP | TEMPORARY] TABLE table_name (
    column1 type1,
    column2 type2 NOT NULL,
    ...
    columnN typeN,
    INDEX index1_name GLOBAL ON ( column ),
    INDEX index2_name GLOBAL ON ( column1, column2, ... ),
    PRIMARY KEY ( column, ... ),
    FAMILY column_family ( family_options, ... )
)
WITH ( key = value, ... )
```

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.

Правила наименования схемных объектов

У каждого схемного объекта базы данных в YDB есть имя. В YQL-выражениях имена схемных объектов указываются с помощью идентификаторов, заключённых в обратные кавычки (`` ` ``) или без этих символов. Для более подробной информации об идентификаторах, см. Ключевые слова и идентификаторы.

Имена схемных объектов в YDB должны соответствовать следующим требованиям:

- Имя объекта может состоять из следующих символов:
  - прописные латинские буквы;
  - строчные латинские буквы;
  - цифры;
  - специальные символы: `.` , `-` и `_` .
- Длина имени объекта не должна превышать 255 символов.
- Объекты не должны создаваться в папках, имена которых начинаются с точки, таких как `.sys` , `.medatata` , `.sys_health` .

Правила наименования колонок

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_` .

Примеры создания таблиц

Создание строковой таблицы

```
  CREATE TABLE <table_name> (
    a Uint64,
```

```
    b Uint64,
    c Float,
    PRIMARY KEY (a, b)
  );
```

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
);
```

При создании колоночных таблиц обязательно нужно использовать конструкцию `PARTITION BY HASH` с указанием первичных ключей, которые имеют высококардинальный тип данных (например, `Timestamp`), так как колоночные таблицы партиционируют данные не по первичным ключам, а по специально выделенным ключам — ключам партицирования. Подробно про ключи партиционирования колоночных таблиц изложено 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.
- Группы колонок.
- Дополнительные параметры.

Для колоночных таблиц при их создании возможно задать:

- Группы колонок.
- Дополнительные параметры.

INDEX

> **Внимание**
>
> Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Конструкция `INDEX` используется для определения вторичного индекса для строковых таблиц:

```
CREATE TABLE table_name (
    ...
    INDEX <index_name> GLOBAL [UNIQUE] [SYNC|ASYNC] [USING <тип_индекса>] ON ( <index_columns> ) COVER ( <cover_columns>
),
    ...
)
```

где:

- **index_name** — уникальное имя индекса, по которому будет возможно обращение к данным.
- **SYNC/ASYNC** — синхронная или асинхронная запись в индекс, если не указано — синхронная.
- **тип_индекса** - тип индекса, в настоящее время поддерживается только `secondary`.
- **UNIQUE** — создаёт индекс с гарантией уникальности для вставляемых значений.
- **index_columns** — имена колонок создаваемой таблицы через запятую, по которым возможен поиск в индексе.
- **cover_columns** — имена колонок создаваемой таблицы через запятую, которые будет сохранены в индексе дополнительно к колонкам поиска, давая возможность получить дополнительные данные без обращения за ними в таблицу.

Примеры создания таблиц со вторичным индексом

Строковая таблица со вторичным индексом

```
CREATE TABLE my_table (
    a Uint64,
    b Uint64,
    c Utf8,
    d Date,
    INDEX idx_d GLOBAL ON (d),
    INDEX idx_ba GLOBAL ASYNC ON (b, a) COVER (c),
    INDEX idx_bc GLOBAL UNIQUE SYNC ON (b, c),
    PRIMARY KEY (a)
)
```

Колоночная таблица со вторичным индексом

```
CREATE TABLE my_table (
    a Uint64 NOT NULL,
    b Uint64,
    c Utf8,
    d Date,
    INDEX idx_d GLOBAL ON (d),
    INDEX idx_ba GLOBAL ASYNC ON (b, a) COVER (c),
    PRIMARY KEY (a)
)
WITH (
    STORE = COLUMN
);
```

Векторный индекс

Внимание

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Внимание

Функционал векторных индексов доступен в тестовом режиме в main. В стабильной версии планируется в 25.1.

В настоящее время не поддерживается:

- изменение строк в таблицах с векторными индексами;
- построение индекса для векторов с битовым квантованием.

Эти ограничения могут быть устранены в будущих версиях.

Важно

Создание пустой таблицы с векторным индексом в настоящее время не имеет практического смысла, так как модификация данных в таблицах с векторными индексами пока не поддерживается.

Следует использовать команду `ALTER TABLE ... ADD INDEX` для добавления векторного индекса в существующую таблицу.

Конструкция `INDEX` используется для определения векторного индекса в строчно-ориентированных таблицах:

```
CREATE TABLE table_name (
    ...
    INDEX <имя_индекса> GLOBAL [SYNC] USING <тип_индекса> ON ( <колонки_индекса> ) COVER ( <покрывающие_колонки> ) WITH
( <параметры_индекса> ),
    ...
)
```

Где:

- **имя_индекса** - уникальное имя индекса для доступа к данным
- **SYNC** - указывает на синхронную запись данных в индекс. Если не указано - синхронная.
- **тип_индекса** - тип индекса, в настоящее время поддерживается только `vector_kmeans_tree`
- **колонки_индекса** - список колонок таблицы через запятую, используемых для поиска по индексу (последняя колонка используется как эмбеддинг, остальные - как префиксные колонки)
- **покрывающие_колонки** - дополнительные колонки таблицы, сохраняемые в индексе для возможности их извлечения без обращения к основной таблице
- **параметры_индекса** - список параметров в формате ключ-значение:
  - общие параметры для всех векторных индексов:
    - `vector_dimension` - размерность вектора эмбеддинга (<= 16384);
    - `vector_type` - тип значений вектора (`float` , `uint8` , `int8` , `bit`);
    - `distance` - функция расстояния (`cosine` , `manhattan` , `euclidean`) или `similarity` - функция схожести (`inner_product` , `cosine`).
  - специфичные параметры для `vector_kmeans_tree` :
    - `clusters` - количество центроидов для алгоритма k-means (значения > 1000 могут ухудшить производительность);
    - `levels` - количество уровней в дереве.

Важно

Параметры `distance` и `similarity` не могут быть указаны одновременно.

Важно

Векторные индексы с `vector_type=bit` в настоящее время не поддерживаются

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),
    PRIMARY KEY (article_id)
)
```

Группы колонок

Колонки одной таблицы можно объединять в группы, чтобы задать следующие параметры:

- `DATA` — тип устройства хранения для данных колонок этой группы. Допустимые значения: `"ssd"`, `"rot"`.

> **Внимание**
>
> Поддерживается только для строковых таблиц.

- `COMPRESSION` — кодек сжатия данных. Допустимые значения: `"off"`, `"lz4"`, `"zstd"`.

> **Внимание**
>
> Кодек `"zstd"` поддерживается только для колоночных таблиц.

- `COMPRESSION_LEVEL` — уровень сжатия кодека, если кодек поддерживает уровень сжатия.

> **Внимание**
>
> Поддерживается только для колоночных таблиц.

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.

В примерах ниже для создаваемых таблиц 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.

Создание строковой таблицы

```
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"
    ),
    FAMILY family_large (
        DATA = "rot",
        COMPRESSION = "lz4"
    )
);
```

Создание колоночной таблицы

```
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 (
        COMPRESSION = "lz4"
    ),
    FAMILY family_large (
        COMPRESSION = "zstd",
        COMPRESSION_LEVEL = 5
    )
)
WITH (STORE = COLUMN);
```

Примечание

Доступные типы устройств хранения зависят от конфигурации кластера YDB.

Создание временных таблиц (TEMPORARY)

Внимание

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

`TEMPORARY` / `TEMP` – временная таблица, которая автоматически удаляется при завершении сессии. Если параметр не задан (оставлен пустым) – создается постоянная таблица. Любые индексы, созданные во временных таблицах, также будут удалены при завершении сессиии, следовательно они тоже являются временными. Допускается существование временной таблицы и постоянной таблицы с одинаковым именем, в этом случае будет выбрана временная таблица.

```
CREATE TEMPORARY TABLE table_name (
    ...
);
```

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:

> **Внимание**
>
> 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
);
```

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;
- к входным таблицам применяется MATCH_RECOGNIZE
- вычисляется SAMPLE / TABLESAMPLE
- 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

```
USE some_cluster;
SELECT * FROM CONCAT(
  `table1`,
  `table2`,
  `table3`);
```

```
USE some_cluster;
$indices = ListFromRange(1, 4);
$tables = ListMap($indices, ($index) -> {
```

```
    RETURN "table" || CAST($index AS String);
});
SELECT * FROM EACH($tables); -- идентично предыдущему примеру
```

```
USE some_cluster;
SELECT * FROM RANGE(`my_folder`);
```

```
SELECT * FROM some_cluster.RANGE( -- Кластер можно указать перед названием функции
  `my_folder`,
  `from_table`,
  `to_table`);
```

```
USE some_cluster;
SELECT * FROM RANGE(
  `my_folder`,
  `from_folder`,
  `to_folder`,
  `my_table`);
```

```
USE some_cluster;
SELECT * FROM RANGE(
  `my_folder`,
  `from_table`,
  `to_table`,
  ``,
  `my_view`);
```

```
USE some_cluster;
SELECT * FROM LIKE(
  `my_folder`,
  "2017-03-%"
);
```

```
USE some_cluster;
SELECT * FROM REGEXP(
  `my_folder`,
  "2017-03-1[2-4]?"
);
```

```
$callable = ($table_name) -> {
    return $table_name > "2017-03-13";
};

USE some_cluster;
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
- SAMPLE
- TABLESAMPLE

- MATCH_RECOGNIZE
- JOIN
- GROUP BY
- FLATTEN
- WINDOW
- VIEW secondary_index
- VIEW vector_index

FROM

Источник данных для `SELECT`. В качестве аргумента может принимать имя строковой или колоночной 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);
```

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 *;
```

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

Сгруппировать таблицу по значениям указанных столбцов или выражений, а также подмножества по времени (окно времени).

Среди столбцов, по которым производится группировка, должна присутствовать конструкция `HOP`, определяющая характеристики окна времени для группировки.

```
HOP(time_extractor, hop, interval, delay)
```

Реализованный вариант окна времени называется **hopping window**. Это окно, продвигающееся вперёд дискретными интервалами (параметр `hop`). Общая длительность окна задаётся параметром `interval`. Для определения времени каждого входного события используется параметр `time_extractor`. Это выражение, зависящее только от входных значений столбцов, должно иметь тип `Timestamp`. Оно указывает, откуда именно в данных доставать значение времени.

При этом происходит следующее:

1. Входная таблица партиционируется по ключам группировки, указанным в `GROUP BY`, без учета HOP. Если кроме HOP в `GROUP BY` ничего нет, то входная таблица попадает в одну партицию.

2. Каждая партиция сортируется по возрастанию значения выражения `time_extractor`.
3. Каждая партиция делится на подмножества (возможно пересекающиеся), на которых вычисляются агрегатные функции.

В каждом потоке (партиции), определяемом значениями всех столбцов группировки, окно продвигается независимо от других потоков. Продвижение окна полностью зависит от самого позднего события в партиции.

Параметры `interval` и `delay` следует задавать кратными параметру `hop`. Некратные интервалы в текущей реализации запрещены. Параметры `interval` и `hop` следует задавать положительными.

Параметр `delay` в текущей реализации игнорируется т.к. данные в одной партиции уже отсортированы.

Для задания `hop`, `interval` и `delay` используется строковое выражение, соответствующее стандарту ISO 8601. Это формат, который используется для конструирования встроенного типа `Interval` из строки.

При выборке столбцов (между `SELECT ... FROM`) можно использовать функции `HOP_START` и `HOP_END` (без параметров), которые возвращают значение типа `Timestamp` и соответствуют началу и концу текущего окна.

Известное в других системах **tumbling window** является частным случаем **hopping window**, когда `interval` == `hop`.

Примеры

```
SELECT
    key,
    COUNT(*)
FROM my_table
GROUP BY
    HOP(CAST(subkey AS Timestamp), "PT10S", "PT1M", "PT30S"),
    key;
-- hop = 10 секунд
-- interval = 1 минута
-- delay = 30 секунд
```

```
SELECT
    double_key,
    HOP_END() as time,
    COUNT(*) as count
FROM my_table
GROUP BY
    key + key AS double_key,
    HOP(ts, "PT1M", "PT1M", "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  ...
```

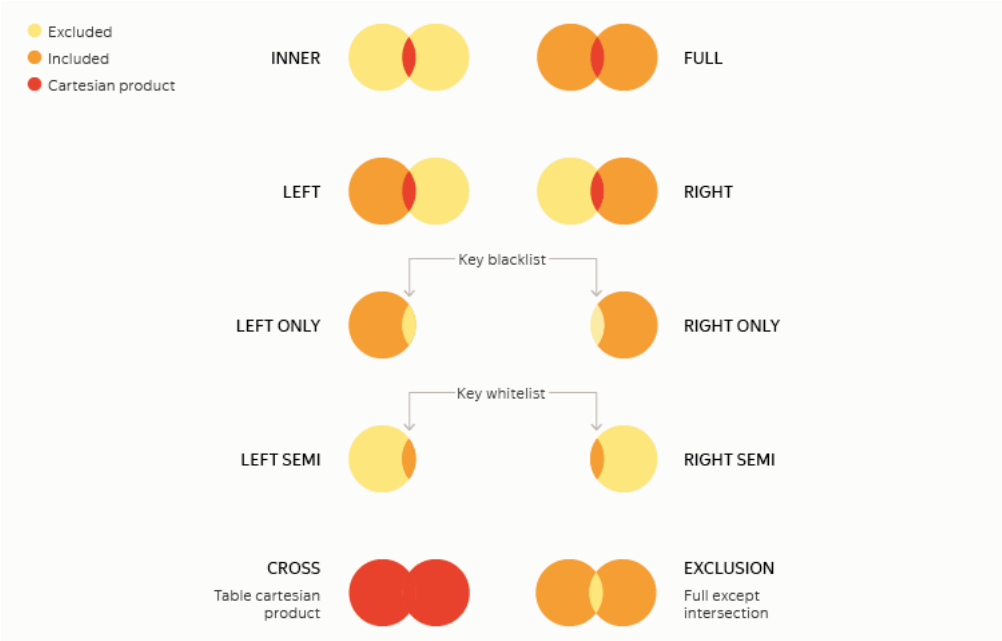
На каждом шаге объединения по заданным правилам определяются соответствия между строками исходной и присоединяемой выборок данных, и формируется новая выборка, в которую попадают все сочетания подошедших под условия объединения строк.

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.



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)

Внимание

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

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 всего UNION
ALL
```

VIEW (INDEX)

> **Внимание**
>
> Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Чтобы сделать запрос `SELECT` по вторичному индексу строковой таблицы, используйте конструкцию:

```
SELECT *
    FROM TableName VIEW IndexName
    WHERE …
```

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)

Важно

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Внимание

Функционал векторных индексов доступен в тестовом режиме в main. В стабильной версии планируется в 25.1.

В настоящее время не поддерживается:

- изменение строк в таблицах с векторными индексами;
- построение индекса для векторов с битовым квантованием.

Эти ограничения могут быть устранены в будущих версиях.

Для выполнения запроса `SELECT` с использованием векторного индекса в строчно-ориентированной таблице используйте следующий синтаксис:

```
SELECT ...
    FROM TableName VIEW IndexName
    WHERE ...
    ORDER BY Knn::DistanceFunction(...)
    LIMIT ...
```

```
SELECT ...
    FROM TableName VIEW IndexName
    WHERE ...
    ORDER BY Knn::SimilarityFunction(...) DESC
    LIMIT ...
```

Примечание

Векторный индекс не будет автоматически выбран оптимизатором, поэтому его нужно указывать явно с помощью выражения `VIEW IndexName`.

Примеры

- Выбор всех полей из таблицы `series` с использованием векторного индекса `views_index`, созданного для `embedding` с мерой близости "скалярное произведение":

  ```
  SELECT series_id, title, info, release_date, views, uploaded_user_id, Knn::InnerProductSimilarity(embedding,
  $target) as similarity
      FROM series VIEW views_index
      ORDER BY similarity DESC
      LIMIT 10
  ```

- Выбор всех полей из таблицы `series` с использованием префиксного векторного индекса `views_index2`, созданного для `embedding` с мерой близости "скалярное произведение" и префиксной колонкой `release_date`:

  ```
  SELECT series_id, title, info, release_date, views, uploaded_user_id, Knn::InnerProductSimilarity(embedding,
  $target) as similarity
      FROM series VIEW views_index2
      WHERE release_date = "2025-03-31"
      ORDER BY Knn::InnerProductSimilarity(embedding, $TargetEmbedding) DESC
      LIMIT 10
  ```

WITH

Задается после источника данных во `FROM` и используется для указания дополнительных подсказок использования строковых и колоночных 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 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>>;
```

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;
```

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
```

TABLESAMPLE и SAMPLE

Внимание

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Построение случайной выборки из указанного во `FROM` источника данных.

`TABLESAMPLE` является частью SQL стандарта и работает следующим образом:

- Указывается режим работы:
  - `BERNOULLI` означает «медленно, честно просмотрев все данные, но по-настоящему случайно»;
  - `SYSTEM` должен использовать знание о физическом хранении данных, чтобы избежать полного их чтения частично жертвуя случайностью выборки.

Данные разбиваются на достаточно большие блоки, и происходит сэмплирование данных блоков целиком. Для прикладных расчётов на достаточно больших таблицах результат вполне может оказаться состоятельным.

- Размер случайной выборки указывается в процентах следом за режимом в круглых скобках.
- Управлять размером блоков для режима `SYSTEM` можно с помощью прагмы `yt.SamplingIoBlockSize`. Если при семплировании размер результата получается меньше одного блока, то выдается пустой результат.
- Опционально далее указывается ключевое слово `REPEATABLE` и целое число в скобках, которое будет использовано как seed для генератора псевдослучайных чисел.

`SAMPLE` является более коротким алиасом, где нет сложных настроек, а размер выборки указывается в долях. На данный момент он соответствует режиму `BERNOULLI`.

Примечание

В режиме `BERNOULLI` при наличии ключевого слова `REPEATABLE` в seed подмешивается chunk id для каждого чанка таблицы. Поэтому выборка для разных таблиц с одинаковым содержимым может давать разные результаты.

Примеры

```
SELECT *
FROM my_table
TABLESAMPLE BERNOULLI(1.0) REPEATABLE(123); -- один процент таблицы
```

```
SELECT *
FROM my_table
TABLESAMPLE SYSTEM(1.0); -- примерно один процент таблицы
```

```
SELECT *
FROM my_table
SAMPLE 1.0 / 3; -- треть таблицы
```

MATCH_RECOGNIZE

Выражение `MATCH_RECOGNIZE` выполняет распознавание паттерна в последовательности строк и возвращает найденные результаты. Данная функциональность важна для различных сфер деятельности, таких как выявление мошенничества, анализ ценообразования в сфере финансов и обработка данных датчиков. Эта область известна как Complex Event Processing (CEP), и распознавание паттерна является важным инструментом для этого. Пример работы `MATCH_RECOGNIZE` приведен по ссылке.

Алгоритм обработки данных

Выражение `MATCH_RECOGNIZE` выполняет следующие действия:

1. Входная таблица разбивается на непересекающиеся группы. Каждая группа состоит из набора строк входной таблицы с одинаковыми значениями в колонках, перечисленных после `PARTITION BY`.
2. Каждая группа упорядочивается в соответствии с блоком `ORDER BY`.
3. В каждой упорядоченной группе независимо выполняется распознавание паттерна из `PATTERN`.
4. Поиск паттерна в последовательности строк проходит пошагово: строки одна за другой проверяются на соответствие паттерну. Среди всех совпадений, начинающихся в первой строке, выбирается то, которое состоит из наибольшего количества строк. Если не было найдено совпадений с началом в первой строке, то поиск продолжается, начиная со следующей строки.
5. После нахождения совпадения вычисляются колонки, задающиеся выражениями в блоке `MEASURES`.
6. В зависимости от режима `ROWS PER MATCH` выводится одна или все строки для найденного совпадения.
7. Режим `AFTER MATCH SKIP` определяет, с какой строки возобновится распознавание паттерна.

Синтаксис

```
MATCH_RECOGNIZE (
    [ PARTITION BY <партиция_1> [ ... , <партиция_N> ] ]
    [ ORDER BY <ключ_сортировки_1> [ ... , <ключ_сортировки_N> ] ]
    [ MEASURES <выражение_1> AS <имя_колонки_1> [ ... , <выражение_N> AS <имя_колонки_N> ] ]
    [ ROWS PER MATCH ]
    [ AFTER MATCH SKIP ]
    PATTERN (<паттерн_для_поиска>)
    DEFINE <переменная_1> AS <предикат_1> [ ... , <переменная_N> AS <предикат_N> ]
)
```

Описание элементов SQL-синтаксиса выражения `MATCH_RECOGNIZE`:

- `DEFINE` – блок объявления переменных, с помощью которых описывается паттерн для поиска, и условий, которым должны соответствовать строки для каждой из переменных.
- `PATTERN` – регулярное выражение, описывающее искомый паттерн для поиска.
- `MEASURES` определяет набор колонок для возвращаемых данных. Каждая колонка задаётся SQL-выражением для ее вычисления.
- `ROWS PER MATCH` определяет структуру возвращаемых данных и количество строк по каждому найденному совпадению.
- `AFTER MATCH SKIP` определяет способ перехода к месту поиска следующего совпадения.
- `ORDER BY` определяет сортировку входных данных. Поиск паттернов выполняется внутри данных, упорядоченных в соответствии со списком колонок или выражениями, перечисленными в `<ключ_сортировки_1> [ ... , <ключ_сортировки_N> ]`.
- `PARTITION BY` разделяет входную таблицу по заданным правилам в соответствии с `<партиция_1> [ ... , <партиция_N> ]`. В каждой из частей поиск паттернов производится независимо.

DEFINE

```
DEFINE <переменная_1> AS <предикат_1> [ ... , <переменная_N> AS <предикат_N> ]
```

С помощью `DEFINE` объявляются переменные, на основе которых описывается искомый паттерн, задаваемый в `PATTERN`. Переменные – это именованные SQL-выражения, вычисляемые над входными данными. Синтаксис SQL-выражений в `DEFINE` совпадает с SQL-выражениями предиката `WHERE`. Например, выражение `button = 1` выполняет поиск строк со значением `1` в колонке `button`. В качестве условий могут выступать любые SQL-выражения, с помощью которых можно выполнять поиск, включая функции агрегации (`LAST`, `FIRST`). Например, `button > 2 AND zone_id < 12` или `LAST(button) > 10`.

В примере ниже SQL-выражение `A.button = 1` объявлено как переменная `A`.

```
DEFINE
    A AS A.button = 1
```

> **Примечание**
>
> В настоящий момент в `DEFINE` не поддерживаются функции агрегации, такие как `AVG`, `MIN`, `MAX`, и функции `PREV` и `NEXT`.

При обработке очередной строки данных производится вычисление всех SQL-выражений, описывающих переменные в `DEFINE`. Когда SQL-выражение, описывающее соответствующую переменную из `DEFINE`, принимает значение `ИСТИНА` (`TRUE`), то такая строка получает метку с названием переменной `DEFINE` и добавляется к списку рассматриваемых на совпадение с паттернами.

Пример

При описании переменных в SQL-выражениях можно использовать ссылки на другие переменные:

```
DEFINE
    A AS A.button = 1 AND LAST(A.zone_id) = 12,
    B AS B.button = 2 AND FIRST(A.zone_id) = 12
```

Строка входных данных будет вычислена как переменная `A`, если в ней присутствует колонка `button` со значением `1`, а в последней строке из множества ранее совпавших с переменной `A` есть колонка `zone_id` со значением `12`. Строка будет вычислена как переменная `B`, если в строке данных присутствует колонка `button` со значением `2`, а в первой строке из множества ранее совпавших с переменной `A` есть колонка `zone_id` со значением `12`.

PATTERN

```
PATTERN (<паттерн_для_поиска>)
```

Ключевое слово `PATTERN` описывает искомый паттерн для поиска в виде, вычисляющийся на основе переменных из блока `DEFINE`. Синтаксис `PATTERN` схож с синтаксисом регулярных выражений.

> **Важно**
>
> Если переменная, использованная в блоке `PATTERN`, не была предварительно описана в блоке `DEFINE`, то считается, что она всегда принимает значение `TRUE`.

В `PATTERN` можно использовать квантификаторы. Они в регулярных выражениях определяют число повторений элемента или подпоследовательности в паттерне для нахождения совпадения. Приведем перечень поддерживаемых квантификаторов:

Квантификатор	Описание
`A+`	Одно или несколько повторений переменной `A`
`A*`	Ноль или несколько повторений переменной `A`
`A?`	Ноль или одно повторение переменной `A`
`B{n}`	Ровно `n` повторений переменной `B`
`C{n, m}`	От `n` до `m` повторений переменной `C`
`D{n,}`	Не менее `n` повторений переменной `D`
`(A|B)`	Появление переменной `A` или `B` в данных
`(A|B){,m}`	От нуля до `m` повторений переменной `A` или `B`

Поддерживаемые последовательности поиска паттернов:

Поддерживаемые последовательности	Синтаксис	Описание
Последовательность	`A B+ C+ D+`	Производится поиск точной указанной последовательности, не допускается появление других переменных внутри последовательности. Поиск паттерна производится в порядке указания переменных паттерна.
Один из	`A | B | C`	Переменные перечисляются в любом порядке с указанием символа | между ними. Производится поиск любой переменной из указанного списка.
Группировка	`(A | B)+ | C`	Переменные внутри круглых скобок считаются единой группой. В этом случае квантификаторы применяются ко всей группе сразу.
Исключение из результата	`{- A B+ C -}`	При выборе режима ALL ROWS PER MATCH строки, найденные по паттерну в скобках, будут исключены из результата

Пример

```
PATTERN (B1 E* B2+ B3)
DEFINE
    B1 as B1.button = 1,
    B2 as B2.button = 2,
    B3 as B3.button = 3
```

В блоке `DEFINE` описаны переменные `B1`, `B2`, `B3`. Переменная `E` в блоке `DEFINE` не описана. Такая запись позволяет интерпретировать `E` как любое событие, поэтому будет искаться следующий паттерн: одно нажатие `кнопки 1`, одно или более нажатий `кнопки 2` и одно нажатие `кнопки 3`. При этом между нажатием `кнопки 1` и `кнопки 2` может быть произвольное число любых других событий.

MEASURES

```
MEASURES <выражение_1> AS <имя_колонки_1> [ ... , <выражение_N> AS <имя_колонки_N> ]
```

`MEASURES` описывает набор возвращаемых колонок при нахождении паттерна. Набор возвращаемых колонок должен быть представлен SQL-выражением с агрегирующими функциями над переменными, объявленными в конструкции `DEFINE`.

Пример

Входными данными для примера являются:

ts	button	device_id	zone_id
100	1	3	0
200	1	3	1
300	2	2	0
400	3	1	1

```
MEASURES
    AGGREGATE_LIST(B1.zone_id * 10 + B1.device_id) AS ids,
    COUNT(DISTINCT B1.zone_id) AS count_zones,
    LAST(B3.ts) - FIRST(B1.ts) AS time_diff,
    42 AS meaning_of_life
PATTERN (B1+ B2 B3)
DEFINE
    B1 AS B1.button = 1,
    B2 AS B2.button = 2,
    B3 AS B3.button = 3
```

Результат:

ids	count_zones	time_diff	meaning_of_life
[3,13]	2	300	42

Колонка `ids` содержит список значений `zone_id * 10 + device_id`, посчитанных среди совпавших с переменной `B1` строк. Колонка `count_zones` содержит количество уникальных значений колонки `zone_id` среди совпавших с переменной `B1` строк. Колонка `time_diff` содержит разницу между значением колонки `ts` в последней строке из множества совпавших с переменной `B3` и значением колонки `ts` в первой строке из множества совпавших с переменной `B1`. Колонка `meaning_of_life` содержит константу `42`. Таким образом, выражение в `MEASURES` может содержать агрегирующие функции над несколькими переменными, но внутри одной агрегирующей функции должна быть только одна переменная.

ROWS PER MATCH

`ROWS PER MATCH` определяет количество строк результата для каждого найденного совпадения, а также множество возвращаемых колонок. Режим по умолчанию - `ONE ROW PER MATCH`.

`ONE ROW PER MATCH` устанавливает режим работы `ROWS PER MATCH` на вывод одной строки для найденного совпадения. Структура возвращаемых данных соответствует колонкам, перечисленным в `PARTITION BY` и `MEASURES`.

`ALL ROWS PER MATCH` устанавливает режим работы `ROWS PER MATCH` на вывод всех строк для найденного совпадения, кроме явно исключенных скобками. Помимо колонок исходной таблицы структура возвращаемых данных включает в себя колонки, перечисленные в `MEASURES`.

Примеры

Входными данными для всех примеров являются:

ts	button
100	1
200	2
300	3

Пример 1

```
MEASURES
    FIRST(B1.ts) AS first_ts,
    FIRST(B2.ts) AS mid_ts,
    LAST(B3.ts) AS last_ts
ONE ROW PER MATCH
PATTERN (B1 {- B2 -} B3)
DEFINE
    B1 AS B1.button = 1,
    B2 AS B2.button = 2,
    B3 AS B3.button = 3
```

Результат:

first_ts	mid_ts	last_ts
100	200	300

Пример 2

```
MEASURES
    FIRST(B1.ts) AS first_ts,
    FIRST(B2.ts) AS mid_ts,
    LAST(B3.ts) AS last_ts
ALL ROWS PER MATCH
PATTERN (B1 {- B2 -} B3)
DEFINE
    B1 AS B1.button = 1,
    B2 AS B2.button = 2,
    B3 AS B3.button = 3
```

Результат:

first_ts	mid_ts	last_ts	button	ts
100	200	300	1	100
100	200	300	3	300

AFTER MATCH SKIP

`AFTER MATCH SKIP` определяет способ перехода от найденного совпадения к поиску следующего. В режиме `AFTER MATCH SKIP TO NEXT ROW` поиск следующего совпадения начинается после первой строки предыдущего, в режиме `AFTER MATCH SKIP PAST LAST ROW` - после последней строки предыдущего совпадения. Режим по умолчанию - `PAST LAST ROW`.

Примеры

Входными данными для всех примеров являются:

ts	button
100	1
200	1
300	2
400	3

Пример 1

```
MEASURES
    FIRST(B1.ts) AS first_ts,
    LAST(B3.ts) AS last_ts
AFTER MATCH SKIP TO NEXT ROW
PATTERN (B1+ B2 B3)
DEFINE
    B1 AS B1.button = 1,
    B2 AS B2.button = 2,
    B3 AS B3.button = 3
```

Результат:

first_ts	last_ts
100	400
200	400

Пример 2

```
MEASURES
    FIRST(B1.ts) AS first_ts,
    LAST(B3.ts) AS last_ts
AFTER MATCH SKIP PAST LAST ROW
PATTERN (B1+ B2 B3)
DEFINE
    B1 AS B1.button = 1,
    B2 AS B2.button = 2,
    B3 AS B3.button = 3
```

Результат:

first_ts	last_ts
100	400

ORDER BY

```
ORDER BY <ключ_сортировки_1> [ ... , <ключ_сортировки_N> ]

<ключ_сортировки> ::= { <имена_колонок> | <выражение> }
```

`ORDER BY` определяет сортировку входных данных. То есть перед выполнением всех операций поиска паттернов, данные будут предварительно отсортированы по указанным ключам или выражениям. Синтаксис аналогичен SQL-выражению `ORDER BY`.

Пример

```
ORDER BY CAST(ts AS Timestamp)
```

PARTITION BY

```
PARTITION BY <партиция_1> [ ... , <партиция_N> ]
```

```
<партиция> ::= { <имена_колонок> | <выражение> }
```

`PARTITION BY` - выражение разбивает исходные данные на несколько непересекающихся групп, в каждой из которых независимо производится поиск паттернов. Если выражение не указывается, то все данные обрабатываются в виде единой группы. В группу попадают записи с одинаковыми значениями колонок, перечисленных после `PARTITION BY`.

Пример

```
PARTITION BY device_id, zone_id
```

Ограничения

Степень поддержки выражения `MATCH_RECOGNIZE` со временем будет соответствовать стандарту SQL-2016, однако сейчас действует следующий набор ограничений:

- `MEASURES`. Не поддерживаются функции `PREV` / `NEXT`.
- `AFTER MATCH SKIP`. Поддерживаются только режимы `AFTER MATCH SKIP TO NEXT ROW` и `AFTER MATCH SKIP PAST LAST ROW`.
- `PATTERN`. Не реализованы Union pattern variables.
- `DEFINE`. Не поддерживаются агрегатные функции.

Пример использования

Разберем практический пример распознавания паттернов в таблице, созданной IoT-устройством с кнопками и событиями, вызываемыми при их активации. Нам нужно найти и обработать следующую последовательность нажатия кнопок: `кнопка 1`, `кнопка 2`, `кнопка 3`.

Структура передаваемых данных:

ts	button	device_id	zone_id
600	3	17	3
500	3	4	2
400	2	17	3
300	2	4	2
200	1	17	3
100	1	4	2

Тело SQL-запроса:

```
PRAGMA FeatureR010="prototype"; -- прагма для включения MATCH_RECOGNIZE

SELECT * FROM input MATCH_RECOGNIZE ( -- выполняем поиск паттернов из таблицы input
    PARTITION BY device_id, zone_id -- разбиваем данные на группы по колонкам device_id и zone_id
    ORDER BY ts -- просматриваем события в порядке возрастания значения колонки ts
    MEASURES
        LAST(B1.ts) AS b1, -- в результатах запроса будем получать последний момент нажатия на кнопку 1
        LAST(B3.ts) AS b3  -- в результатах запроса будем получать последний момент нажатия на кнопку 3
    ONE ROW PER MATCH            -- будем получать одну строку результатов на найденное совпадение
    AFTER MATCH SKIP TO NEXT ROW -- после нахождения совпадения перейдем на следующую строку
    PATTERN (B1 B2+ B3)      -- ищем паттерн, состоящий из одного нажатия на кнопку 1, одного или нескольких нажатий на
кнопку 2 и одного нажатия на кнопку 3
    DEFINE
        B1 AS B1.button = 1, -- определяем переменную B1 как событие нажатия кнопки 1 (значение поля button равно 1)
        B2 AS B2.button = 2, -- определяем переменную B2 как событие нажатия кнопки 2 (значение поля button равно 2)
        B3 AS B3.button = 3  -- определяем переменную B3 как событие нажатия кнопки 3 (значение поля button равно 3)
);
```

Результат:

b1	b3	device_id	zone_id
100	500	4	2
200	600	17	3

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 агрументов `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 экзмепляр 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.

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
  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("P1DT2H3M4.567890S"),
  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), -- "\u0001\u0000\u0000\u0000"
    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; -- в реальности лучше использовать TABLESAMPLE

$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]
```

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 перцентилей
    PERCENTILE(numeric_column, AsStruct(0.01 as p01, 0.5 as median, 0.99 as p99)), -- используя структуру, значениям
перцентиля можно дать удобные имена
    PERCENTILE(numeric_column, ListFromRange(0.00, 1.05, 0.05)),                   -- подсчет множества перцентилей (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

> **Внимание**
>
> 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::IntervalFromDays(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 создавть 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
(неопределенного) 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, [String]])->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)
3. **(Опционально)** `PASSING` секция

PASSING секция

Позволяет передавать значения в JsonPath запрос в качестве переменных.

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

В модуле DateTime основным внутренним форматом представления является `Resource<TM>`, хранящий следующие компоненты даты:

- Year (12 бит);
- 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 во внутреннее представление. Всегда успешно при непустом входе.

Список функций

- `DateTime::Split(Date{Flags:AutoMap}) -> Resource<TM>`
- `DateTime::Split(Datetime{Flags:AutoMap}) -> Resource<TM>`
- `DateTime::Split(Timestamp{Flags:AutoMap}) -> Resource<TM>`
- `DateTime::Split(TzDate{Flags:AutoMap}) -> Resource<TM>`
- `DateTime::Split(TzDatetime{Flags:AutoMap}) -> Resource<TM>`
- `DateTime::Split(TzTimestamp{Flags:AutoMap}) -> Resource<TM>`

Функции, принимающие на вход `Resource<TM>`, 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 из внутреннего представления. Всегда успешна при непустом входе.

Список функций

- `DateTime::MakeDate(Resource<TM>{Flags:AutoMap}) -> Date`
- `DateTime::MakeDatetime(Resource<TM>{Flags:AutoMap}) -> Datetime`
- `DateTime::MakeTimestamp(Resource<TM>{Flags:AutoMap}) -> Timestamp`
- `DateTime::MakeTzDate(Resource<TM>{Flags:AutoMap}) -> TzDate`
- `DateTime::MakeTzDatetime(Resource<TM>{Flags:AutoMap}) -> TzDatetime`
- `DateTime::MakeTzTimestamp(Resource<TM>{Flags:AutoMap}) -> TzTimestamp`

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...

Взятие компоненты внутреннего представления.

Список функций

- `DateTime::GetYear(Resource<TM>{Flags:AutoMap}) -> Uint16`
- `DateTime::GetDayOfYear(Resource<TM>{Flags:AutoMap}) -> Uint16`

- `DateTime::GetMonth(Resource<TM>{Flags:AutoMap}) -> Uint8`
- `DateTime::GetMonthName(Resource<TM>{Flags:AutoMap}) -> String`
- `DateTime::GetWeekOfYear(Resource<TM>{Flags:AutoMap}) -> Uint8`
- `DateTime::GetWeekOfYearIso8601(Resource<TM>{Flags:AutoMap}) -> Uint8`
- `DateTime::GetDayOfMonth(Resource<TM>{Flags:AutoMap}) -> Uint8`
- `DateTime::GetDayOfWeek(Resource<TM>{Flags:AutoMap}) -> Uint8`
- `DateTime::GetDayOfWeekName(Resource<TM>{Flags:AutoMap}) -> String`
- `DateTime::GetHour(Resource<TM>{Flags:AutoMap}) -> Uint8`
- `DateTime::GetMinute(Resource<TM>{Flags:AutoMap}) -> Uint8`
- `DateTime::GetSecond(Resource<TM>{Flags:AutoMap}) -> Uint8`
- `DateTime::GetMillisecondOfSecond(Resource<TM>{Flags:AutoMap}) -> Uint32`
- `DateTime::GetMicrosecondOfSecond(Resource<TM>{Flags:AutoMap}) -> Uint32`
- `DateTime::GetTimezoneId(Resource<TM>{Flags:AutoMap}) -> Uint16`
- `DateTime::GetTimezoneName(Resource<TM>{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:Uint8?,
Second:Uint8?, Microsecond:Uint32?, Timezone:String? ]) -> Resource<TM>?
```

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($tm, "Europe/Moscow" as Timezone)), -- 2018-12-31T22:02:03.456789Z
(конвертация в UTC)
    DateTime::MakeTzTimestamp(DateTime::Update($tm, "Europe/Moscow" as Timezone)); -- 2019-01-
01T01:02:03.456789,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::FromMilliseconds(Uint64{Flags:AutoMap}) -> Timestamp`
- `DateTime::FromMicroseconds(Uint64{Flags:AutoMap}) -> Timestamp`

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::ToMilliseconds(Date/DateTime/Timestamp/TzDate/TzDatetime/TzTimestamp{Flags:AutoMap}) -> Uint64`
- `DateTime::ToMicroseconds(Date/DateTime/Timestamp/TzDate/TzDatetime/TzTimestamp{Flags:AutoMap}) -> Uint64`

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}) -> Int16`
- `DateTime::ToHours(Interval{Flags:AutoMap}) -> Int32`
- `DateTime::ToMinutes(Interval{Flags:AutoMap}) -> Int32`
- `DateTime::ToSeconds(Interval{Flags:AutoMap}) -> Int32`
- `DateTime::ToMilliseconds(Interval{Flags:AutoMap}) -> Int64`
- `DateTime::ToMicroseconds(Interval{Flags:AutoMap}) -> Int64`
- `DateTime::IntervalFromDays(Int16{Flags:AutoMap}) -> Interval`
- `DateTime::IntervalFromHours(Int32{Flags:AutoMap}) -> Interval`
- `DateTime::IntervalFromMinutes(Int32{Flags:AutoMap}) -> Interval`
- `DateTime::IntervalFromSeconds(Int32{Flags:AutoMap}) -> Interval`
- `DateTime::IntervalFromMilliseconds(Int64{Flags:AutoMap}) -> Interval`
- `DateTime::IntervalFromMicroseconds(Int64{Flags:AutoMap}) -> Interval`

Важно

Функция `DateTime::ToSeconds` не поддерживает работу с интервалами с длительностью большей чем 68 лет, в этом случае можно использовать выражение `DateTime::ToMilliseconds(x) / 1000`

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. Поддерживаются периоды до недели, включительно.

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::EndOfYear(Resource<TM>{Flags:AutoMap}) -> Resource<TM>?`
- `DateTime::StartOfQuarter(Resource<TM>{Flags:AutoMap}) -> Resource<TM>?`
- `DateTime::EndOfQuarter(Resource<TM>{Flags:AutoMap}) -> Resource<TM>?`
- `DateTime::StartOfMonth(Resource<TM>{Flags:AutoMap}) -> Resource<TM>?`
- `DateTime::EndOfMonth(Resource<TM>{Flags:AutoMap}) -> Resource<TM>?`
- `DateTime::StartOfWeek(Resource<TM>{Flags:AutoMap}) -> Resource<TM>?`
- `DateTime::EndOfWeek(Resource<TM>{Flags:AutoMap}) -> Resource<TM>?`
- `DateTime::StartOfDay(Resource<TM>{Flags:AutoMap}) -> Resource<TM>?`
- `DateTime::EndOfDay(Resource<TM>{Flags:AutoMap}) -> Resource<TM>?`

- `DateTime::StartOf(Resource<TM>{Flags:AutoMap}, Interval{Flags:AutoMap}) -> Resource<TM>?`
- `DateTime::EndOf(Resource<TM>{Flags:AutoMap}, Interval{Flags:AutoMap}) -> Resource<TM>?`

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`

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::ShiftQuarters(Resource<TM>{Flags:AutoMap}, Int32) -> Resource<TM>?`
- `DateTime::ShiftMonths(Resource<TM>{Flags:AutoMap}, Int32) -> Resource<TM>?`

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<TM>{Flags:AutoMap}) -> String`

Для строки форматирования реализовано множество спецификаторов:

- `%%` - символ %;
- `%Y` - год 4 цифры;
- `%m` - месяц 2 цифры;
- `%d` - день 2 цифры;
- `%H` - час 2 цифры;
- `%M` - минуты 2 цифры;
- `%S` - секунды 2 цифры -- или xx.xxxxxx в случае непустых микросекунд (и только если флаг `alwaysWriteFractionalSeconds` не выставлен в `True`);
- `%z` - +hhmm or -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

Распарсить строку во внутреннее представление, используя произвольную строку форматирования. Для незаполненных полей используются значения по умолчанию. При возникновении ошибок возвращается NULL.

Список функций

- `DateTime::Parse(String) -> (String{Flags:AutoMap}) -> Resource<TM>?`

Реализованные спецификаторы:

- `%%` - символ %;
- `%Y` - год 4 цифры (1970);
- `%m` - месяц 2 цифры (1);
- `%d` - день 2 цифры (1);
- `%H` - час 2 цифры (0);
- `%M` - минуты 2 цифры (0);
- `%S` - секунды (0), может принимать и микросекунды в форматах от xx. до xx.xxxxxx
- `%Z` - IANA имя таймзоны (GMT).
- `%b` - короткое трехбуквенное регистронезависимое английское название месяца (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");

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)
```

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($datetime_parse_tz("2019-09-16 00:00:00" || " Europe/Moscow"))),  --
1568581200
    DateTime::ToSeconds(DateTime::MakeDatetime(DateTime::Update($datetime_parse("2019-09-16 00:00:00"), "Europe/Moscow"
as Timezone))), -- 1568581200

    -- НЕПРАВИЛЬНО (Date импортирует время как GMT, а AddTimezone никак не влияет на ToSeconds, которая всегда
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, -- 1568592000
    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, -- 1568581200
    DateTime::ToSeconds(DateTime::MakeDatetime($datetime_parse_tz("2019-09-16 00:00:00 Europe/Moscow"))) AS md_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 Europe/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, 14282600258804776266)
```

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 выражения
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 не осуществляют полный перебор исходных данных. Это позволяет значительно ускорить процесс поиска, хотя и может привести к некоторому снижению качества результатов.

Скалярное квантование — это метод уменьшения размерности векторов, при котором множество координат отображается в пространство меньшей размерности.

YDB поддерживает векторный поиск по векторам типов `Float`, `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` каждый вектор сокращается в 32 раза.

Приближенный векторный поиск без индекса использует очень простую дополнительную структуру данных - множество векторов с другими 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.

Примечание

В настоящий момент YDB не поддерживает построение векторных индексов для бинарных векторов `BitVector`.

Сигнатуры функций

```
Knn::ToBinaryStringFloat(List<Float>{Flags:AutoMap})->Tagged<String, "FloatVector">
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>?
```

Детали имплементации

`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 вектора 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, "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: CosineDistance,
error: Arguments should have same tags, but 'FloatVector' is not equal to 'Uint8Vector'
```

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), "FloatVector"));
```

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), "FloatVector"),
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, возвращается 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, 99999999999999999999pn), --  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`	
`TzDatetime64`	`text`	
`TzTimestamp64`	`text`	
`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<roaring_bitmap>
Roaring::OrWithBinary(Resource<roaring_bitmap>{Flags:AutoMap}, String{Flags:AutoMap})->Resource<roaring_bitmap>

Roaring::And(Resource<roaring_bitmap>{Flags:AutoMap}, Resource<roaring_bitmap>{Flags:AutoMap})->Resource<roaring_bitmap>
Roaring::AndWithBinary(Resource<roaring_bitmap>{Flags:AutoMap}, String{Flags:AutoMap})->Resource<roaring_bitmap>

Roaring::AndNot(Resource<roaring_bitmap>{Flags:AutoMap}, Resource<roaring_bitmap>{Flags:AutoMap})-
>Resource<roaring_bitmap>
Roaring::AndNotWithBinary(Resource<roaring_bitmap>{Flags:AutoMap}, String{Flags:AutoMap})->Resource<roaring_bitmap>

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:Uint64?,NeverCapture:
->
Struct<CaseSensitive:Bool,DotNl:Bool,Literal:Bool,LogErrors:Bool,LongestMatch:Bool,MaxMem:Uint64,NeverCapture:Bool,NeverNl:
```

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"
```

`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("ҫурт"); -- "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("\u200ыкль\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("ваоао"u, "[вао]"u); -- true
SELECT Unicode::IsUnicodeSet("ваоао"u, "[вабе]"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; по умолчанию
false
  Strict:Bool?,           -- если false - ошибки парсинга игнорируются, ошибочные поля пропускаются; по умолчанию 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; по умолчанию
false
  Strict:Bool?,           -- если false - ошибки парсинга игнорируются, ошибочные поля пропускаются; по умолчанию 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}) ->
Resource<'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 результат, 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

Совместимость YDB с PostgreSQL

Важно

Поддержка синтаксиса PostgreSQL в YDB **находится в разработке**. Использовать её в production окружениях **не рекомендуется**. Запросы в PostgreSQL могут исполняться до нескольких раз медленнее по сравнению с аналогичными запросами на YQL.

Основной сценарий, который можно тестировать — выполнение аналитических запросов к хранимым в YDB данным.

PostgreSQL совместимость – это механизм выполнения SQL запросов в PostgreSQL диалекте на инфраструктуре YDB с использованием сетевого протокола PostgreSQL. Благодаря этой возможности можно использовать привычные инструменты работы с PostgreSQL, такие, как psql и драйвера (например, pq для Golang и psycopg2 для Python). Можно разрабатывать запросы на привычном PostgreSQL синтаксисе и получать такие преимущества YDB, как горизонтальная масштабируемость и отказоустойчивость.

PostgreSQL совместимость упрощает миграцию приложений, ранее работавших в экосистеме PostgreSQL. Сейчас поддерживается ограниченное количество инструкций и функций PostgreSQL 16. PostgreSQL совместимость позволяет переключаться с PostgreSQL на YDB без изменения кода проекта (в случае полной поддержки совместимостью используемых в проекте SQL-конструкций), просто изменив параметры подключения.

В этом разделе описываются:

- Выполнение запросов
- Интероперабельность PostgreSQL и YDB
- PostgreSQL функции
- Импорт данных из PostgreSQL

Выполнение запросов

> **Важно**
>
> Поддержка синтаксиса PostgreSQL в YDB **находится в разработке**. Использовать её в production окружениях **не рекомендуется**. Запросы в PostgreSQL могут исполняться до нескольких раз медленнее по сравнению с аналогичными запросами на YQL.
>
> Основной сценарий, который можно тестировать — выполнение аналитических запросов к хранимым в YDB данным.

Существует несколько способов запуска запросов на диалекте PostgreSQL:

1. Использование нативных PostgreSQL-инструментов. YDB обладает сервисом совместимости для поддержки нативного сетевого протокола PostgreSQL - pgwire, поэтому к YDB можно подключаться стандартными клиентами PostgreSQL.
2. Использование нативных YDB инструментов и явного указания, что запрос выполняется в синтаксисе PostgreSQL с помощью параметров (настроек) этих инструментов при выполнении запросов.
3. Использование любых инструментов, взаимодействующих с YDB и явного указания, что запрос выполняется в синтаксисе PostgreSQL с помощью специального комментария в начале тела запроса `--!syntax_pg`.

Нативный сетевой протокол PostgreSQL

Для подключения к YDB через PostgreSQL-протокол можно использовать любые клиентские библиотеки и приложения, поддерживающие этот протокол. Ниже приведен пример подключения с помощью утилиты `psql`, входящей в комплект поставки PostgreSQL.

Для подключения необходимо выполнить предварительные шаги:

1. Получить адрес кластера, к которому производится подключение.
2. Получить название базы данных в кластере, к которому производится подключение.

Выполнить запрос в YDB:

```
psql -h <ydb_address> -U <user_name> <database_name>
```

Где:

- `<ydb_address>` - адрес кластера YDB, к которому выполняется подключение.
- `<database_name>` - название базы данных в кластере. Может быть сложным именем, например, `mycluster/tenant1/database`.
- `<user_name>` - логин пользователя.

Использование стандартных инструментов YDB

Выполнять запросы на PostgreSQL-диалекте можно с помощью любых инструментов YDB, явно указав в начале тела запроса признак диалекта с помощью специального комментария: `--!syntax_pg`.

Для подключения необходимо выполнить предварительные шаги:

1. Получить адрес кластера, к которому производится подключение.
2. Получить название базы данных в кластере, к которому производится подключение.

Выполнить запрос к YDB:

```
ydb -e <ydb_address> -d <database_name> --user <user_name> sql -s '--!syntax_pg
SELECT 1;
'
```

Где:

- `<ydb_address>` — адрес кластера YDB, к которому выполняется подключение.
- `<database_name>` — название базы данных в кластере. Может быть сложным именем, например, `mycluster/tenant1/database/`.
- `<user_name>` — логин пользователя.

Интероперабельность PostgreSQL и YDB

Важно

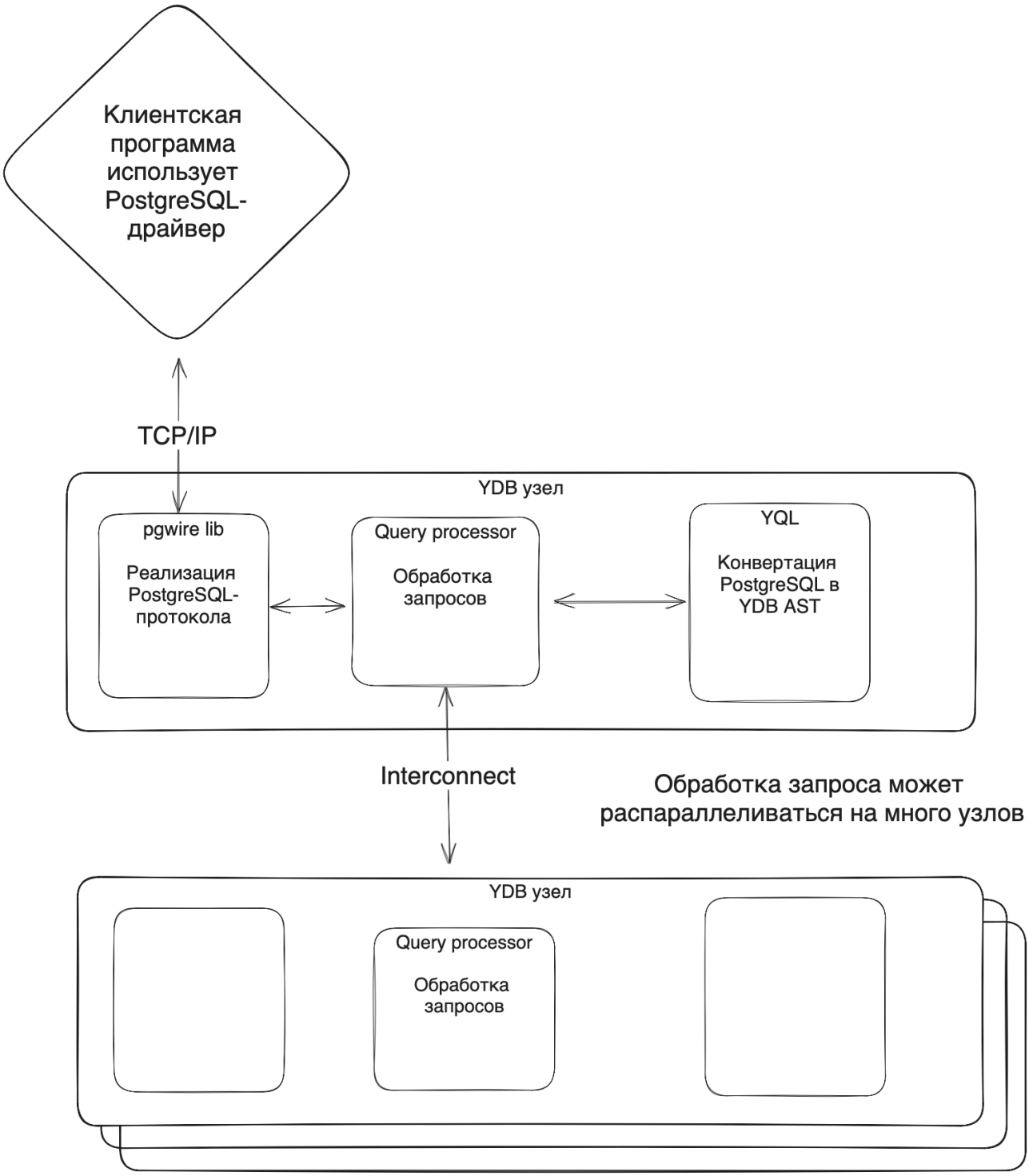
Поддержка синтаксиса PostgreSQL в YDB **находится в разработке**. Использовать её в production окружениях **не рекомендуется**. Запросы в PostgreSQL могут исполняться до нескольких раз медленнее по сравнению с аналогичными запросами на YQL.

Основной сценарий, который можно тестировать — выполнение аналитических запросов к хранимым в YDB данным.

Поддержка исполнения запросов в YDB в синтаксисе PostgreSQL реализована с помощью слоя совместимости:

1. Программа отправляет запросы в YDB, где их обрабатывает компонент под названием `pgwire`. Pgwire реализует сетевой протокол PostgreSQL и передает команды в query processor.
2. Query processor транслирует PostgreSQL запросы в YQL AST.
3. После обработки запросов результаты собираются и отправляются обратно в программу, отправившую запрос, по сетевому протоколу PostgreSQL. При обработке запроса он может распараллеливаться и исполняться на произвольном количестве узлов YDB.

Графически работу PostgreSQL совместимости можно представить так:



Такая архитектура интеграции с PostgreSQL позволяет выполнять запросы на PostgreSQL над YDB типами данных и наоборот, выполнять YQL-запросы над типами данных PostgreSQL, обеспечивая интероперабельность работы с данными.

Проиллюстрируем это с помощью следующего сценария:

1. Создадим таблицу YDB с помощью YQL-синтаксиса

```
CREATE TABLE `test_table`(col INT, PRIMARY KEY(col));
```

2. Добавим туда тестовые данные

```
INSERT INTO test_table(col) VALUES(1)
```

3. Прочитаем эти данные с помощью PostgreSQL-синтаксиса

```
psql -h <ydb_address> -d <database_name> -U <user_name> -c "SELECT * FROM test_table"
col
---
```

```
 1
(1 row)
```

Где:

- `<ydb_address>` - адрес кластера YDB, к которому выполняется подключение.
- `<database_name>` - название базы данных в кластере. Может быть сложным именем, например, `mycluster/tenant1/database`.
- `<user_name>` - логин пользователя.

Соответствие типов данных

Система типов данных YDB и PostgreSQL похожи, но при этом не совпадают.

Использование данных таблиц, созданных в YQL синтаксисе, в синтаксисе PostgreSQL

Типы данных из YDB автоматически преобразовываются в соответствующие им типы PostgreSQL. Преобразование выполняется неявно с помощью команды `ToPg`.

Пример:

1. Создадим таблицу YDB с помощью YQL-синтаксиса

```
CREATE TABLE `test_table`(col INT, PRIMARY KEY(col));
```

2. Добавим туда тестовые данные

```
INSERT INTO test_table(col) VALUES(1)
```

3. Прочитаем эти данные с помощью PostgreSQL-синтаксиса. YQL-тип данных `INT` был автоматически переведен в PostgreSQL-тип `int4`, над которым была выполнена операция инкремента.

```
psql -c "SELECT col+1 AS col FROM test_table"
col
---
2
(1 row)
```

Так как все вычисления выполняются внутри YDB, то для для каждого PostgreSQL типа создан "зеркальный тип" в YDB. Например, тип `text` из PostgreSQL при обработке внутри YDB будет иметь тип `pgtext`. Это сделано, чтобы обеспечить точную семантику работы типов PostgreSQL внутри YDB. При преобразовании типа из PostgreSQL в YDB применяется правило, что для каждого такого типа добавляется префикс `pg`, после чего используется оригинальное имя типа из PostgreSQL.

Таблица соответствия типов данных YQL, при их использовании в PostgreSQL запросах:

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`	
`TzDatetime64`	`text`	
`TzTimestamp64`	`text`	
`Decimal`	`numeric`	`pgnumeric`
`DyNumber`	`numeric`	`pgnumeric`

Использование данных таблиц, созданных в PostgreSQL синтаксисе, в синтаксисе YQL

При использовании данных из таблиц, созданных в PostgreSQL синтаксисе, YQL интерпретирует данные в этих колонках, как специальные типы из семейства `pg*`.

Пример:

1. Создадим таблицу YDB с помощью PostgreSQL-синтаксиса

```
CREATE TABLE test_table_pg(numeric INT, PRIMARY KEY(col));
```

2. Добавим туда тестовые данные

```
INSERT INTO test_table_pg(col) VALUES(10)
```

3. Прочитаем эти данные с помощью YQL-синтаксиса

```
ydb sql -s "SELECT col+1 AS col FROM test_table_pg"
col
---
"11" -- pgnumeric
(1 row)
```

Правила преобразования типов PostgreSQL в типы YQL приведены в таблице:

PostgreSQL	YQL
`bool`	`pgbool`

`int2`	`pgint2`
`int4`	`pgint4`
`int8`	`pgint8`
`numeric`	`pgnumeric`
`float4`	`pgfloat4`
`float8`	`pgfloat8`
`bytea`	`pgbytea`
`text`	`pgtext`
`bytea`	`pgbytea`
`json`	`pgjson`
`uuid`	`pguuid`
`jsonb`	`pgjsonb`
`date`	`pgdate`
`timestamp`	`pgtimestamp`
`interval`	`pginterval`
`text`	`pgtext`
`date`	`pgdate`
`timestamp`	`pgtimestamp`
`interval`	`pginterval`
`numeric`	`pgnumeric`

ваны на работу с собственными типами данных, например, `Ip::FromString` получает на вход типы данных `Utf8` или `String`. Поэтому встроенные функции YQL не могут работать с типами данных PostgreSQL. Для решения задачи конвертации типов существует функция `FromPg`, выполняющая преобразование данных из типов PostgreSQL в типы YQL.

Пример:

1. Создадим таблицу YDB с помощью PostgreSQL-синтаксиса

```
CREATE TABLE test_table_pg_ip(col text, PRIMARY KEY(col));
```

2. Добавим туда тестовые данные

```
INSERT INTO test_table_pg_ip(col) VALUES('::ffff:77.75.155.3')
```

3. Прочитаем эти данные с помощью YQL-синтаксиса:

```
ydb sql -s "SELECT Ip::ToString(Ip::GetSubnet(Ip::FromString(col))) AS subnet
    FROM test_table_pg_ip"
Status: GENERIC_ERROR
Issues:
<main>: Error: Type annotation, code: 1030
    <main>:1:1: Error: At function: RemovePrefixMembers, At function: PersistableRepr, At function: SqlProject, At
function: SqlProjectItem
        <main>:1:12: Error: At function: Apply, Callable is produced by Udf: Ip.FromString
            <main>:1:23: Error: Mismatch type argument #1, type diff: String!=pgtext
```

4. Для работы функции `Ip::FromString` необходимо предварительно выполнить преобразование типов данных с помощью функции `PgFrom`:

```
ydb sql -s "SELECT Ip::ToString(Ip::GetSubnet(Ip::FromString(FromPg(col))) AS subnet
    FROM test_table_pg_ip"
┌─────────┐
```

```
| subnet |
├────────┤
| "::"   |
└────────┘
```

Кроме этого, при работе с PostgreSQL типами данных можно использовать функции самого PostgreSQL непосредственно из YQL-синтаксиса:

1. Создадим таблицу YDB с помощью PostgreSQL-синтаксиса

```
CREATE TABLE test_table_array(col INT, col2 _INT, PRIMARY KEY(col));
```

2. Добавим туда тестовые данные

```
INSERT INTO test_table_array(col, col2) VALUES(1, '{1,2,3}')
```

3. Вызовем встроенную функцию PostgreSQL `array_length`:

```
ydb sql -s "select Pg::array_length(col2, 1) FROM test_table_array"
col
---
"3" -- число элементов в массиве
(1 row)
```

PostgreSQL функции

Важно

Поддержка синтаксиса PostgreSQL в YDB **находится в разработке**. Использовать её в production окружениях **не рекомендуется**. Запросы в PostgreSQL могут исполняться до нескольких раз медленнее по сравнению с аналогичными запросами на YQL.

Основной сценарий, который можно тестировать — выполнение аналитических запросов к хранимым в YDB данным.

В данном разделе содержатся PostgreSQL функции, поддерживаемые в режиме совместимости YDB с PostgreSQL. В разделе сохранены оригинальная структура документации PostgreSQL и примеры применения функций. Данная статья автоматически пополняется и тестируется, поэтому текст представлен только на английском языке.

9.1. Logical Operators

The usual logical operators are available:

- boolean AND boolean → boolean
- boolean OR boolean → boolean
- NOT boolean → boolean

9.2. Comparison Functions and Operators

The usual comparison operators are available, as shown in Table 9.1.

Table 9.1. Comparison Operators

Operator	Description
datatype < datatype → boolean	Less than
datatype > datatype → boolean	Greater than
datatype <= datatype → boolean	Less than or equal to
datatype >= datatype → boolean	Greater than or equal to
datatype = datatype → boolean	Equal
datatype <> datatype → boolean	Not equal
datatype != datatype → boolean	Not equal

There are also some comparison predicates, as shown in Table 9.2. These behave much like operators, but have special syntax mandated by the SQL standard.

Predicate	Description	Example(s)

datatype BETWEEN datatype AND datatype → boolean	Between (inclusive of the range endpoints).	`2 BETWEEN 1 AND 3 → true` `2 BETWEEN 3 AND 1 → false`
datatype NOT BETWEEN datatype AND datatype → boolean	Not between (the negation of BETWEEN).	`2 NOT BETWEEN 1 AND 3 → false`
datatype BETWEEN SYMMETRIC datatype AND datatype → boolean	Between, after sorting the two endpoint values.	`2 BETWEEN SYMMETRIC 3 AND 1 → true`
datatype NOT BETWEEN SYMMETRIC datatype AND datatype → boolean	Not between, after sorting the two endpoint values.	`2 NOT BETWEEN SYMMETRIC 3 AND 1 → false`

datatype IS DISTINCT FROM datatype → boolean	Not equal, treating null as a comparable value. (NOT SUPPORTED)	`#1 IS DISTINCT FROM NULL → true` `#NULL IS DISTINCT FROM NULL → false`
datatype IS NOT DISTINCT FROM datatype → boolean	Equal, treating null as a comparable value. (NOT SUPPORTED)	`#1 IS NOT DISTINCT FROM NULL → false` `#NULL IS NOT DISTINCT FROM NULL → true`
datatype IS NULL → boolean	Test whether value is null.	`1.5 IS NULL → false`
datatype IS NOT NULL → boolean	Test whether value is not null.	`'null' IS NOT NULL → true`
datatype ISNULL → boolean	Test whether value is null (nonstandard syntax)	`1.5 ISNULL → false`

datatype NOTNULL → boolean	Test whether value is not null (nonstandard syntax)	`1.5 NOTNULL → true`

Operator	Description	Example(s)
numeric_type + numeric_type → numeric_type	Addition	`2 + 3 → 5`
+ numeric_type → numeric_type	Unary plus (no operation)	`+ 3.5 → 3.5`
numeric_type - numeric_type → numeric_type	Subtraction	`2 - 3 → -1`
- numeric_type → numeric_type	Negation	`- (-4) → 4`

numeric_type * numeric_type → numeric_type	Multiplication	`2 * 3 → 6`

numeric_type / numeric_type → numeric_type	Division (for integral types, division truncates the result towards zero)	`5.0 / 2 → 2.5000000000000000`
		`5 / 2 → 2`
		`(-5) / 2 → -2`

numeric_type % numeric_type → numeric_type	Modulo (remainder); available for smallint, integer, bigint, and numeric	`5 % 4 → 1`

numeric ^ numeric → numeric
 	
double precision ^ double precision → double precision
	Exponentiation.
	
	Unlike typical mathematical practice, multiple uses of ^ will associate left to right by default.
		`2 ^ 3 → 8`
		`2 ^ 3 ^ 3 → 512`
		`2 ^ (3 ^ 3) → 134217728`

|/ double precision → double precision	Square root	`|/ 25.0 → 5`

||/ double precision → double precision	Cube root	`||/ 64.0 → 4`
@ numeric_type → numeric_type	Absolute value	`@ -5.0 → 5.0`
integral_type & integral_type → integral_type	Bitwise AND	`91 & 15 → 11`
integral_type | integral_type → integral_type	Bitwise OR	`32 | 3 → 35`
integral_type # integral_type → integral_type	Bitwise exclusive OR	`17 # 5 → 20`

~ integral_type → integral_type	Bitwise NOT	`~1 → -2`
integral_type << integer → integral_type	Bitwise shift left	`1 << 4 → 16`
integral_type >> integer → integral_type	Bitwise shift right	`8 >> 2 → 2`

Function	Description	Example(s)
abs (numeric_type) → numeric_type	Absolute value	`abs(-17.4) → 17.4`
cbrt (double precision) → double precision	Cube root	`cbrt(64.0) → 4`

ceil (numeric) → numeric ceil (double precision) → double precision	Nearest integer greater than or equal to argument	`ceil(42.2) → 43` `ceil(-42.8) → -42`
ceiling (numeric) → numeric ceiling (double precision) → double precision	Nearest integer greater than or equal to argument(same as ceil)	`ceiling(95.3) → 96`
degrees (double precision) → double precision	Converts radians to degrees	`degrees(0.5) → 28.64788975654116`
div (y numeric, x numeric) → numeric	Integer quotient of y/x (truncates towards zero)	`div(9, 4) → 2`
exp (numeric) → numeric exp (double precision) → double precision	Exponential (e raised to the given power)	`exp(1.0) → 2.7182818284590452`

factorial (bigint) → numeric	Factorial	`factorial(5) → 120`
floor (numeric) → numeric floor (double precision) → double precision	Nearest integer less than or equal to argument	`floor(42.8) → 42` `floor(-42.8) → -43`
gcd (numeric_type, numeric_type) → numeric_type	Greatest common divisor (the largest positive number that divides both inputs with no remainder); returns 0 if both inputs are zero; available for integer, bigint, and numeric	`gcd(1071, 462) → 21`
lcm (numeric_type, numeric_type) → numeric_type	Least common multiple (the smallest strictly positive number that is an integral multiple of both inputs); returns 0 if either input is zero; available for integer, bigint, and numeric	`lcm(1071, 462) → 23562`
ln (numeric) → numeric ln (double precision) → double precision	Natural logarithm	`ln(2.0) → 0.6931471805599453`

log (numeric) → numeric log (double precision) → double precision	Base 10 logarithm	`log(100) → 2`
log10 (numeric) → numeric log10 (double precision) → double precision	Base 10 logarithm (same as log)	`log10(1000) → 3`
log (b numeric, x numeric) → numeric	Logarithm of x to base b	`log(2.0, 64.0) → 6.0000000000000000`
min_scale (numeric) → integer	Minimum scale (number of fractional decimal digits) needed to represent the supplied value precisely	`min_scale(8.4100) → 2`
mod (y numeric_type, x numeric_type) → numeric_type	Remainder of y/x; available for smallint, integer, bigint, and numeric	`mod(9, 4) → 1`

pi ( ) → double precision	Approximate value of π	`pi() → 3.141592653589793`
power (a numeric, b numeric) → numeric power (a double precision, b double precision) → double precision	a raised to the power of b	`power(9, 3) → 729`
radians (double precision) → double precision	Converts degrees to radians	`radians(45.0) → 0.7853981633974483`
round (numeric) → numeric round (double precision) → double precision	Rounds to nearest integer. For numeric, ties are broken by rounding away from zero. For double precision, the tie-breaking behavior is platform dependent, but “round to nearest even” is the most common rule.	`round(42.4) → 42`
round (v numeric, s integer) → numeric	Rounds v to s decimal places. Ties are broken by rounding away from zero.	`round(42.4382, 2) → 42.44` `round(1234.56, -1) → 1230`

Function	Description	Example(s)
random ( ) → double precision	Returns a random value in the range 0.0 <= x < 1.0	
setseed (double precision) → void	Sets the seed for subsequent random() calls; argument must be between -1.0 and 1.0, inclusive (NOT SUPPORTED)	

Table 9.7 shows the available trigonometric functions. Each of these functions comes in two variants, one that measures angles in radians and one that measures angles in degrees.

Table 9.7. Trigonometric Functions

Function	Description	Example(s)
acos (double precision) → double precision	Inverse cosine, result in radians	`acos(1) → 0`
acosd (double precision) → double precision	Inverse cosine, result in degrees	`acosd(0.5) → 60`
asin (double precision) → double precision	Inverse sine, result in radians	`asin(1) → 1.5707963267948966`
asind (double precision) → double precision	Inverse sine, result in degrees	`asind(0.5) → 30`
atan (double precision) → double precision	Inverse tangent, result in radians	`atan(1) → 0.7853981633974483`

atand (double precision) → double precision	Inverse tangent, result in degrees	`atand(1) → 45`
atan2 (y double precision, x double precision) → double precision	Inverse tangent of y/x, result in radians	`atan2(1, 0) → 1.5707963267948966`
atan2d (y double precision, x double precision) → double precision	Inverse tangent of y/x, result in degrees	`atan2d(1, 0) → 90`
cos (double precision) → double precision	Cosine, argument in radians	`cos(0) → 1`
cosd (double precision) → double precision	Cosine, argument in degrees	`cosd(60) → 0.5`

cot (double precision) → double precision	Cotangent, argument in radians	`cot(0.5) → 1.830487721712452`
cotd (double precision) → double precision	Cotangent, argument in degrees	`cotd(45) → 1`
sin (double precision) → double precision	Sine, argument in radians	`sin(1) → 0.8414709848078965`
sind (double precision) → double precision	Sine, argument in degrees	`sind(30) → 0.5`
tan (double precision) → double precision	Tangent, argument in radians	`tan(1) → 1.5574077246549023`

tand (double precision) → double precision	Tangent, argument in degrees	

Function	Description	Example(s)
sinh (double precision) → double precision	Hyperbolic sine	`sinh(1) → 1.1752011936438014`
cosh (double precision) → double precision	Hyperbolic cosine	`cosh(0) → 1`
tanh (double precision) → double precision	Hyperbolic tangent	`tanh(1) → 0.7615941559557649`
asinh (double precision) → double precision	Inverse hyperbolic sine	`asinh(1) → 0.881373587019543`

acosh (double precision) → double precision	Inverse hyperbolic cosine	`acosh(1) → 0`
atanh (double precision) → double precision	Inverse hyperbolic tangent	`atanh(0.5) → 0.5493061443340548`

the types
. They will
r is applied,

SQL also

Function/Operator	Description	Example(s)
text || text → text	Concatenates the two strings	`'Post' || 'greSQL' → PostgreSQL`
text || anynonarray → text anynonarray || text → text	Converts the non-string input to text, then concatenates the two strings. (The non-string input cannot be of an array type, because that would create ambiguity with the array || operators. If you want to concatenate an array's text equivalent, cast it to text explicitly.) (UNSUPPORTED)	`#'Value: ' || 42 → Value: 42`

text IS [NOT] [form] NORMALIZED → boolean	Checks whether the string is in the specified Unicode normalization form. The optional form key word specifies the form: NFC (the default), NFD, NFKC, or NFKD. This expression can only be used when the server encoding is UTF8. Note that checking for normalization using this expression is often faster than normalizing possibly already normalized strings.	`U&'\0061\0308bc' IS NFD NORMALIZED → true`
bit_length (text) → integer	Returns number of bits in the string (8 times the octet_length)	`bit_length('jose') → 32`
char_length (text) → integer character_length (text) → integer	Returns number of characters in the string.	`char_length('josé') → 4`
lower (text) → text	Converts the string to all lower case, according to the rules of the database's locale.	`lower('TOM') → tom`
normalize (text [, form]) → text	Converts the string to the specified Unicode normalization form. The optional form key word specifies the form: NFC (the default), NFD, NFKC, or NFKD. This function can only be used when the server encoding is UTF8.	`normalize(U&'\0061\0308bc', NFC) → äbc`

octet_length (text) → integer	Returns number of bytes in the string.	`octet_length('josé') → 5`
octet_length (character) → integer	Returns number of bytes in the string. Since this version of the function accepts type character directly, it will not strip trailing spaces.	`octet_length('abc '::character(4)) → 4`
overlay (string text PLACING newsubstring text FROM start integer [FOR count integer]) → text	Replaces the substring of string that starts at the start'th character and extends for count characters with newsubstring. If count is omitted, it defaults to the length of newsubstring.	`overlay('Txxxxas' placing 'hom' from 2 for 4) → Thomas`
position (substring text IN string text) → integer	Returns first starting index of the specified substring within string, or zero if it's not present.	`position('om' in 'Thomas') → 3`

substring (string text [FROM start integer] [FOR count integer]) → text	Extracts the substring of string starting at the start'th character if that is specified, and stopping after count characters if that is specified. Provide at least one of start and count.	`substring('Thomas' from 2 for 3) → hom`
		`substring('Thomas' from 3) → omas`
		`substring('Thomas' for 2) → Th`

substring (string text FROM pattern text) → text	Extracts the first substring matching POSIX regular expression; see Section 9.7.3.	`substring('Thomas' from '...$') → mas`
substring (string text SIMILAR pattern text ESCAPE escape text) → text substring (string text FROM pattern text FOR escape text) → text	Extracts the first substring matching SQL regular expression; see Section 9.7.2. The first form has been specified since SQL:2003; the second form was only in SQL:1999 and should be considered obsolete.	`substring('Thomas' similar '%#"o_a#"_' escape '#') → oma`
trim ([LEADING | TRAILING | BOTH] [characters text] FROM string text) → text	Removes the longest string containing only characters in characters (a space by default) from the start, end, or both ends (BOTH is the default) of string.	`trim(both 'xyz' from`

ent the SQL-standard

Function	Description	Example(s)
ascii (text) → integer	Returns the numeric code of the first character of the argument. In UTF8 encoding, returns the Unicode code point of the character. In other multibyte encodings, the argument must be an ASCII character.	`ascii('x') → 120`
btrim (string text [, characters text]) → text	Removes the longest string containing only characters in characters (a space by default) from the start and end of string.	`btrim('xyxtrimyyx', 'xyz') → trim`

chr (integer) → text	Returns the character with the given code. In UTF8 encoding the argument is treated as a Unicode code point. In other multibyte encodings the argument must designate an ASCII character. chr(0) is disallowed because text data types cannot store that character.	`chr(65) → A`
concat (val1 "any" [, val2 "any" [, ...]]) → text	Concatenates the text representations of all the arguments. NULL arguments are ignored. (NOT SUPPORTED)	`concat('abcde', 2, NULL, 22) → abcde222`
concat_ws (sep text, val1 "any" [, val2 "any" [, ...]]) → text	Concatenates all but the first argument, with separators. The first argument is used as the separator string, and should not be NULL. Other NULL arguments are ignored. (NOT SUPPORTED)	`concat_ws(',', 'abcde', 2, NULL, 22) → abcde,2,22`
format (formatstr text [, formatarg "any" [, ...]]) → text	Formats arguments according to a format string; see Section 9.4.1. This function is similar to the C function sprintf. (NOT SUPPORTED)	`format('Hello %s, %1$s', 'World') → Hello World, World`
initcap (text) → text	Converts the first letter of each word to upper case and the rest to lower case. Words are sequences of alphanumeric characters separated by non-alphanumeric characters.	`initcap('hi THOMAS') → Hi Thomas`

left (string text, n integer) → text	Returns first n characters in the string, or when n is negative, returns all but last |n| characters.	`left('abcde', 2) → ab`
length (text) → integer	Returns the number of characters in the string.	`length('jose') → 4`
lpad (string text, length integer [, fill text]) → text	Extends the string to length length by prepending the characters fill (a space by default). If the string is already longer than length then it is truncated (on the right).	`lpad('hi', 5, 'xy') → xyxhi`
ltrim (string text [, characters text]) → text	Removes the longest string containing only characters in characters (a space by default) from the start of string.	`ltrim('zzzytest', 'xyz') → test`
md5 (text) → text	Computes the MD5 hash of the argument, with the result written in hexadecimal.	`md5('abc') → 900150983cd24fb0d6963f7d28e17f72`

parse_ident (qualified_identifier text [, strict_mode boolean DEFAULT true]) → text[]	Splits qualified_identifier into an array of identifiers, removing any quoting of individual identifiers. By default, extra characters after the last identifier are considered an error; but if the second parameter is false, then such extra characters are ignored. (This behavior is useful for parsing names for objects like functions.) Note that this function does not truncate over-length identifiers. If you want truncation you can cast the result to name[].	`parse_ident('"SomeSchema".someTable') → {SomeSchema,someTable}`
pg_client_encoding ( ) → name	Returns current client encoding name.	`pg_client_encoding() → UTF8`
quote_ident (text) → text	Returns the given string suitably quoted to be used as an identifier in an SQL statement string. Quotes are added only if necessary (i.e., if the string contains non-identifier characters or would be case-folded). Embedded quotes are properly doubled. See also Example 43.1.	`quote_ident('Foo bar') → "Foo bar"`
quote_literal (text) → text	Returns the given string suitably quoted to be used as a string literal in an SQL statement string. Embedded single-quotes and backslashes are properly doubled. Note that quote_literal returns null on null input; if the argument might be null, quote_nullable is often more suitable. See also Example 43.1.	`quote_literal(E'O\'Reilly') → ''O''Reilly''`
quote_literal (anyelement) → text	Converts the given value to text and then quotes it as a literal. Embedded single-quotes and backslashes are properly doubled. (NOT SUPPORTED)	`#quote_literal(42.5) → '42.5'`

quote_nullable (text) → text	Returns the given string suitably quoted to be used as a string literal in an SQL statement string; or, if the argument is null, returns NULL. Embedded single-quotes and backslashes are properly doubled. See also Example 43.1.	`quote_nullable(NULL) → NULL`
quote_nullable (anyelement) → text	Converts the given value to text and then quotes it as a literal; or, if the argument is null, returns NULL. Embedded single-quotes and backslashes are properly doubled. (NOT SUPPORTED)	`#quote_nullable(42.5) → '42.5'`
regexp_match (string text, pattern text [, flags text]) → text[]	Returns captured substrings resulting from the first match of a POSIX regular expression to the string; see Section 9.7.3.	`regexp_match('foobarbequebaz', '(bar)(beque)') → {bar,beque}`
regexp_matches (string text, pattern text [, flags text]) → setof text[]	Returns captured substrings resulting from the first match of a POSIX regular expression to the string, or multiple matches if the g flag is used; see Section 9.7.3. (NOT SUPPORTED)	`#regexp_matches('foobarbequebaz', 'ba.', 'g') → {bar},{baz}`
regexp_replace (string text, pattern text, replacement text [, flags text]) → text	Replaces substrings resulting from the first match of a POSIX regular expression, or multiple substring matches if the g flag is used; see Section 9.7.3.	`regexp_replace('Thomas', '.[mN]a.', 'M') → ThM`

regexp_split_to_array (string text, pattern text [, flags text]) → text[]	Splits string using a POSIX regular expression as the delimiter, producing an array of results; see Section 9.7.3.	`regexp_split_to_array('hello world', '\s+') → {hello,world}`
regexp_split_to_table (string text, pattern text [, flags text]) → setof text	Splits string using a POSIX regular expression as the delimiter, producing a set of results; see Section 9.7.3. (NOT SUPPORTED)	`#regexp_split_to_table('hello world', '\s+') → hello,world`
repeat (string text, number integer) → text	Repeats string the specified number of times.	`repeat('Pg', 4) → PgPgPgPg`
replace (string text, from text, to text) → text	Replaces all occurrences in string of substring from with substring to.	`replace('abcdefabcdef', 'cd', 'XX') → abXXefabXXef`
reverse (text) → text	Reverses the order of the characters in the string.	`reverse('abcde') → edcba`

right (string text, n integer) → text	Returns last n characters in the string, or when n is negative, returns all but first |n| characters.	`right('abcde', 2) → de`
rpad (string text, length integer [, fill text]) → text	Extends the string to length length by appending the characters fill (a space by default). If the string is already longer than length then it is truncated.	`rpad('hi', 5, 'xy') → hixyx`
rtrim (string text [, characters text]) → text	Removes the longest string containing only characters in characters (a space by default) from the end of string.	`rtrim('testxxzx', 'xyz') → test`
split_part (string text, delimiter text, n integer) → text	Splits string at occurrences of delimiter and returns the n'th field (counting from one), or when n is negative, returns the |n|'th-from-last field.	`split_part('abc~@~def~@~ghi', '~@~', 2) → def` `split_part('abc,def,ghi,jkl', ',', -2) → ghi`
strpos (string text, substring text) → integer	Returns first starting index of the specified substring within string, or zero if it's not present. (Same as position(substring in string), but note the reversed argument order.)	`strpos('high', 'ig') → 2`

substr (string text, start integer [, count integer]) → text	Extracts the substring of string starting at the start'th character, and extending for count characters if that is specified. (Same as substring(string from start for count).)	`substr('alphabet', 3) → phabet` `substr('alphabet', 3, 2) → ph`
starts_with (string text, prefix text) → boolean	Returns true if string starts with prefix.	`starts_with('alphabet', 'alph') → true`
string_to_array (string text, delimiter text [, null_string text]) → text[]	Splits the string at occurrences of delimiter and forms the resulting fields into a text array. If delimiter is NULL, each character in the string will become a separate element in the array. If delimiter is an empty string, then the string is treated as a single field. If null_string is supplied and is not NULL, fields matching that string are replaced by NULL.	`string_to_array('xx~~yy~~zz', '~~', 'yy') → {xx,NULL,zz}`
string_to_table (string text, delimiter text [, null_string text]) → setof text	Splits the string at occurrences of delimiter and returns the resulting fields as a set of text rows. If delimiter is NULL, each character in the string will become a separate row of the result. If delimiter is an empty string, then the string is treated as a single field. If null_string is supplied and is not NULL, fields matching that string are replaced by NULL. (NOT SUPPORTED)	`#string_to_table('xx~^~yy~^~zz', '~^~', 'yy') → [xx,NULL,zz]`
to_ascii (string text) → text to_ascii (string text, encoding name) → text to_ascii (string text	Converts string to ASCII from another encoding, which may be identified by name or number. If encoding is omitted the database encoding is assumed (which in practice is the only useful case). The conversion consists primarily of dropping accents. Conversion is only supported from LATIN1, LATIN2, LATIN9, and WIN1250	`#to_ascii('Karél') → Karel`

ytea. Many of these are

able 9.11. PostgreSQL also

Function/Operator	Description	Example(s)
bytea || bytea → bytea	Concatenates the two binary strings.	`'\x123456'::bytea || '\x789a00bcde'::bytea → \x123456789a00bcde`

bit_length (bytea) → integer	Returns number of bits in the binary string (8 times the octet_length).	`bit_length('\x123456'::bytea) → 24`
octet_length (bytea) → integer	Returns number of bytes in the binary string.	`octet_length('\x123456'::bytea) → 3`
overlay (bytes bytea PLACING newsubstring bytea FROM start integer [FOR count integer]) → bytea	Replaces the substring of bytes that starts at the start'th byte and extends for count bytes with newsubstring. If count is omitted, it defaults to the length of newsubstring.	`overlay('\x1234567890'::bytea placing '\002\003'::bytea from 2 for 3) → \x12020390`
position (substring bytea IN bytes bytea) → integer	Returns first starting index of the specified substring within bytes, or zero if it's not present.	`position('\x5678'::bytea in '\x1234567890'::bytea) → 3`
substring (bytes bytea [FROM start integer] [FOR count integer]) → bytea	Extracts the substring of bytes starting at the start'th byte if that is specified, and stopping after count bytes if that is specified. Provide at least one of start and count.	`substring('\x1234567890'::bytea from 3 for 2) → \x5678`

Additional binary string manipulation functions are available and are listed in Table 9.12. Some of them are used internally to implement the SQL-standard string functions listed in Table 9.11.

Table 9.12. Other Binary String Functions

Function	Description	Example(s)

bit_count (bytes bytea) → bigint	Returns the number of bits set in the binary string (also known as "popcount").	`bit_count('\x1234567890'::bytea) → 15`
btrim (bytes bytea, bytesremoved bytea) → bytea	Removes the longest string containing only bytes appearing in bytesremoved from the start and end of bytes.	`btrim('\x1234567890'::bytea, '\x9012'::bytea) → \x345678`
get_bit (bytes bytea, n bigint) → integer	Extracts n'th bit from binary string.	`get_bit('\x1234567890'::bytea, 30) → 1`
get_byte (bytes bytea, n integer) → integer	Extracts n'th byte from binary string.	`get_byte('\x1234567890'::bytea, 4) → 144`
length (bytea) → integer	Returns the number of bytes in the binary string.	`length('\x1234567890'::bytea) → 5`

length (bytes bytea, encoding name) → integer	Returns the number of characters in the binary string, assuming that it is text in the given encoding.	`length('jose'::bytea, 'UTF8') → 4`
ltrim (bytes bytea, bytesremoved bytea) → bytea	Removes the longest string containing only bytes appearing in bytesremoved from the start of bytes.	`ltrim('\x1234567890'::bytea, '\x9012'::bytea) → \x34567890`
md5 (bytea) → text	Computes the MD5 hash of the binary string, with the result written in hexadecimal.	`md5('Th\000omas'::bytea) → 8ab2d3c9689aaf18b4958c334c82d8b1`
rtrim (bytes bytea, bytesremoved bytea) → bytea	Removes the longest string containing only bytes appearing in bytesremoved from the end of bytes.	`rtrim('\x1234567890'::bytea, '\x9012'::bytea) → \x12345678`
set_bit (bytes bytea, n bigint, newvalue integer) → bytea	Sets n'th bit in binary string to newvalue.	`set_bit('\x1234567890'::bytea, 30, 0) → \x1234563890`

t_bit and set_bit number bits from the right within
nificant bit of the second byte.

SHA-2 functions return type bytea. Use the functions
) to get a hex-encoded text representation, or

ing arbitrary binary data in textual form, are shown
se's default encoding, while arguments or results of

Function	Description	Example(s)
convert (bytes bytea, src_encoding name, dest_encoding name) → bytea	Converts a binary string representing text in encoding src_encoding to a binary string in encoding dest_encoding (see Section 24.3.4 for available conversions). (NOT SUPPORTED)	`#convert('text_in_utf8', 'UTF8', 'LATIN1') → \x746578745f696e5f75746638`
convert_from (bytes bytea, src_encoding name) → text	Converts a binary string representing text in encoding src_encoding to text in the database encoding (see Section 24.3.4 for available conversions).	`convert_from('text_in_utf8', 'UTF8') → text_in_utf8`

convert_to (string text, dest_encoding name) → bytea	Converts a text string (in the database encoding) to a binary string encoded in encoding dest_encoding (see Section 24.3.4 for available conversions).	`convert_to('some_text', 'UTF8') → \x736f6d655f74657874`
encode (bytes bytea, format text) → text	Encodes binary data into a textual representation; supported format values are: base64, escape, hex.	`encode('123\000\001', 'base64') → MTIzAAE=`
decode (string text, format text) → bytea	Decodes binary data from a textual representation; supported format values are the same as for encode.	`ase64') →`

bit and bit varying. (While only usual comparison operators

Operator	Description	Example(s)
Concatenation		
Bitwise AND (inputs must be of equal length)		
Bitwise OR (inputs must be of equal length)		
Bitwise exclusive OR (inputs must be of equal length)		
Bitwise NOT		
Bitwise shift left (string length is preserved)		
Bitwise shift right (string length is preserved)		

Some of the functions available for binary strings are also available for bit strings, as shown in Table 9.15.

Table 9.15. Bit String Functions

Function	Description	Example(s)

bit_count (bit) → bigint	Returns the number of bits set in the bit string (also known as “popcount”).	`bit_count(B'10111') → 4`
bit_length (bit) → integer	Returns number of bits in the bit string.	`bit_length(B'10111') → 5`
length (bit) → integer	Returns number of bits in the bit string.	`length(B'10111') → 5`
octet_length (bit) → integer	Returns number of bytes in the bit string.	`octet_length(B'1011111011') → 2`
overlay (bits bit PLACING newsubstring bit FROM start integer [FOR count integer]) → bit	Replaces the substring of bits that starts at the start'th bit and extends for count bits with newsubstring. If count is omitted, it defaults to the length of newsubstring.	`overlay(B'01010101010101010' placing B'11111' from 2 for 3) → 0111110101010101010`

position (substring bit IN bits bit) → integer	Returns first starting index of the specified substring within bits, or zero if it's not present.	`position(B'010' in B'000001101011') → 8`
substring (bits bit [FROM start integer] [FOR count integer]) → bit	Extracts the substring of bits starting at the start'th bit if that is specified, and stopping after count bits if that is specified. Provide at least one of start and count.	`substring(B'110010111111' from 3 for 2) → 00`
get_bit (bits bit, n integer) → integer	Extracts n'th bit from bit string; the first (leftmost) bit is bit 0.	`get_bit(B'101010101010101010', 6) → 1`
set_bit (bits bit, n integer, newvalue integer) → bit	Sets n'th bit in bit string to newvalue; the first (leftmost) bit is bit 0.	`set_bit(B'101010101010101010', 6, 0) → 101010001010101010`

ghtmost n bits. Casting an integer to a

eger.

pression returns false if LIKE returns

that case LIKE acts like the equals
s any sequence of zero or more

LIKE pattern matching always covers the entire string. Therefore, if it's desired to match a sequence anywhere within a string, the pattern must start and end with a percent sign.

To match a literal underscore or percent sign without matching other characters, the respective character in pattern must be preceded by the escape character. The default escape character is the backslash but a different one can be selected by using the ESCAPE clause. To match the escape character itself, write two escape characters.

Note
If you have standard_conforming_strings turned off, any backslashes you write in literal string constants will need to be doubled. See Section 4.1.2.1 for more information.

It's also possible to select no escape character by writing ESCAPE ''. This effectively disables the escape mechanism, which makes it impossible to turn off the special meaning of underscore and percent signs in the pattern.

According to the SQL standard, omitting ESCAPE means there is no escape character (rather than defaulting to a backslash), and a zero-length ESCAPE value is disallowed. PostgreSQL's behavior in this regard is therefore slightly nonstandard.

The key word ILIKE can be used instead of LIKE to make the match case-insensitive according to the active locale. This is not in the SQL standard but is a PostgreSQL extension.

The operator `~~` is equivalent to LIKE, and `~~*` corresponds to ILIKE. There are also `!~~` and `!~~*` operators that represent NOT LIKE and NOT ILIKE, respectively. All of these operators are PostgreSQL-specific. You may see these operator names in EXPLAIN output and similar places, since the parser actually translates LIKE et al. to these operators.

The phrases LIKE, ILIKE, NOT LIKE, and NOT ILIKE are generally treated as operators in PostgreSQL syntax; for example they can be used in expression operator ANY (subquery) constructs, although an ESCAPE clause cannot be included there. In some obscure cases it may be necessary to use the underlying operator names instead.

Also see the prefix operator ^@ and corresponding starts_with function, which are useful in cases where simply matching the beginning of a string is needed.

9.7.2. SIMILAR TO Regular Expressions

```
string SIMILAR TO pattern [ESCAPE escape-character] (NOT SUPPORTED)
string NOT SIMILAR TO pattern [ESCAPE escape-character] (NOT SUPPORTED)
```

The SIMILAR TO operator returns true or false depending on whether its pattern matches the given string. It is similar to LIKE, except that it interprets the pattern using the SQL standard's definition of a regular expression. SQL regular expressions are a curious cross between LIKE notation and common (POSIX) regular expression notation.

Like LIKE, the SIMILAR TO operator succeeds only if its pattern matches the entire string; this is unlike common regular expression behavior where the pattern can match any part of the string. Also like LIKE, SIMILAR TO uses _ and % as wildcard characters denoting any single character and any string, respectively (these are comparable to . and .* in POSIX regular expressions).

In addition to these facilities borrowed from LIKE, SIMILAR TO supports these pattern-matching metacharacters borrowed from POSIX regular expressions:

| denotes alternation (either of two alternatives).

- denotes repetition of the previous item zero or more times.
- denotes repetition of the previous item one or more times.

? denotes repetition of the previous item zero or one time.

{m} denotes repetition of the previous item exactly m times.

{m,} denotes repetition of the previous item m or more times.

{m,n} denotes repetition of the previous item at least m and not more than n times.

Parentheses () can be used to group items into a single logical item.

A bracket expression [...] specifies a character class, just as in POSIX regular expressions.

Notice that the period (.) is not a metacharacter for SIMILAR TO.

As with LIKE, a backslash disables the special meaning of any of these metacharacters. A different escape character can be specified with ESCAPE, or the escape capability can be disabled by writing ESCAPE ''.

According to the SQL standard, omitting ESCAPE means there is no escape character (rather than defaulting to a backslash), and a zero-length ESCAPE value is disallowed. PostgreSQL's behavior in this regard is therefore slightly nonstandard.

Another nonstandard extension is that following the escape character with a letter or digit provides access to the escape sequences defined for POSIX regular expressions; see Table 9.20, Table 9.21, and Table 9.22 below.

Some examples:

```
#'abc' SIMILAR TO 'abc'          → true
#'abc' SIMILAR TO 'a'            → false
#'abc' SIMILAR TO '%(b|d)%'      → true
#'abc' SIMILAR TO '(b|c)%'       → false
#'-abc-' SIMILAR TO '%\mabc\M%'  → true
#'xabcy' SIMILAR TO '%\mabc\M%'  → false
```

The substring function with three parameters provides extraction of a substring that matches an SQL regular expression pattern. The function can be written according to standard SQL syntax:

```
substring(string similar pattern escape escape-character)
```

or using the now obsolete SQL:1999 syntax:

```
substring(string from pattern for escape-character)
```

or as a plain three-argument function:

```
substring(string, pattern, escape-character)
```

As with SIMILAR TO, the specified pattern must match the entire data string, or else the function fails and returns null. To indicate the part of the pattern for which the matching data sub-string is of interest, the pattern should contain two occurrences of the escape character followed by a double quote ("). The text matching the portion of the pattern between these separators is returned when the match is successful.

The escape-double-quote separators actually divide substring's pattern into three independent regular expressions; for example, a vertical bar (|) in any of the three sections affects only that section. Also, the first and third of these regular expressions are defined to match the smallest possible amount of text, not the largest, when there is any ambiguity about how much of the data string matches which pattern. (In POSIX parlance, the first and third regular expressions are forced to be non-greedy.)

As an extension to the SQL standard, PostgreSQL allows there to be just one escape-double-quote separator, in which case the third regular expression is taken as empty; or no separators, in which case the first and third regular expressions are taken as empty.

Some examples, with #" delimiting the return string:

```
substring('foobar' similar '%#"o_b#"%' escape '#')   → oob
substring('foobar' similar '#"o_b#"%' escape '#')    → NULL
```

9.7.3. POSIX Regular Expressions
Table 9.16 lists the available operators for pattern matching using POSIX regular expressions.

Table 9.16. Regular Expression Match Operators

Operator	Description	Example(s)
ext ~ text → boolean	String matches regular expression, case sensitively	`'thomas' ~ 't.*ma' → true`
text ~* text → boolean	String matches regular expression, case insensitively	`'thomas' ~* 'T.*ma' → true`
text !~ text → boolean	String does not match regular expression, case sensitively	`'thomas' !~ 't.*max' → true`
text !~* text → boolean	String does not match regular expression, case insensitively	`'thomas' !~* 'T.*ma' → false`

POSIX regular expressions provide a more powerful means for pattern matching than the LIKE and SIMILAR TO operators. Many Unix tools such as egrep, sed, or awk use a pattern matching language that is similar to the one described here.

A regular expression is a character sequence that is an abbreviated definition of a set of strings (a regular set). A string is said to match a regular expression if it is a member of the regular set described by the regular expression. As with LIKE, pattern characters match string characters exactly

unless they are special characters in the regular expression language — but regular expressions use different special characters than LIKE does. Unlike LIKE patterns, a regular expression is allowed to match anywhere within a string, unless the regular expression is explicitly anchored to the beginning or end of the string.

Some examples:

```
'abcd' ~ 'bc'     → true
'abcd' ~ 'a.c'    → true /* dot matches any character */
'abcd' ~ 'a.*d'   → true /* * repeats the preceding pattern item */
'abcd' ~ '(b|x)'  → true /* | means OR, parentheses group */
'abcd' ~ '^a'     → true /* ^ anchors to start of string */
'abcd' ~ '^(b|c)' → false /* would match except for anchoring */
```

The POSIX pattern language is described in much greater detail below.

The substring function with two parameters, substring(string from pattern), provides extraction of a substring that matches a POSIX regular expression pattern. It returns null if there is no match, otherwise the first portion of the text that matched the pattern. But if the pattern contains any parentheses, the portion of the text that matched the first parenthesized subexpression (the one whose left parenthesis comes first) is returned. You can put parentheses around the whole expression if you want to use parentheses within it without triggering this exception. If you need parentheses in the pattern before the subexpression you want to extract, see the non-capturing parentheses described below.

Some examples:

```
substring('foobar' from 'o.b')      → oob
substring('foobar' from 'o(.)b')    → o
```

The regexp_replace function provides substitution of new text for substrings that match POSIX regular expression patterns. It has the syntax regexp_replace(source, pattern, replacement [, flags]). The source string is returned unchanged if there is no match to the pattern. If there is a match, the source string is returned with the replacement string substituted for the matching substring. The replacement string can contain \n, where n is 1 through 9, to indicate that the source substring matching the n'th parenthesized subexpression of the pattern should be inserted, and it can contain & to indicate that the substring matching the entire pattern should be inserted. Write \ if you need to put a literal backslash in the replacement text. The flags parameter is an optional text string containing zero or more single-letter flags that change the function's behavior. Flag i specifies case-insensitive matching, while flag g specifies replacement of each matching substring rather than only the first one. Supported flags (though not g) are described in Table 9.24.

Some examples:

```
regexp_replace('foobarbaz', 'b..', 'X') → fooXbaz
regexp_replace('foobarbaz', 'b..', 'X', 'g') → fooXX
regexp_replace('foobarbaz', 'b(..)', 'X\1Y', 'g') → fooXarYXazY
```

The regexp_match function returns a text array of captured substring(s) resulting from the first match of a POSIX regular expression pattern to a string. It has the syntax regexp_match(string, pattern [, flags]). If there is no match, the result is NULL. If a match is found, and the pattern contains no parenthesized subexpressions, then the result is a single-element text array containing the substring matching the whole pattern. If a match is found, and the pattern contains parenthesized subexpressions, then the result is a text array whose n'th element is the substring matching the n'th parenthesized subexpression of the pattern (not counting “non-capturing” parentheses; see below for details). The flags parameter is an optional text string containing zero or more single-letter flags that change the function's behavior. Supported flags are described in Table 9.24.

Some examples:

```
regexp_match('foobarbequebaz', 'bar.*que') → {barbeque}
regexp_match('foobarbequebaz', '(bar)(beque)') → {bar,beque}
```

In the common case where you just want the whole matching substring or NULL for no match, write something like

```
#(regexp_match('foobarbequebaz', 'bar.*que'))[1] → barbeque
```

The regexp_matches function returns a set of text arrays of captured substring(s) resulting from matching a POSIX regular expression pattern to a string. It has the same syntax as regexp_match. This function returns no rows if there is no match, one row if there is a match and the g flag is not given, or N rows if there are N matches and the g flag is given. Each returned row is a text array containing the whole matched substring or the substrings matching parenthesized subexpressions of the pattern, just as described above for regexp_match. regexp_matches accepts all the flags shown in Table 9.24, plus the g flag which commands it to return all matches, not just the first one.

Some examples:

```
SELECT * FROM regexp_matches('foo', 'not there') a → [
]
```

```
SELECT * FROM regexp_matches('foobarbequebazilbarfbonk', '(b[^b]+)(b[^b]+)', 'g') a → [
{bar,beque}
```

```
{bazil,barf}
]
```

Tip
In most cases regexp_matches() should be used with the g flag, since if you only want the first match, it's easier and more efficient to use regexp_match(). However, regexp_match() only exists in PostgreSQL version 10 and up. When working in older versions, a common trick is to place a regexp_matches() call in a sub-select, for example:

```
SELECT col1, (SELECT regexp_matches(col2, '(bar)(beque)')) FROM tab;
```

This produces a text array if there's a match, or NULL if not, the same as regexp_match() would do. Without the sub-select, this query would produce no output at all for table rows without a match, which is typically not the desired behavior.

The regexp_split_to_table function splits a string using a POSIX regular expression pattern as a delimiter. It has the syntax regexp_split_to_table(string, pattern [, flags]). If there is no match to the pattern, the function returns the string. If there is at least one match, for each match it returns the text from the end of the last match (or the beginning of the string) to the beginning of the match. When there are no more matches, it returns the text from the end of the last match to the end of the string. The flags parameter is an optional text string containing zero or more single-letter flags that change the function's behavior. regexp_split_to_table supports the flags described in Table 9.24.

The regexp_split_to_array function behaves the same as regexp_split_to_table, except that regexp_split_to_array returns its result as an array of text. It has the syntax regexp_split_to_array(string, pattern [, flags]). The parameters are the same as for regexp_split_to_table.

Some examples:

```
SELECT foo FROM regexp_split_to_table('the quick brown fox jumps over the lazy dog', '\s+') AS foo → [
the
quick
brown
fox
jumps
over
the
lazy
dog
]
```

```
SELECT regexp_split_to_array('the quick brown fox jumps over the lazy dog', '\s+') →
{the,quick,brown,fox,jumps,over,the,lazy,dog}
```

```
SELECT foo FROM regexp_split_to_table('the quick brown fox', '\s*') AS foo → [
t
h
e
q
u
i
c
k
b
r
o
w
n
f
o
x
]
```

As the last example demonstrates, the regexp split functions ignore zero-length matches that occur at the start or end of the string or immediately after a previous match. This is contrary to the strict definition of regexp matching that is implemented by regexp_match and regexp_matches, but is usually the most convenient behavior in practice. Other software systems such as Perl use similar definitions.

9.7.3.5. Regular Expression Matching Rules
In the event that an RE could match more than one substring of a given string, the RE matches the one starting earliest in the string. If the RE could match more than one substring starting at that point, either the longest possible match or the shortest possible match will be taken, depending on whether the RE is greedy or non-greedy.

Whether an RE is greedy or not is determined by the following rules:

Most atoms, and all constraints, have no greediness attribute (because they cannot match variable amounts of text anyway).

Adding parentheses around an RE does not change its greediness.

A quantified atom with a fixed-repetition quantifier ({m} or {m}?) has the same greediness (possibly none) as the atom itself.

A quantified atom with other normal quantifiers (including {m,n} with m equal to n) is greedy (prefers longest match).

A quantified atom with a non-greedy quantifier (including {m,n}? with m equal to n) is non-greedy (prefers shortest match).

A branch — that is, an RE that has no top-level | operator — has the same greediness as the first quantified atom in it that has a greediness attribute.

An RE consisting of two or more branches connected by the | operator is always greedy.

The above rules associate greediness attributes not only with individual quantified atoms, but with branches and entire REs that contain quantified atoms. What that means is that the matching is done in such a way that the branch, or whole RE, matches the longest or shortest possible substring as a whole. Once the length of the entire match is determined, the part of it that matches any particular subexpression is determined on the basis of the greediness attribute of that subexpression, with subexpressions starting earlier in the RE taking priority over ones starting later.

An example of what this means:

```
SUBSTRING('XY1234Z', 'Y*([0-9]{1,3})')  → 123
SUBSTRING('XY1234Z', 'Y*?([0-9]{1,3})') → 1
```

In the first case, the RE as a whole is greedy because `Y*` is greedy. It can match beginning at the Y, and it matches the longest possible string starting there, i.e., Y123. The output is the parenthesized part of that, or 123. In the second case, the RE as a whole is non-greedy because `Y*?` is non-greedy. It can match beginning at the Y, and it matches the shortest possible string starting there, i.e., Y1. The subexpression [0-9]{1,3} is greedy but it cannot change the decision as to the overall match length; so it is forced to match just 1.

In short, when an RE contains both greedy and non-greedy subexpressions, the total match length is either as long as possible or as short as possible, according to the attribute assigned to the whole RE. The attributes assigned to the subexpressions only affect how much of that match they are allowed to “eat” relative to each other.

The quantifiers {1,1} and {1,1}? can be used to force greediness or non-greediness, respectively, on a subexpression or a whole RE. This is useful when you need the whole RE to have a greediness attribute different from what's deduced from its elements. As an example, suppose that we are trying to separate a string containing some digits into the digits and the parts before and after them. We might try to do that like this:

```
regexp_match('abc01234xyz', '(.*)(\d+)(.*)') → {abc0123,4,xyz}
```

That didn't work: the first .* is greedy so it “eats” as much as it can, leaving the \d+ to match at the last possible place, the last digit. We might try to fix that by making it non-greedy:

```
regexp_match('abc01234xyz', '(.*?)(\d+)(.*)') → {abc,0,""}
```

That didn't work either, because now the RE as a whole is non-greedy and so it ends the overall match as soon as possible. We can get what we want by forcing the RE as a whole to be greedy:

```
regexp_match('abc01234xyz', '(?:(.*?)(\d+)(.*)){1,1}') → {abc,01234,xyz}
```

Controlling the RE's overall greediness separately from its components' greediness allows great flexibility in handling variable-length patterns.

When deciding what is a longer or shorter match, match lengths are measured in characters, not collating elements. An empty string is considered longer than no match at all. For example: `bb*` matches the three middle characters of abbbc; (week|wee)(night|knights) matches all ten characters of weeknights; when `(.*).*` is matched against abc the parenthesized subexpression matches all three characters; and when (a*)* is matched against bc both the whole RE and the parenthesized subexpression match an empty string.

If case-independent matching is specified, the effect is much as if all case distinctions had vanished from the alphabet. When an alphabetic that exists in multiple cases appears as an ordinary character outside a bracket expression, it is effectively transformed into a bracket expression containing both cases, e.g., x becomes [xX]. When it appears inside a bracket expression, all case counterparts of it are added to the bracket expression, e.g., [x] becomes [xX] and [^x] becomes [^xX].

If newline-sensitive matching is specified, . and bracket expressions using ^ will never match the newline character (so that matches will not cross lines unless the RE explicitly includes a newline) and ^ and $ will match the empty string after and before a newline respectively, in addition to matching at beginning and end of string respectively. But the ARE escapes \A and \Z continue to match beginning or end of string only. Also, the character class shorthands \D and \W will match a newline regardless of this mode. (Before PostgreSQL 14, they did not match newlines when in newline-sensitive mode. Write [^[:digit:]] or [^[:word:]] to get the old behavior.)

If partial newline-sensitive matching is specified, this affects . and bracket expressions as with newline-sensitive matching, but not ^ and $.

If inverse partial newline-sensitive matching is specified, this affects ^ and $ as with newline-sensitive matching, but not . and bracket expressions. This isn't very useful but is provided for symmetry

9.8. Data Type Formatting Functions

The PostgreSQL formatting functions provide a powerful set of tools for converting various data types (date/time, integer, floating point, numeric) to formatted strings and for converting from formatted strings to specific data types. Table 9.25 lists them. These functions all follow a common calling convention: the first argument is the value to be formatted and the second argument is a template that defines the output or input format.

Table 9.25. Formatting Functions

Function	Description	Example(s)
to_char (timestamp with time zone, text) → text	Converts time stamp to string according to the given format.	`to_char(timestamp '2002-04-20 17:31:12.66', 'HH12:MI:SS') → 05:31:12`
to_char (interval, text) → text	Converts interval to string according to the given format.	`to_char(interval '15h 2m 12s', 'HH24:MI:SS') → 15:02:12`

to_char (numeric_type, text) → text	Converts number to string according to the given format; available for integer, bigint, numeric, real, double precision.	`to_char(125, '999') → ' 125'`
		`to_char(125.8::real, '999D9') → ' 125.8'`
		`to_char(-125.8, '999D99S') → '125.80-'`

to_date (text, text) → date	Converts string to date according to the given format.	`to_date('05 Dec 2000', 'DD Mon YYYY') → '2000-12-05'`

Table 9.30 shows some examples of the use of the to_char function.

Table 9.30. to_char Examples

```
to_char('2000-06-06 05:39:18'::timestamp, 'Day, DD  HH12:MI:SS') → 'Tuesday  , 06  05:39:18'
to_char('2000-06-06 05:39:18'::timestamp, 'FMDay, FMDD  HH12:MI:SS') → 'Tuesday, 6  05:39:18'
to_char(-0.1, '99.99') → '  -.10'
to_char(-0.1, 'FM9.99') → '-.1'
to_char(-0.1, 'FM90.99') → '-0.1'
to_char(0.1, '0.9') → ' 0.1'
to_char(12, '9990999.9') → '    0012.0'
to_char(12, 'FM9990999.9') → '0012.'
to_char(485, '999') → ' 485'
to_char(-485, '999') → '-485'
to_char(485, '9 9 9') → ' 4 8 5'
to_char(1485, '9,999') → ' 1,485'
```

to_number (text, text) → numeric	Converts string to numeric according to the given format.	`to_number('12,454.8-', '99G999D9S') → '-12454.8'`
to_timestamp (text, text) → timestamp with time zone	Converts string to time stamp according to the given format. (See also to_timestamp(double precision) in Table 9.32.)	`cast(to_timestamp('05 Dec 2000', 'DD Mon YYYY') as timestamp) → '2000-12-05 00:00:00'`

```
to_char(485.8,    'Pre: 999  Post: .999' ) → 'Pre: 485 Post: .800'
to_char(12, '99V999') → ' 12000'
to_char(12.4, '99V999') → ' 12400'
to_char(12.45, '99V9') → ' 125'
to_char(0.0004859, '9.99EEEE') → ' 4.86e-04'
```

9.9. Date/Time Functions and Operators

Table 9.32 shows the available functions for date/time value processing, with details appearing in the following subsections. Table 9.31 illustrates the behaviors of the basic arithmetic operators (+, *, etc.). For formatting functions, refer to Section 9.8. You should be familiar with the background information on date/time data types from Section 8.5.

In addition, the usual comparison operators shown in Table 9.1 are available for the date/time types. Dates and timestamps (with or without time zone) are all comparable, while times (with or without time zone) and intervals can only be compared to other values of the same data type. When comparing a timestamp without time zone to a timestamp with time zone, the former value is assumed to be given in the time zone specified by the TimeZone configuration parameter, and is rotated to UTC for comparison to the latter value (which is already in UTC internally). Similarly, a date value is assumed to represent midnight in the TimeZone zone when comparing it to a timestamp.

All the functions and operators described below that take time or timestamp inputs actually come in two variants: one that takes time with time zone or timestamp with time zone, and one that takes time without time zone or timestamp without time zone. For brevity, these variants are not shown separately. Also, the + and * operators come in commutative pairs (for example both date + integer and integer + date); we show only one of each such pair.

Table 9.31. Date/Time Operators

Operator	Description	Example(s)
date + integer → date	Add a number of days to a date	`date '2001-09-28' + 7 → 2001-10-05`

date + interval → timestamp	Add an interval to a date	`date '2001-09-28' + interval '1 hour' → 2001-09-28 01:00:00`
date + time → timestamp	Add a time-of-day to a date	`date '2001-09-28' + time '03:00' → 2001-09-28 03:00:00`
interval + interval → interval	Add intervals	`interval '1 day' + interval '1 hour' → 1 day 01:00:00`
timestamp + interval → timestamp	Add an interval to a timestamp	`timestamp '2001-09-28 01:00' + interval '23 hours' → 2001-09-29 00:00:00`
time + interval → time	Add an interval to a time	`time '01:00' + interval '3 hours' → 04:00:00`

• interval → interval	Negate an interval	`- interval '23 hours' → -23:00:00`
date - date → integer	Subtract dates, producing the number of days elapsed	`date '2001-10-01' - date '2001-09-28' → 3`
date - integer → date	Subtract a number of days from a date	`date '2001-10-01' - 7 → 2001-09-24`
date - interval → timestamp	Subtract an interval from a date	`date '2001-09-28' - interval '1 hour' → 2001-09-27 23:00:00`
time - time → interval	Subtract times	`time '05:00' - time '03:00' → 02:00:00`

time - interval → time	Subtract an interval from a time	

Function	Description	Example(s)
age (timestamp, timestamp) → interval	Subtract arguments, producing a "symbolic" result that uses years and months, rather than just days	`age(timestamp '2001-04-10', timestamp '1957-06-13') → 43 years 9 mons 27 days`
age (timestamp) → interval	Subtract argument from current_date (at midnight)	`age(timestamp '1957-06-13') ~→ 62 years 6 mons 10 days`
clock_timestamp ( ) → timestamp with time zone	Current date and time (changes during statement execution); see Section 9.9.5	`clock_timestamp() ~→ 2019-12-23 14:39:53.662522-05`
current_date → date	Current date; see Section 9.9.5	`current_date ~→ 2019-12-23`

```
interval '2 hours' →
```

```
1-09-28 23:00' - interval
001-09-28 00:00:00
```

```
y' - interval '1 hour' → 1
```

```
1-09-29 03:00' - timestamp
:00' → 63 days 15:00:00
```

interval * double precision → interval	Multiply an interval by a scalar	

```
interval '1 second' * 900 → 00:15:00
interval '1 day' * 21 → 21 days
interval '1 hour' * 3.5 → 03:30:00
```

current_time → time with time zone	Current time of day; see Section 9.9.5	`current_time ~→ 14:39:53.662522-05`
current_time (integer) → time with time zone	Current time of day, with limited precision; see Section 9.9.5	`current_time(2) ~→ 14:39:53.66-05`
current_timestamp → timestamp with time zone	Current date and time (start of current transaction); see Section 9.9.5	`current_timestamp ~→ 2019-12-23 14:39:53.662522-05`
current_timestamp (integer) → timestamp with time zone	Current date and time (start of current transaction), with limited precision; see Section 9.9.5	`current_timestamp(0) ~→ 2019-12-23 14:39:53-05`
date_bin (interval, timestamp, timestamp) → timestamp	Bin input into specified interval aligned with specified origin; see Section 9.9.3	`date_bin('15 minutes', timestamp '2001-02-16 20:38:40', timestamp '2001-02-16 20:05:00') → 2001-02-16 20:35:00`

`ur' / 1.5 → 00:40:00`

date_part (text, timestamp) → double precision	Get timestamp subfield (equivalent to extract); see Section 9.9.1	`date_part('hour', timestamp '2001-02-16 20:38:40') → 20`
date_part (text, interval) → double precision	Get interval subfield (equivalent to extract); see Section 9.9.1	`date_part('month', interval '2 years 3 months') → 3`
date_trunc (text, timestamp) → timestamp	Truncate to specified precision; see Section 9.9.2	`date_trunc('hour', timestamp '2001-02-16 20:38:40') → 2001-02-16 20:00:00`
date_trunc (text, timestamp with time zone, text) → timestamp with time zone	Truncate to specified precision in the specified time zone; see Section 9.9.2 (NOT SUPPORTED)	`#date_trunc('day', timestamptz '2001-02-16 20:38:40+00', 'Australia/Sydney') → 2001-02-16 13:00:00+00`

date_trunc (text, interval) → interval	Truncate to specified precision; see Section 9.9.2	`date_trunc('hour', interval '2 days 3 hours 40 minutes') → 2 days 03:00:00`
extract (field from timestamp) → numeric	Get timestamp subfield; see Section 9.9.1	`extract(hour from timestamp '2001-02-16 20:38:40') → 20`
extract (field from interval) → numeric	Get interval subfield; see Section 9.9.1	`extract(month from interval '2 years 3 months') → 3`
isfinite (date) → boolean	Test for finite date (not +/-infinity)	`isfinite(date '2001-02-16') → true`
isfinite (timestamp) → boolean	Test for finite timestamp (not +/-infinity)	`isfinite(timestamp 'infinity') → false`

In addition to these functions, the SQL OVERLAPS operator is supported: (NOT SUPPORTED)

```
(start1, end1) OVERLAPS (start2, end2)
(start1, length1) OVERLAPS (start2, length2)
```

This expression yields true when two time periods (defined by their endpoints) overlap, false when they do not overlap. The endpoints can be specified as pairs of dates, times, or time stamps; or as a date, time, or time stamp followed by an interval. When a pair of values is provided, either the start or the end can be written first; OVERLAPS automatically takes the earlier value of the pair as the start. Each time period is considered to represent the half-open interval start <= time < end, unless start and end are equal in which case it represents that single time instant. This means for instance that two time periods with only an endpoint in common do not overlap.

```
(DATE '2001-02-16', DATE '2001-12-21') OVERLAPS (DATE '2001-10-30', DATE '2002-10-30') → true
(DATE '2001-02-16', INTERVAL '100 days') OVERLAPS (DATE '2001-10-30', DATE '2002-10-30') → false
(DATE '2001-10-29', DATE '2001-10-30') OVERLAPS (DATE '2001-10-30', DATE '2001-10-31') → false
(DATE '2001-10-30', DATE '2001-10-30') OVERLAPS (DATE '2001-10-30', DATE '2001-10-31') → true
```

isfinite (interval) → boolean	Test for finite interval (currently always true)	`isfinite(interval '4 hours') → true`
justify_days (interval) → interval	Adjust interval so 30-day time periods are represented as months	`justify_days(interval '35 days') → 1 mon 5 days`
justify_hours (interval) → interval	Adjust interval so 24-hour time periods are represented as days	`justify_hours(interval '27 hours') → 1 day 03:00:00`
justify_interval (interval) → interval	Adjust interval using justify_days and justify_hours, with additional sign adjustments	`justify_interval(interval '1 mon -1 hour') → 29 days 23:00:00`
localtime → time	Current time of day; see Section 9.9.5 (NOT SUPPORTED)	`#localtime ~→ 14:39:53.662522`

np with time zone value, the days component advances or
lays, keeping the time of day the same. Across daylight saving
ST), this means interval '1 day' does not necessarily equal
':

```
' ~→ 2005-04-03 12:00:00-06
urs' ~→ 2005-04-03 13:00:00-06
```

ne at 2005-04-03 02:00:00 in time zone America/Denver.

months have different numbers of days. PostgreSQL's approach
For example, age('2004-06-01', '2004-04-30') uses April to yield 1
while April has only 30.

le way to perform subtraction is to convert each value to a
this produces the number of seconds between the two values.
light saving time adjustments. Subtraction of date or timestamp
tes/seconds between the values, making the same adjustments.
ming field-by-field subtraction and then adjusting for negative field
ample results were produced with timezone = 'US/Eastern'; there

```
CH FROM timestamptz '2013-03-01 12:00:00') ~→

OCH FROM timestamptz '2013-03-01 12:00:00')) / 60 /

0' ~→ 121 days 23:00:00
0:00') → 4 mons
```

s. source must be a value expression of type timestamp, time, or
d as well.) field is an identifier or string that selects what field to
The following are valid field names:

```
EXTRACT(CENTURY FROM TIMESTAMP '2000-12-16 12:21:13') → 20
EXTRACT(CENTURY FROM TIMESTAMP '2001-02-16 20:38:40') → 21
```

The first century starts at 0001-01-01 00:00:00 AD, although they did not know it at the time. This definition applies to all Gregorian calendar countries. There is no century number 0, you go from -1 century to 1 century. If you disagree with this, please write your complaint to: Pope, Cathedral Saint-Peter of Roma, Vatican.

day
For timestamp values, the day (of the month) field (1–31) ; for interval values, the number of days

```
EXTRACT(DAY FROM TIMESTAMP '2001-02-16 20:38:40') → 16
 EXTRACT(DAY FROM INTERVAL '40 days 1 minute') → 40
```

decade
The year field divided by 10

```
EXTRACT(DECADE FROM TIMESTAMP '2001-02-16 20:38:40') → 200
```

dow
The day of the week as Sunday (0) to Saturday (6)

```
EXTRACT(DOW FROM TIMESTAMP '2001-02-16 20:38:40') → 5
```

Note that extract's day of the week numbering differs from that of the to_char(..., 'D') function.

doy
The day of the year (1–365/366)

```
EXTRACT(DOY FROM TIMESTAMP '2001-02-16 20:38:40') → 47
```

localtime (integer) → time	Current time of day, with limited precision; see Section 9.9.5 (NOT SUPPORTED)	`#localtime(0) ~→ 14:39:53`
localtimestamp → timestamp	Current date and time (start of current transaction); see Section 9.9.5 (NOT SUPPORTED)	`#localtimestamp ~→ 2019-12-23 14:39:53.662522`
localtimestamp (integer) → timestamp	Current date and time (start of current transaction), with limited precision; see Section 9.9.5 (NOT SUPPORTED)	`#localtimestamp(2) ~→ 2019-12-23 14:39:53.66`
make_date (year int, month int, day int) → date	Create date from year, month and day fields (negative years signify BC)	`make_date(2013, 7, 15) → 2013-07-15`
make_interval ([years int [, months int [, weeks int [, days int [, hours int [, mins int [, secs double precision]]]]]]) → interval	Create interval from years, months, weeks, days, hours, minutes and seconds fields, each of which can default to zero (NOT SUPPORTED)	`#make_interval(days => 10) → 10 days`

:00:00 UTC (negative for timestamps before that); for date and
hout regard to timezone or daylight-savings rules; for interval

```
-08') → 982384720.120000
120000
```

tamp:

```
```

p value could produce a misleading result: the result will
be the case.

```
```

week numbering.

ls)

```
```

ing the 4th of January, so in early January or late December the
formation.

julian
The Julian Date corresponding to the date or timestamp (not applicable to intervals). Timestamps that are not local midnight result in a fractional value. See Section B.7 for more information.

```
EXTRACT(JULIAN FROM DATE '2006-01-01') → 2453737
EXTRACT(JULIAN FROM TIMESTAMP '2006-01-01 12:00') → 2453737.50000000000000000000
```

microseconds
The seconds field, including fractional parts, multiplied by 1 000 000; note that this includes full seconds

```
EXTRACT(MICROSECONDS FROM TIME '17:12:28.5') → 28500000
```

millennium
The millennium

```
EXTRACT(MILLENNIUM FROM TIMESTAMP '2001-02-16 20:38:40') → 3
```

Years in the 1900s are in the second millennium. The third millennium started January 1, 2001.

milliseconds
The seconds field, including fractional parts, multiplied by 1000. Note that this includes full seconds.

```
EXTRACT(MILLISECONDS FROM TIME '17:12:28.5') → 28500.000
```

minute
The minutes field (0–59)

```
EXTRACT(MINUTE FROM TIMESTAMP '2001-02-16 20:38:40') → 38
```

make_time (hour int, min int, sec double precision) → time	Create time from hour, minute and seconds fields	`make_time(8, 15, 23.5) → 08:15:23.5`
make_timestamp (year int, month int, day int, hour int, min int, sec double precision) → timestamp	Create timestamp from year, month, day, hour, minute and seconds fields (negative years signify BC)	`make_timestamp(2013, 7, 15, 8, 15, 23.5) → 2013-07-15 08:15:23.5`
make_timestamptz (year int, month int, day int, hour int, min int, sec double precision [, timezone text]) → timestamp with time zone	Create timestamp with time zone from year, month, day, hour, minute and seconds fields (negative years signify BC). If timezone is not specified, the current time zone is used; the examples assume the session time zone is Europe/London	`make_timestamptz(2013, 7, 15, 8, 15, 23.5) ~→ 2013-07-15 08:15:23.5+01` `#make_timestamptz(2013, 7, 15, 8, 15, 23.5, 'America/New_York') ~→ 2013-07-15 13:15:23.5+01`
now ( ) → timestamp with time zone	Current date and time (start of current transaction); see Section 9.9.5	`now() ~→ 2019-12-23 14:39:53.662522-05`

l values, the number of months, modulo 12 (0–11)

0000

to time zones east of UTC, negative values to zones west of handled.)

weeks start on Mondays and the first week of a year contains hat year.

t of the 52nd or 53rd week of the previous year, and for late- 1-01 is part of the 53rd week of year 2004, and 2006-01-01 is part It's recommended to use the isoyear field together with week to

The year field. Keep in mind there is no 0 AD, so subtracting BC years from AD years should be done with care.

```
EXTRACT(YEAR FROM TIMESTAMP '2001-02-16 20:38:40') → 2001
```

Note

When the input value is +/-Infinity, extract returns +/-Infinity for monotonically-increasing fields (epoch, julian, year, isoyear, decade, century, and millennium). For other fields, NULL is returned. PostgreSQL versions before 9.6 returned zero for all cases of infinite input.

The extract function is primarily intended for computational processing. For formatting date/time values for display, see Section 9.8.

The date_part function is modeled on the traditional Ingres equivalent to the SQL-standard function extract:

```
date_part('field', source)
```

Note that here the field parameter needs to be a string value, not a name. The valid field names for date_part are the same as for extract. For historical reasons, the date_part function returns values of type double precision. This can result in a loss of precision in certain uses. Using extract is recommended instead.

```
date_part('day', TIMESTAMP '2001-02-16 20:38:40') → 16
date_part('hour', INTERVAL '4 hours 3 minutes') → 4
```

9.9.2. date_trunc

The function date_trunc is conceptually similar to the trunc function for numbers.

```
date_trunc(field, source [, time_zone ])
```

source is a value expression of type timestamp, timestamp with time zone, or interval. (Values of type date and time are cast automatically to timestamp or interval, respectively.) field selects to which precision to truncate the input value. The return value is likewise of type timestamp, timestamp with time zone, or interval, and it has all fields that are less significant than the selected one set to zero (or one, for day and month).

statement_timestamp ( ) → timestamp with time zone	Current date and time (start of current statement); see Section 9.9.5	`statement_timestamp() ~→ 2019-12-23 14:39:53.662522-05`
timeofday ( ) → text	Current date and time (like clock_timestamp, but as a text string); see Section 9.9.5	`timeofday() ~→ Mon Dec 23 14:39:53.662522 2019 EST`
transaction_timestamp ( ) → timestamp with time zone	Current date and time (start of current transaction); see Section 9.9.5	`transaction_timestamp() ~→ 2019-12-23 14:39:53.662522-05`
to_timestamp (double precision) → timestamp with time zone	Convert Unix epoch (seconds since 1970-01-01 00:00:00+00) to timestamp with time zone	`to_timestamp(1284352323) → 2010-09-13 04:32:03+00`

ned with respect to a particular time zone; for example, truncation
e with respect to the current TimeZone setting, but the optional
ne name can be specified in any of the ways described in Section

interval inputs. These are always taken at face value.

```
0:00:00
0:00:00
 ) ~→ 2001-02-16 00:00:00-05
', 'Australia/Sydney') ~→ 2001-02-16 08:00:00-05
```

ide) aligned with a specified origin.

```
date_bin(stride, source, origin)
```

source is a value expression of type timestamp or timestamp with time zone. (Values of type date are cast automatically to timestamp.) stride is a value expression of type interval. The return value is likewise of type timestamp or timestamp with time zone, and it marks the beginning of the bin into which the source is placed.

Examples:

```
date_bin('15 minutes', TIMESTAMP '2020-02-11 15:44:17', TIMESTAMP '2001-01-01') → 2020-02-11 15:30:00
date_bin('15 minutes', TIMESTAMP '2020-02-11 15:44:17', TIMESTAMP '2001-01-01 00:02:30') → 2020-02-11 15:32:30
```

In the case of full units (1 minute, 1 hour, etc.), it gives the same result as the analogous date_trunc call, but the difference is that date_bin can truncate to an arbitrary interval.

The stride interval must be greater than zero and cannot contain units of month or larger.

9.9.4. AT TIME ZONE

The AT TIME ZONE operator converts time stamp without time zone to/from time stamp with time zone, and time with time zone values to different time zones. Table 9.33 shows its variants. (NOT SUPPORTED)

Table 9.33. AT TIME ZONE Variants

Operator	Description	Example(s)	
Converts given time stamp without time zone to time stamp with time zone, assuming the given value is in the named time zone.			
Converts given time stamp with time zone to time stamp without time zone, as the time would appear in that zone.			
Converts given time with time zone to a new time zone. Since no date is supplied, this uses the currently active UTC offset for the named destination zone.			

In these expressions, the desired time zone zone can be specified either as a text value (e.g., 'America/Los_Angeles') or as an interval (e.g., INTERVAL '-08:00'). In the text case, a time zone name can be specified in any of the ways described in Section 8.5.3. The interval case is only useful for zones that have fixed offsets from UTC, so it is not very common in practice.

Examples (assuming the current TimeZone setting is America/Los_Angeles):

```
#TIMESTAMP '2001-02-16 20:38:40' AT TIME ZONE 'America/Denver' ~→ 2001-02-16 19:38:40-08
#TIMESTAMP WITH TIME ZONE '2001-02-16 20:38:40-05' AT TIME ZONE 'America/Denver' ~→ 2001-02-16 18:38:40
#TIMESTAMP '2001-02-16 20:38:40' AT TIME ZONE 'Asia/Tokyo' AT TIME ZONE 'America/Chicago' ~→ 2001-02-16 05:38:40
```

The first example adds a time zone to a value that lacks it, and displays the value using the current TimeZone setting. The second example shifts the time stamp with time zone value to the specified time zone, and returns the value without a time zone. This allows storage and display of values different from the current TimeZone setting. The third example converts Tokyo time to Chicago time.

The function timezone(zone, timestamp) is equivalent to the SQL-conforming construct timestamp AT TIME ZONE zone.

9.9.5. Current Date/Time

PostgreSQL provides a number of functions that return values related to the current date and time. These SQL-standard functions all return values based on the start time of the current transaction:

```
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_TIME(precision)
CURRENT_TIMESTAMP(precision)
LOCALTIME
LOCALTIMESTAMP
LOCALTIME(precision)
LOCALTIMESTAMP(precision)
```

CURRENT_TIME and CURRENT_TIMESTAMP deliver values with time zone; LOCALTIME and LOCALTIMESTAMP deliver values without time zone.

CURRENT_TIME, CURRENT_TIMESTAMP, LOCALTIME, and LOCALTIMESTAMP can optionally take a precision parameter, which causes the result to be rounded to that many fractional digits in the seconds field. Without a precision parameter, the result is given to the full available precision.

Some examples:

```
CURRENT_TIME ~→ 14:39:53.662522-05
CURRENT_DATE ~→ 2019-12-23
CURRENT_TIMESTAMP ~→ 2019-12-23 14:39:53.662522-05
CURRENT_TIMESTAMP(2) ~→ 2019-12-23 14:39:53.66-05
#LOCALTIMESTAMP ~→ 2019-12-23 14:39:53.662522
```

Since these functions return the start time of the current transaction, their values do not change during the transaction. This is considered a feature: the intent is to allow a single transaction to have a consistent notion of the “current” time, so that multiple modifications within the same transaction bear the same time stamp.

Note
Other database systems might advance these values more frequently.

PostgreSQL also provides functions that return the start time of the current statement, as well as the actual current time at the instant the function is called. The complete list of non-SQL-standard time functions is:

```
transaction_timestamp()
statement_timestamp()
clock_timestamp()
timeofday()
now()
```

transaction_timestamp() is equivalent to CURRENT_TIMESTAMP, but is named to clearly reflect what it returns. statement_timestamp() returns the start time of the current statement (more specifically, the time of receipt of the latest command message from the client). statement_timestamp() and transaction_timestamp() return the same value during the first command of a transaction, but might differ during subsequent commands. clock_timestamp() returns the actual current time, and therefore its value changes even within a single SQL command. timeofday() is a historical PostgreSQL function. Like clock_timestamp(), it returns the actual current time, but as a formatted text string rather than a timestamp with time zone value. now() is a traditional PostgreSQL equivalent to transaction_timestamp().

All the date/time data types also accept the special literal value now to specify the current date and time (again, interpreted as the transaction start time). Thus, the following three all return the same result:

```
SELECT CURRENT_TIMESTAMP;
SELECT now();
SELECT TIMESTAMP 'now';  /* but see tip below */
```

Tip
Do not use the third form when specifying a value to be evaluated later, for example in a DEFAULT clause for a table column. The system will convert now to a timestamp as soon as the constant is parsed, so that when the default value is needed, the time of the table creation would be used! The first two forms will not be evaluated until the default value is used, because they are function calls. Thus they will give the desired behavior of defaulting to the time of row insertion. (See also Section 8.5.1.4.)

9.9.6. Delaying Execution

The following functions are available to delay execution of the server process:

```
pg_sleep ( double precision )
pg_sleep_for ( interval )
pg_sleep_until ( timestamp with time zone )
```

pg_sleep makes the current session's process sleep until the given number of seconds have elapsed. Fractional-second delays can be specified. pg_sleep_for is a convenience function to allow the sleep time to be specified as an interval. pg_sleep_until is a convenience function for when a specific wake-up time is desired. For example:

```
SELECT pg_sleep(1.5);
SELECT pg_sleep_for('5 minutes');
SELECT pg_sleep_until('tomorrow 03:00');
```

Note
The effective resolution of the sleep interval is platform-specific; 0.01 seconds is a common value. The sleep delay will be at least as long as specified. It might be longer depending on factors such as server load. In particular, pg_sleep_until is not guaranteed to wake up exactly at the specified time, but it will not wake up any earlier.

Warning
Make sure that your session does not hold more locks than necessary when calling pg_sleep or its variants. Otherwise other sessions might have to wait for your sleeping process, slowing down the entire system.

9.10. Enum Support Functions (NOT SUPPORTED)

9.11. Geometric Functions and Operators

The geometric types point, box, lseg, line, path, polygon, and circle have a large set of native support functions and operators, shown in Table 9.35, Table 9.36, and Table 9.37.

Table 9.35. Geometric Operators

Operator	Description	Example(s)
geometric_type + point → geometric_type	Adds the coordinates of the second point to those of each point of the first argument, thus performing translation. Available for point, box, path, circle.	`box '(1,1),(0,0)' + point '(2,0)' → (3,1),(2,0)`

path + path → path	Concatenates two open paths (returns NULL if either path is closed).	`path '[(0,0),(1,1)]' + path '[(2,2),(3,3),(4,4)]' → [(0,0),(1,1),(2,2),(3,3),(4,4)]`
geometric_type - point → geometric_type	Subtracts the coordinates of the second point from those of each point of the first argument, thus performing translation. Available for point, box, path, circle.	`box '(1,1),(0,0)' - point '(2,0)' → (-1,1),(-2,0)`
geometric_type * point → geometric_type	Multiplies each point of the first argument by the second point (treating a point as being a complex number represented by real and imaginary parts, and performing standard complex multiplication). If one interprets the second point as a vector, this is equivalent to scaling the object's size and distance from the origin by the length of the vector, and rotating it counterclockwise around the origin by the vector's angle from the x axis. Available for point, box,[a] path, circle.	`path '((0,0),(1,0),(1,1))' * point '(3.0,0)' → ((0,0),(3,0),(3,3))` `path '((0,0),(1,0),(1,1))' * point(cosd(45), sind(45)) → ((0,0),(0.7071067811865475,0.7071067811865475),(0,1.414213562373095))`
geometric_type / point → geometric_type	Divides each point of the first argument by the second point (treating a point as being a complex number represented by real and imaginary parts, and performing standard complex division). If one interprets the second point as a vector, this is equivalent to scaling the object's size and distance from the origin down by the length of the vector, and rotating it clockwise around the origin by the vector's angle from the x axis. Available for point, box,[a] path, circle.	`path '((0,0),(1,0),(1,1))' / point '(2.0,0)' → ((0,0),(0.5,0),(0.5,0.5))` `path '((0,0),(1,0),(1,1))' / point(cosd(45), sind(45)) → ((0,0),(0.7071067811865476,-0.7071067811865476),(1.4142135623730951,0))`

@-@ geometric_type → double precision	Computes the total length. Available for lseg, path.	`@-@ path '[(0,0),(1,0),(1,1)]' → 2`
@@ geometric_type → point	Computes the center point. Available for box, lseg, polygon, circle.	`@@ box '(2,2),(0,0)' → (1,1)`
# geometric_type → integer	Returns the number of points. Available for path, polygon.	`# path '((1,0),(0,1),(-1,0))' → 3`
geometric_type # geometric_type → point	Computes the point of intersection, or NULL if there is none. Available for lseg, line.	`lseg '[(0,0),(1,1)]' # lseg '[(1,0),(0,1)]' → (0.5,0.5)`
box # box → box	Computes the intersection of two boxes, or NULL if there is none.	`box '(2,2),(-1,-1)' # box '(1,1),(-2,-2)' → (1,1),(-1,-1)`

geometric_type ## geometric_type → point	Computes the closest point to the first object on the second object. Available for these pairs of types: (point, box), (point, lseg), (point, line), (lseg, box), (lseg, lseg), (line, lseg).	`point '(0,0)' ## lseg '[(2,0),(0,2)]' → (1,1)`
geometric_type <-> geometric_type → double precision	Computes the distance between the objects. Available for all geometric types except polygon, for all combinations of point with another geometric type, and for these additional pairs of types: (box, lseg), (lseg, line), (polygon, circle) (and the commutator cases).	`circle '<(0,0),1>' <-> circle '<(5,0),1>' → 3`
geometric_type @> geometric_type → boolean	Does first object contain second? Available for these pairs of types: (box, point), (box, box), (path, point), (polygon, point), (polygon, polygon), (circle, point), (circle, circle).	`circle '<(0,0),2>' @> point '(1,1)' → true`
geometric_type <@ geometric_type → boolean	Is first object contained in or on second? Available for these pairs of types: (point, box), (point, lseg), (point, line), (point, path), (point, polygon), (point, circle), (box, box), (lseg, box), (lseg, line), (polygon, polygon), (circle, circle).	`point '(1,1)' <@ circle '<(0,0),2>' → true`
geometric_type && geometric_type → boolean	Do these objects overlap? (One point in common makes this true.) Available for box, polygon, circle.	`box '(1,1),(0,0)' && box '(2,2),(0,0)' → true`

geometric_type << geometric_type → boolean	Is first object strictly left of second? Available for point, box, polygon, circle.	`circle '<(0,0),1>' << circle '<(5,0),1>' → true`
geometric_type >> geometric_type → boolean	Is first object strictly right of second? Available for point, box, polygon, circle.	`circle '<(5,0),1>' >> circle '<(0,0),1>' → true`
geometric_type &< geometric_type → boolean	Does first object not extend to the right of second? Available for box, polygon, circle.	`box '(1,1),(0,0)' &< box '(2,2),(0,0)' → true`
geometric_type &>	Does first object not extend to the left of second? Available for box, polygon, circle.	

x is still considered to have sides parallel to the axes. Hence the box's

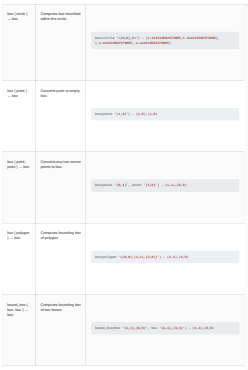
for the point, box, polygon, and circle types. Some of the geometric
er scalar comparison operators (<= and so on), where available for

s point <<| point and point |>> point were respectively called <^ and >^.
moved.

Function	Description	Example(s)
area (geometric_type) → double precision	Computes area. Available for box, path, circle. A path input must be closed, else NULL is returned. Also, if the path is self-intersecting, the result may be meaningless.	`area(box '(2,2),(0,0)') → 4`
center (geometric_type) → point	Computes center point. Available for box, circle.	`center(box '(1,2),(0,0)') → (0.5,1)`
diagonal (box) → lseg	Extracts box's diagonal as a line segment (same as lseg(box)).	`diagonal(box '(1,2),(0,0)') → [(1,2),(0,0)]`

diameter (circle) → double precision	Computes diameter of circle.	`diameter(circle '<(0,0),2>') → 4`
height (box) → double precision	Computes vertical size of box.	`height(box '(1,2),(0,0)') → 2`
isclosed (path) → boolean	Is path closed?	`isclosed(path '((0,0),(1,1),(2,0))') → true`
isopen (path) → boolean	Is path open?	`isopen(path '[(0,0),(1,1),(2,0)]') → true`
length (geometric_type) → double precision	Computes the total length. Available for lseg, path.	`length(path '((-1,0),(1,0))') → 4`

npoints (geometric_type) → integer	Returns the number of points. Available for path, polygon.	`npoints(path '[(0,0),(1,1),(2,0)]') → 3`
pclose (path) → path	Converts path to closed form.	`pclose(path '[(0,0),(1,1),(2,0)]') → ((0,0),(1,1),(2,0))`
popen (path) → path	Converts path to open form.	`popen(path '((0,0),(1,1),(2,0))') → [(0,0),(1,1),(2,0)]`
radius (circle) → double precision	Computes radius of circle.	`radius(circle '<(0,0),2>') → 2`

	Computes slope of a line drawn through the two points.	`slope(point '(0,0)', point '(2,1)') → 0.5`
width (box) → double precision	Computes horizontal size of box.	`width(box '(1,2),(0,0)') → 1`

9.12. Network Address Functions and Operators

The IP network address types, cidr and inet, support the usual comparison operators shown in Table 9.1 as well as the specialized operators and functions shown in Table 9.38 and Table 9.39.

Any cidr value can be cast to inet implicitly; therefore, the operators and functions shown below as operating on inet also work on cidr values. (Where there are separate functions for inet and cidr, it is because the behavior should be different for the two cases.) Also, it is permitted to cast an inet value to cidr. When this is done, any bits to the right of the netmask are silently zeroed to create a valid cidr value.

Table 9.38. IP Address Operators

Operator	Description	Example(s)

inet << inet → boolean	Is subnet strictly contained by subnet? This operator, and the next four, test for subnet inclusion. They consider only the network parts of the two addresses (ignoring any bits to the right of the netmasks) and determine whether one network is identical to or a subnet of the other.	`inet '192.168.1.5' << inet '192.168.1/24' → true`
		`inet '192.168.0.5' << inet '192.168.1/24' → false`
		`inet '192.168.1/24' << inet '192.168.1/24' → false`

inet <<= inet → boolean	Is subnet contained by or equal to subnet?	`inet '192.168.1/24' <<= inet '192.168.1/24' → true`
inet >> inet → boolean	Does subnet strictly contain subnet?	`inet '192.168.1/24' >> inet '192.168.1.5' → true`
inet >>= inet → boolean	Does subnet contain or equal subnet?	`inet '192.168.1/24' >>= inet '192.168.1/24' → true`
inet && inet → boolean	Does either subnet contain or equal the other?	`inet '192.168.1/24' && inet '192.168.1.80/28' → true` `inet '192.168.1/24' && inet '192.168.2.0/28' → false`

~ inet → inet	Computes bitwise NOT.	`~ inet '192.168.1.6' → 63.87.254.249`
inet & inet → inet	Computes bitwise AND.	`inet '192.168.1.6' & inet '0.0.0.255' → 0.0.0.6`
inet | inet → inet	Computes bitwise OR.	`inet '192.168.1.6' | inet '0.0.0.255' → 192.168.1.255`
inet + bigint → inet	Adds an offset to an address.	`inet '192.168.1.6' + 25 → 192.168.1.31`
bigint + inet → inet	Adds an offset to an address. (NOT SUPPORTED)	`#200 + inet '::ffff:fff0:1' → ::ffff:255.240.0.201`

inet - bigint → inet	Subtracts an offset from an address.

Function	Description	Example(s)
abbrev (inet) → text	Creates an abbreviated display format as text. (The result is the same as the inet output function produces; it is “abbreviated” only in comparison to the result of an explicit cast to text, which for historical reasons will never suppress the netmask part.)	`abbrev(inet '10.1.0.0/32') → 10.1.0.0`
abbrev (cidr) → text	Creates an abbreviated display format as text. (The abbreviation consists of dropping all-zero octets to the right of the netmask; more examples are in Table 8.22.)	`abbrev(cidr '10.1.0.0/16') → 10.1/16`
broadcast (inet) → inet	Computes the broadcast address for the address's network.	`broadcast(inet '192.168.1.5/24') → 192.168.1.255/24`
family (inet) → integer	Returns the address's family: 4 for IPv4, 6 for IPv6.	`family(inet '::1') → 6`

`'192.168.1.43' - 36 → 168.1.7`

`'192.168.1.43' - inet '168.1.19' → 24`

`'::1' - inet '::ffff:1' 294901760`

host (inet) → text	Returns the IP address as text, ignoring the netmask.	`host(inet '192.168.1.0/24') → 192.168.1.0`
hostmask (inet) → inet	Computes the host mask for the address's network.	`hostmask(inet '192.168.23.20/30') → 0.0.0.3`
inet_merge (inet, inet) → cidr	Computes the smallest network that includes both of the given networks.	`inet_merge(inet '192.168.1.5/24', inet '192.168.2.5/24') → 192.168.0.0/22`
inet_same_family (inet, inet) → boolean	Tests whether the addresses belong to the same IP family.	`inet_same_family(inet '192.168.1.5/24', inet '::1') → false`

masklen (inet) → integer	Returns the netmask length in bits.	`masklen(inet '192.168.1.5/24') → 24`
netmask (inet) → inet	Computes the network mask for the address's network.	`netmask(inet '192.168.1.5/24') → 255.255.255.0`

s well as the specialized functions
s shown above for IP addresses.

Function	Description	Example(s)
trunc (macaddr) → macaddr	Sets the last 3 bytes of the address to zero. The remaining prefix can be associated with a particular manufacturer (using data not included in PostgreSQL).	`trunc(macaddr '12:34:56:78:90:ab') → 12:34:56:00:00:00`
trunc (macaddr8) → macaddr8	Sets the last 5 bytes of the address to zero. The remaining prefix can be associated with a particular manufacturer (using data not included in PostgreSQL).	`trunc(macaddr8 '12:34:56:78:90:ab:cd:ef') → 12:34:56:00:00:00:00:00`
macaddr8_set7bit (macaddr8) → macaddr8	Sets the 7th bit of the address to one, creating what is known as modified EUI-64, for inclusion in an IPv6 address.	`macaddr8_set7bit(macaddr8 '00:34:56:ab:cd:ef') → 02:34:56:ff:fe:ab:cd:ef`

9.13. Text Search Functions and Operators (NOT SUPPORTED)

9.14. UUID Functions

PostgreSQL includes one function to generate a UUID:

gen_random_uuid () → uuid

This function returns a version 4 (random) UUID. This is the most commonly used type of UUID and is appropriate for most applications.

9.15. XML Functions

The functions and function-like expressions described in this section operate on values of type xml. See Section 8.13 for information about the xml type. The function-like expressions xmlparse and xmlserialize for converting to and from type xml are documented there, not in this section.

set_masklen (cidr, integer) → cidr	Sets the netmask length for a cidr value. Address bits to the right of the new netmask are set to zero.	`set_masklen(cidr '192.168.1.0/24', 16) → 192.168.0.0/16`
text (inet) → text	Returns the unabbreviated IP address and netmask length as text. (This has the same result as an explicit cast to text.)	`text(inet '192.168.1.5') → 192.168.1.5/32`

they are particularly suitable for

The text cannot contain “--” or end
is null.

content fragment. Null values are

Example:

```
#xmlconcat('<abc/>', '<bar>foo</bar>') → <abc/><bar>foo</bar>
```

XML declarations, if present, are combined as follows. If all argument values have the same XML version declaration, that version is used in the result, else no version is used. If all argument values have the standalone declaration value “yes”, then that value is used in the result. If all argument values have a standalone declaration value and at least one is “no”, then that is used in the result. Else the result will have no standalone declaration. If the result is determined to require a standalone declaration but no version declaration, a version declaration with version 1.0 will be used because XML requires an XML declaration to contain a version declaration. Encoding declarations are ignored and removed in all cases.

Example:

```
#xmlconcat('<?xml version="1.1"?><foo/>', '<?xml version="1.1" standalone="no"?><bar/>') → <?xml version="1.1"?><foo/>
<bar/>
```

9.15.1.3. Xmlelement

xmlelement (NAME name [, XMLATTRIBUTES (attvalue [AS attname] [, ...])] [, content [, ...]]) → xml (NOT SUPPORTED)

```
#xmlelement(name foo) → <foo/>
#xmlelement(name foo, xmlattributes('xyz' as bar)) → <foo bar="xyz"/>
#xmlelement(name foo, xmlattributes(current_date as bar), 'cont', 'ent') ~→ <foo bar="2007-01-26">content</foo>
```

Element and attribute names that are not valid XML names are escaped by replacing the offending characters by the sequence *xHHHH*, where HHHH is the character's Unicode codepoint in hexadecimal notation. For example:

```
#xmlelement(name "foo$bar", xmlattributes('xyz' as "a&b")) → <foo_x0024_bar a_x0026_b="xyz"/>
```

An explicit attribute name need not be specified if the attribute value is a column reference, in which case the column's name will be used as the attribute name by default. In other cases, the attribute must be given an explicit name. So this example is valid:

Element content, if specified, will be formatted according to its data type. If the content is itself of type xml, complex XML documents can be constructed. For example:

```
#xmlelement(name foo, xmlattributes('xyz' as bar), xmlelement(name abc), xmlcomment('test'), xmlelement(name xyz)) →
<foo bar="xyz"><abc/><!--test--><xyz/></foo>
```

Content of other types will be formatted into valid XML character data. This means in particular that the characters <, >, and & will be converted to entities. Binary data (data type bytea) will be represented in base64 or hex encoding, depending on the setting of the configuration parameter xmlbinary. The particular behavior for individual data types is expected to evolve in order to align the PostgreSQL mappings with those specified in SQL:2006 and later, as discussed in Section D.3.1.3.

9.15.1.4. Xmlforest

xmlforest (content [AS name] [, ...]) → xml (NOT SUPPORTED)

9.15.1.5. Xmlpi

xmlpi (NAME name [, content]) → xml (NOT SUPPORTED)

9.15.1.6. Xmlroot

xmlroot (xml, VERSION {text|NO VALUE} [, STANDALONE {YES|NO|NO VALUE}]) → xml (NOT SUPPORTED)

9.15.1.7. Xmlagg

xmlagg (xml) → xml

The function xmlagg is, unlike the other functions described here, an aggregate function. It concatenates the input values to the aggregate function call, much like xmlconcat does, except that concatenation occurs across rows rather than across expressions in a single row. See Section 9.21 for additional information about aggregate functions.

Example:

```
SELECT xmlagg(x::xml) from (values ('<a/>'),('<b/>')) a(x) → <a/><b/>
```

9.15.2. XML Predicates
The expressions described in this section check properties of xml values.

9.15.2.1. IS DOCUMENT

xml IS DOCUMENT → boolean (NOT SUPPORTED)

The expression IS DOCUMENT returns true if the argument XML value is a proper XML document, false if it is not (that is, it is a content fragment), or null if the argument is null. See Section 8.13 about the difference between documents and content fragments.

9.15.2.2. IS NOT DOCUMENT

xml IS NOT DOCUMENT → boolean (NOT SUPPORTED)

The expression IS NOT DOCUMENT returns false if the argument XML value is a proper XML document, true if it is not (that is, it is a content fragment), or null if the argument is null.

9.15.2.3. XMLEXISTS

XMLEXISTS (text PASSING [BY {REF|VALUE}] xml [BY {REF|VALUE}]) → boolean

The function xmlexists evaluates an XPath 1.0 expression (the first argument), with the passed XML value as its context item. The function returns false if the result of that evaluation yields an empty node-set, true if it yields any other value. The function returns null if any argument is null. A nonnull value passed as the context item must be an XML document, not a content fragment or any non-XML value.

Example:

```
xmlexists('//town[text() = ''Toronto'']' PASSING BY VALUE '<towns><town>Toronto</town><town>Ottawa</town></towns>') →
true
```

9.15.2.4. Xml_is_well_formed (NOT SUPPORTED)

xml_is_well_formed (text) → boolean
xml_is_well_formed_document (text) → boolean
xml_is_well_formed_content (text) → boolean

9.15.3. Processing XML
To process values of data type xml, PostgreSQL offers the functions xpath and xpath_exists, which evaluate XPath 1.0 expressions, and the XMLTABLE table function.

9.15.3.1. Xpath

xpath (xpath text, xml xml [, nsarray text[]]) → xml[] (NOT SUPPORTED)

9.15.3.2. Xpath_exists

xpath_exists (xpath text, xml xml [, nsarray text[]]) → boolean

The function xpath_exists is a specialized form of the xpath function. Instead of returning the individual XML values that satisfy the XPath 1.0 expression, this function returns a Boolean indicating whether the query was satisfied or not (specifically, whether it produced any value other than an empty node-set). This function is equivalent to the XMLEXISTS predicate, except that it also offers support for a namespace mapping argument.

Example:

```
xpath_exists('/my:a/text()', '<my:a xmlns:my="http://example.com">test</my:a>',ARRAY[ARRAY['my', 'http://example.com']])
→ true
```

9.15.3.3. Xmltable

XMLTABLE (
[XMLNAMESPACES (namespace_uri AS namespace_name [, ...]),]
row_expression PASSING [BY {REF|VALUE}] document_expression [BY {REF|VALUE}]
COLUMNS name { type [PATH column_expression] [DEFAULT default_expression] [NOT NULL | NULL]
| FOR ORDINALITY }

[, ...]
) → setof record (NOT SUPPORTED)

9.15.4. Mapping Tables to XML (NOT SUPPORTED)

The following functions map the contents of relational tables to XML values. They can be thought of as XML export functionality:

table_to_xml (table regclass, nulls boolean,
tableforest boolean, targetns text) → xml
query_to_xml (query text, nulls boolean,
tableforest boolean, targetns text) → xml
cursor_to_xml (cursor refcursor, count integer, nulls boolean,
tableforest boolean, targetns text) → xml

9.16. JSON Functions and Operators

This section describes:

functions and operators for processing and creating JSON data

the SQL/JSON path language

To learn more about the SQL/JSON standard, see [sqltr-19075-6]. For details on JSON types supported in PostgreSQL, see Section 8.14.

9.16.1. Processing and Creating JSON Data

Table 9.44 shows the operators that are available for use with JSON data types (see Section 8.14). In addition, the usual comparison operators shown in Table 9.1 are available for jsonb, though not for json. The comparison operators follow the ordering rules for B-tree operations outlined in Section 8.14.4. See also Section 9.21 for the aggregate function json_agg which aggregates record values as JSON, the aggregate function json_object_agg which aggregates pairs of values into a JSON object, and their jsonb equivalents, jsonb_agg and jsonb_object_agg.

Table 9.44. json and jsonb Operators

Operator	Description	Example(s)
json -> integer → json jsonb -> integer → jsonb	Extracts n'th element of JSON array (array elements are indexed from zero, but negative integers count from the end).	`'[{"a":"foo"},{"b":"bar"},{"c":"baz"}]'::json -> 2 → {"c":"baz"}` `'[{"a":"foo"},{"b":"bar"},{"c":"baz"}]'::json -> -3 → {"a":"foo"}`
json -> text → json jsonb -> text → jsonb	Extracts JSON object field with the given key.	`'{"a": {"b":"foo"}}'::json -> 'a' → {"b":"foo"}`

json ->> integer → text jsonb ->> integer → text	Extracts n'th element of JSON array, as text.	`'[1,2,3]'::json ->> 2 → 3`
json ->> text → text jsonb ->> text → text	Extracts JSON object field with the given key, as text.	`'{"a":1,"b":2}'::json ->> 'b' → 2`
json #> text[] → json jsonb #> text[] → jsonb	Extracts JSON sub-object at the specified path, where path elements can be either field keys or array indexes. (NOT SUPPORTED)	`#'{"a": {"b": ["foo","bar"]}}'::json #> '{a,b,1}' → "bar"`
json #>> text[]	Extracts JSON sub-object at the specified path as text. (NOT	

o match the

to effectively search

Operator	Description	Example(s)
jsonb @> jsonb → boolean	Does the first JSON value contain the second? (See Section 8.14.3 for details about containment.)	`'{"a":1, "b":2}'::jsonb @> '{"b":2}'::jsonb → true`
jsonb <@ jsonb → boolean	Is the first JSON value contained in the second?	`'{"b":2}'::jsonb <@ '{"a":1, "b":2}'::jsonb → true`

jsonb ? text → boolean	Does the text string exist as a top-level key or array element within the JSON value?	`'{"a":1, "b":2}'::jsonb ? 'b'` → `true` `'["a", "b", "c"]'::jsonb ? 'b'` → `true`
jsonb ?| text[] → boolean	Do any of the strings in the text array exist as top-level keys or array elements?	`'{"a":1, "b":2, "c":3}'::jsonb ?| array['b', 'd']` → `true`
jsonb ?& text[] → boolean	Do all of the strings in the text array exist as top-level keys or array elements?	`'["a", "b", "c"]'::jsonb ?& array['a', 'b']` → `true`

jsonb || jsonb → jsonb	Concatenates two jsonb values. Concatenating two arrays generates an array containing all the elements of each input. Concatenating two objects generates an object containing the union of their keys, taking the second object's value when there are duplicate keys. All other cases are treated by converting a non-array input into a single-element array, and then proceeding as for two arrays. Does not operate recursively: only the top-level array or object structure is merged.	`'["a", "b"]'::jsonb || '["a", "d"]'::jsonb → ["a", "b", "a", "d"]`
		`'{"a": "b"}'::jsonb || '{"c": "d"}'::jsonb → {"a": "b", "c": "d"}`
		`'[1, 2]'::jsonb || '3'::jsonb → [1, 2, 3]`
		`'{"a": "b"}'::jsonb || '42'::jsonb → [{"a": "b"}, 42]`
		
		To append an array to another array as a single entry, wrap it in an additional layer of array, for example:
		
		`'[1, 2]'::jsonb || jsonb_build_array('[3, 4]'::jsonb) → [1, 2, [3, 4]]`

jsonb - text → jsonb	Deletes a key (and its value) from a JSON object, or matching string value(s) from a JSON array.	`'{"a": "b", "c": "d"}'::jsonb - 'a' → {"c": "d"}` `'["a", "b", "c", "b"]'::jsonb - 'b' → ["a", "c"]`
jsonb - text[] → jsonb	Deletes all matching keys or array elements from the left operand. (NOT SUPPORTED)	`#'{"a": "b", "c": "d"}'::jsonb - '{a,c}'::text[] → {}`

N item type, datetime
s behavior might be

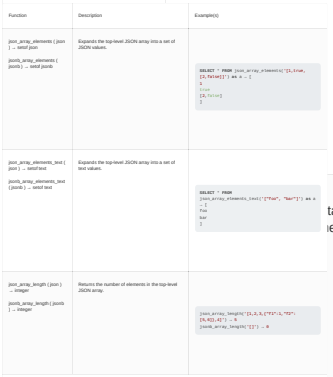
Table 9.46 shows the functions that are available for constructing json and jsonb values.

Table 9.46. JSON Creation Functions

Function	Description	Example(s)

to_json (anyelement) → json to_jsonb (anyelement) → jsonb	Converts any SQL value to json or jsonb. Arrays and composites are converted recursively to arrays and objects (multidimensional arrays become arrays of arrays in JSON). Otherwise, if there is a cast from the SQL data type to json, the cast function will be used to perform the conversion;[a] otherwise, a scalar JSON value is produced. For any scalar other than a number, a Boolean, or a null value, the text representation will be used, with escaping as necessary to make it a valid JSON string value. (NOT SUPPORTED)	`#to_json('Fred said "Hi."'::text) → "Fred said \"Hi.\""` `#to_jsonb(row(42, 'Fred said "Hi."'::text)) → {"f1": 42, "f2": "Fred said \"Hi.\""}`	`'::jsonb - 1 →`
array_to_json (anyarray [, boolean]) → json	Converts an SQL array to a JSON array. The behavior is the same as to_json except that line feeds will be added between top-level array elements if the optional boolean parameter is true.	`array_to_json('{{1,5},{99,100}}'::int[]) → [[1,5],[99,100]]`	`:1}]'::jsonb #-` `"a", {}]`
row_to_json (record [, boolean]) → json	Converts an SQL composite value to a JSON object. The behavior is the same as to_json except that line feeds will be added between top-level elements if the optional boolean parameter is true. (NOT SUPPORTED)	`#row_to_json(row(1,'foo')) → {"f1":1,"f2":"foo"}`	`3,4,5]}'::jsonb @?` `@ > 2)' → true`
json_build_array (VARIADIC "any") → json jsonb_build_array (VARIADIC "any") → jsonb	Builds a possibly-heterogeneously-typed JSON array out of a variadic argument list. Each argument is converted as per to_json or to_jsonb. (NOT SUPPORTED)	`json_build_array(1, 2, 'foo', 4, 5) → [1, 2, "foo", 4, 5]`	`3,4,5]}'::jsonb @@` `' → true`

json_build_object (VARIADIC "any") → json jsonb_build_object (VARIADIC "any") → jsonb	Builds a JSON object out of a variadic argument list. By convention, the argument list consists of alternating keys and values. Key arguments are coerced to text; value arguments are converted as per to_json or to_jsonb. (NOT SUPPORTED)	`#json_build_object('foo', 1, 2, row(3,'bar')) → {"foo" : 1, "2" : {"f1":3,"f2":"bar"}}`
json_object (text[]) → json jsonb_object (text[]) → jsonb	Builds a JSON object out of a text array. The array must have either exactly one dimension with an even number of members, in which case they are taken as alternating key/value pairs, or two dimensions such that each inner array has exactly two elements, which are taken as a key/value pair. All values are converted to JSON strings.	`json_object('{a, 1, b, "def", c, 3.5}') → {"a" : "1", "b" : "def", "c" : "3.5"}` `json_object('{{a, 1}, {b, "def"}, {c, 3.5}}') → {"a" : "1", "b" : "def", "c" : "3.5"}`
	takes keys and values pairwise from erwise it is identical to the one-	`json_object('{a,b}', '{1,2}') → {"a" : "1", "b" : "2"}`



Function	Description	Example(s)
json_array_elements (json) → setof json jsonb_array_elements (jsonb) → setof jsonb	Expands the top-level JSON array into a set of JSON values.	`SELECT * FROM json_array_elements('[1,true, [2,false]]') as a → [ 1 true [2,false] ]`
json_array_elements_text (json) → setof text jsonb_array_elements_text (jsonb) → setof text	Expands the top-level JSON array into a set of text values.	`SELECT * FROM json_array_elements_text('["foo", "bar"]') as a → [ foo bar ]`
json_array_length (json) → integer jsonb_array_length (jsonb) → integer	Returns the number of elements in the top-level JSON array.	`json_array_length('[1,2,3,{"f1":1,"f2":[5,6]},4]') → 5` `jsonb_array_length('[]') → 0`

json_each (json) → setof record (key text, value json) jsonb_each (jsonb) → setof record (key text, value jsonb)	Expands the top-level JSON object into a set of key/value pairs.	
json_each_text (json) → setof record (key text, value text) jsonb_each_text (jsonb) → setof record (key text, value text)	Expands the top-level JSON object into a set of key/value pairs. The returned values will be of type text.	
json_extract_path (from_json json, VARIADIC path_elems text[]) → json jsonb_extract_path (from_json jsonb, VARIADIC path_elems text[]) → jsonb	Extracts JSON sub-object at the specified path. (This is functionally equivalent to the #> operator, but writing the path out as a variadic list can be more convenient in some cases.) [NOT SUPPORTED]	
json_extract_path_text (from_json json, VARIADIC path_elems text[]) → text jsonb_extract_path_text (from_json jsonb, VARIADIC path_elems text[]) → text	Extracts JSON sub-object at the specified path as text. (This is functionally equivalent to the #>> operator.) [NOT SUPPORTED]	

9.16.2. The SQL/JSON Path Language

SQL/JSON path expressions specify the items to be retrieved from the JSON data, similar to XPath expressions used for SQL access to XML. In PostgreSQL, path expressions are implemented as the jsonpath data type and can use any elements described in Section 8.14.7.

JSON query functions and operators pass the provided path expression to the path engine for evaluation. If the expression matches the queried JSON data, the corresponding JSON item, or set of items, is returned. Path expressions are written in the SQL/JSON path language and can include arithmetic expressions and functions.

A path expression consists of a sequence of elements allowed by the jsonpath data type. The path expression is normally evaluated from left to right, but you can use parentheses to change the order of operations. If the evaluation is successful, a sequence of JSON items is produced, and the evaluation result is returned to the JSON query function that completes the specified computation.

To refer to the JSON value being queried (the context item), use the $ variable in the path expression. It can be followed by one or more accessor operators, which go down the JSON structure level by level to retrieve sub-items of the context item. Each operator that follows deals with the result of the previous evaluation step.

For example, suppose you have some JSON data from a GPS tracker that you would like to parse, such as:

{
"track": {
"segments": [
{
"location": [47.763, 13.4034],
"start time": "2018-10-14 10:05:14",
"HR": 73
},
{
"location": [47.706, 13.2635],
"start time": "2018-10-14 10:39:21",
"HR": 135
}
]
}
}
To retrieve the available track segments, you need to use the .key accessor operator to descend through surrounding JSON objects:

json_object_keys (json) → setof text jsonb_object_keys (jsonb) → setof text	Returns the set of keys in the top-level JSON object.	`SELECT * FROM json_object_keys('{"f1":"abc","f2":{"f3":"a", "f4":"b"}}') as x ; x -- f1 f2`

ou typically use the [*] operator. For example, the following path will return the location coordinates for all the

segment only, you can specify the corresponding subscript in the [] accessor operator. Recall that JSON array

ep can be processed by one or more jsonpath operators and methods listed in Section 9.16.2.2. Each method r example, you can get the size of an array:

erators and methods within path expressions appear below in Section 9.16.2.2.

se one or more filter expressions that work similarly to the WHERE clause in SQL. A filter expression begins condition in parentheses:

Filter expressions must be written just after the path evaluation step to which they should apply. The result of that step is filtered to include only those items that satisfy the provided condition. SQL/JSON defines three-valued logic, so the condition can be true, false, or unknown. The unknown value plays the same role as SQL NULL and can be tested for with the is unknown predicate. Further path evaluation steps use only those items for which the filter expression returned true.

The functions and operators that can be used in filter expressions are listed in Table 9.49. Within a filter expression, the @ variable denotes the value being filtered (i.e., one result of the preceding path step). You can write accessor operators after @ to retrieve component items.

For example, suppose you would like to retrieve all heart rate values higher than 130. You can achieve this using the following expression:

$.track.segments[*].HR ? (@ > 130)
To get the start times of segments with such values, you have to filter out irrelevant segments before returning the start times, so the filter expression is applied to the previous step, and the path used in the condition is different:

$.track.segments[*] ? (@.HR > 130)."start time"
You can use several filter expressions in sequence, if required. For example, the following expression selects start times of all segments that contain locations with relevant coordinates and high heart rate values:

$.track.segments[*] ? (@.location[1] < 13.4) ? (@.HR > 130)."start time"
Using filter expressions at different nesting levels is also allowed. The following example first filters all segments by location, and then returns high heart rate values for these segments, if available:

$.track.segments[*] ? (@.location[1] < 13.4).HR ? (@ > 130)
You can also nest filter expressions within each other:

$.track ? (exists(@.segments[*] ? (@.HR > 130))).segments.size()
This expression returns the size of the track if it contains any segments with high heart rate values, or an empty sequence otherwise.

PostgreSQL's implementation of the SQL/JSON path language has the following deviations from the SQL/JSON standard:

A path expression can be a Boolean predicate, although the SQL/JSON standard allows predicates only in filters. This is necessary for implementation of the @@ operator. For example, the following jsonpath expression is valid in PostgreSQL:

$.track.segments[*].HR < 70
There are minor differences in the interpretation of regular expression patterns used in like_regex filters, as described in Section 9.16.2.3.

9.16.2.1. Strict And Lax Modes

When you query JSON data, the path expression may not match the actual JSON data structure. An attempt to access a non-existent member of an object or element of an array results in a structural error. SQL/JSON path expressions have two modes of handling structural errors:

lax (default) — the path engine implicitly adapts the queried data to the specified path. Any remaining structural errors are suppressed and converted to empty SQL/JSON sequences.

strict — if a structural error occurs, an error is raised.

The lax mode facilitates matching of a JSON document structure and path expression if the JSON data does not conform to the expected schema. If an operand does not match the requirements of a particular operation, it can be automatically wrapped as an SQL/JSON array or unwrapped by converting its elements into an SQL/JSON sequence before performing this operation. Besides, comparison operators automatically unwrap their operands in the lax mode, so you can compare SQL/JSON arrays out-of-the-box. An array of size 1 is considered equal to its sole element. Automatic unwrapping is not performed only when:

The path expression contains type() or size() methods that return the type and the number of elements in the array, respectively.

The queried JSON data contain nested arrays. In this case, only the outermost array is unwrapped, while all the inner arrays remain unchanged. Thus, implicit unwrapping can only go one level down within each path evaluation step.

For example, when querying the GPS data listed above, you can abstract from the fact that it stores an array of segments when using the lax mode:

lax $.track.segments.location
In the strict mode, the specified path must exactly match the structure of the queried JSON document to return an SQL/JSON item, so using this path expression will cause an error. To get the same result as in the lax mode, you have to explicitly unwrap the segments array:

strict $.track.segments[*].location
The .** accessor can lead to surprising results when using the lax mode. For instance, the following query selects every HR value twice:

json_populate_record (base anyelement, from_json json) → anyelement jsonb_populate_record (base anyelement, from_json jsonb) → anyelement	Expands the top-level JSON object to a row having the composite type of the base argument. The JSON object is scanned for fields whose names match column names of the output row type, and their values are inserted into those columns of the output. (Fields that do not correspond to any output column name are ignored.) In typical use, the value of base is just NULL, which means that any output columns that do not match any object field will be filled with nulls. However, if base isn't NULL then the values it contains will be used for unmatched columns. (NOT SUPPORTED) To convert a JSON value to the SQL type of an output column, the following rules are applied in sequence: A JSON null value is converted to an SQL null in all cases. If the output column is of type json or jsonb, the JSON value is just reproduced exactly. If the output column is a composite (row) type, and the JSON value is a JSON object, the fields of the object are converted to columns of the output row type by recursive application of these rules. Likewise, if the output column is an array type and the JSON value is a JSON array, the elements of the JSON array are converted to elements of the output array by recursive application of these rules. Otherwise, if the JSON value is a string, the contents of the string are fed to the input conversion function for the column's data type. Otherwise, the ordinary text representation of the JSON value is fed to the input conversion function for the column's data type. While the example below uses a constant JSON value, typical use would be to reference a json or jsonb column laterally from another table in the

;or selects both the segments array and each of its elements, while the .HR accessor automatically unwraps
avoid surprising results, we recommend using the .** accessor only in the strict mode. The following query

And Methods

methods available in jsonpath. Note that while the unary operators and methods can be applied to multiple
h step, the binary operators (addition etc.) can only be applied to single values.

Methods

Operator/Method	Description	Example(s)
number + number → number	Addition	`jsonb_path_query_array('[2]', '$[0] + 3', '{}', false) → [5]`
+ number → number	Unary plus (no operation); unlike addition, this can iterate over multiple values	`jsonb_path_query_array('{"x": [2,3,4]}', '+ $.x', '{}', false) → [2, 3, 4]`
number - number → number	Subtraction	`jsonb_path_query_array('[2]', '7 - $[0]', '{}', false) → [5]`

- number → number	Negation; unlike subtraction, this can iterate over multiple values	`jsonb_path_query_array('{"x": [2,3,4]}', '- $.x', '{}', false) → [-2, -3, -4]`
number * number → number	Multiplication	`jsonb_path_query_array('[4]', '2 * $[0]', '{}', false) → [8]`
number / number → number	Division	`jsonb_path_query_array('[8.5]', '$[0] / 2', '{}', false) → [4.250000000000000]`
number % number → number	Modulo (remainder)	`jsonb_path_query_array('[32]', '$[0] % 10', '{}', false) → [2]`
value . type() → string	Type of the JSON item (see json_typeof)	`jsonb_path_query_array('[1, "2", {}]', '$[*].type()', '{}', false) → ["number", "string", "object"]`

Note
The result type of the datetime() and datetime(template) methods can be date, timetz, time, timestamptz, or timestamp. Both methods determine their result type dynamically.

The datetime() method sequentially tries to match its input string to the ISO formats for date, timetz, time, timestamptz, and timestamp. It stops on the first matching format and emits the corresponding data type.

The datetime(template) method determines the result type according to the fields used in the provided template string.

The datetime() and datetime(template) methods use the same parsing rules as the to_timestamp SQL function does (see Section 9.8), with three exceptions. First, these methods don't allow unmatched template patterns. Second, only the following separators are allowed in the template string: minus sign, period, solidus (slash), comma, apostrophe, semicolon, colon and space. Third, separators in the template string must exactly match the input string.

If different date/time types need to be compared, an implicit cast is applied. A date value can be cast to timestamp or timestamptz, timestamp can be cast to timestamptz, and time to timetz. However, all but the first of these conversions depend on the current TimeZone setting, and thus can only be performed within timezone-aware jsonpath functions.

Table 9.49 shows the available filter expression elements.

Table 9.49. jsonpath Filter Expression Elements

Predicate/Value	Description	Example(s)

value == value → boolean	Equality comparison (this, and the other comparison operators, work on all JSON scalar values)	`jsonb_path_query_array('[1, "a", 1, 3]', '$[*] ? (@ == 1)', '{}', false)` → `[1, 1]` `jsonb_path_query_array('[1, "a", 1, 3]', '$[*] ? (@ == "a")', '{}', false)` → `["a"]`
value != value → boolean value <> value → boolean	Non-equality comparison	`jsonb_path_query_array('[1, 2, 1, 3]', '$[*] ? (@ != 1)', '{}', false)` → `[2, 3]` `jsonb_path_query_array('["a", "b", "c"]', '$[*] ? (@ <> "b")', '{}', false)` → `["a", "c"]`
value < value → boolean	Less-than comparison	`jsonb_path_query_array('[1, 2, 3]', '$[*] ? (@ < 2)', '{}', false)` → `[1]`
value <= value → boolean	Less-than-or-equal-to comparison	`jsonb_path_query_array('["a", "b", "c"]', '$[*] ? (@ <= "b")', '{}', false)` → `["a", "b"]`

value > value → boolean	Greater-than comparison	`jsonb_path_query_array('[1, 2, 3]', '$[*] ? (@ > 2)', '{}', false)` → `[3]`
value >= value → boolean	Greater-than-or-equal-to comparison	`jsonb_path_query_array('[1, 2, 3]', '$[*] ? (@ >= 2)', '{}', false)` → `[2, 3]`
true → boolean	JSON constant true	`jsonb_path_query_array('[{"name": "John", "parent": false}, {"name": "Chris", "parent": true}]', '$[*] ? (@.parent == true)', '{}', false)` → `[{"name": "Chris", "parent": true}]`
false → boolean	JSON constant false	`jsonb_path_query_array('[{"name": "John", "parent": false}, {"name": "Chris", "parent": true}]', '$[*] ? (@.parent == false)', '{}', false)` → `[{"name": "John", "parent": false}]`

ession with the like_regex filter. For example, the following SQL/JSON path query an English vowel:

for case-insensitive match, m to allow ^ and $ to match at newlines, s to allow . behavior to a simple substring match).

ons from the LIKE_REGEX operator, which in turn uses the XQuery standard. Therefore, the like_regex filter is implemented using the POSIX regular minor discrepancies from standard SQL/JSON behavior, which are cataloged in bilities described there do not apply to SQL/JSON, as it translates the XQuery flag

ON path string literal, written according to the rules given in Section 8.14.7. This means ular expression must be doubled. For example, to match string values of the root

9.17. Sequence Manipulation Functions (NOT SUPPORTED)

9.18. Conditional Expressions

9.18.1. CASE

The SQL CASE expression is a generic conditional expression, similar to if/else statements in other programming languages:

CASE WHEN condition THEN result
[WHEN ...]
[ELSE result]
END

CASE clauses can be used wherever an expression is valid. Each condition is an expression that returns a boolean result. If the condition's result is true, the value of the CASE expression is the result that follows the condition, and the remainder of the CASE expression is not processed. If the condition's result is not true, any subsequent WHEN clauses are examined in the same manner. If no WHEN condition yields true, the value of the CASE expression is the result of the ELSE clause. If the ELSE clause is omitted and no condition is true, the result is null.

An example:

```
SELECT a,
       CASE WHEN a=1 THEN 'one'
            WHEN a=2 THEN 'two'
            ELSE 'other'
       END as case
    FROM (VALUES (1),(2),(3)) as x(a)

 a | case
---+-------
 1 | one
 2 | two
 3 | other
```

The data types of all the result expressions must be convertible to a single output type. See Section 10.5 for more details.

There is a “simple” form of CASE expression that is a variant of the general form above:

CASE expression
WHEN value THEN result
[WHEN ...]
[ELSE result]
END

The first expression is computed, then compared to each of the value expressions in the WHEN clauses until one is found that is equal to it. If no match is found, the result of the ELSE clause (or a null value) is returned. This is similar to the switch statement in C.

The example above can be written using the simple CASE syntax:

null → value	JSON constant null (note that, unlike in SQL, comparison to null works normally)	`jsonb_path_query_array('[{"name": "Mary", "job": null}, {"name": "Michael", "job": "driver"}]', '$[*] ? (@.job == null) .name', '{}', false) → ["Mary"]`
boolean && boolean → boolean	Boolean AND	`jsonb_path_query_array('[1, 3, 7]', '$[*] ? (@ > 1 && @ < 5)', '{}', false) → [3]`
boolean || boolean → boolean	Boolean OR	`jsonb_path_query_array('[1, 3, 7]', '$[*] ? (@ < 1 || @ > 5)', '{}', false) → [7]`
! boolean → boolean	Boolean NOT	`jsonb_path_query_array('[1, 3, 7]', '$[*] ? (!(@ < 5))', '{}', false) → [7]`
boolean is unknown → boolean	Tests whether a Boolean condition is unknown.	`jsonb_path_query_array('[-1, 2, 7, "foo"]', '$[*] ? ((@ > 0) is unknown)', '{}', false) → ["foo"]`

hat are not needed to determine the result. For example, this is a possible way of

```
ELSE false END;
```

n which subexpressions of an expression are evaluated at different times, so that the ns" is not ironclad. For example a constant 1/0 subexpression will usually result in a CASE arm that would never be entered at run time.

at is not null. Null is returned only if all arguments are null. It is often used to substitute lay, for example:

```
SELECT COALESCE(description, short_description, '(none)') ...
```

This returns description if it is not null, otherwise short_description if it is not null, otherwise (none).

The arguments must all be convertible to a common data type, which will be the type of the result (see Section 10.5 for details).

Like a CASE expression, COALESCE only evaluates the arguments that are needed to determine the result; that is, arguments to the right of the first non-null argument are not evaluated. This SQL-standard function provides capabilities similar to NVL and IFNULL, which are used in some other database systems.

9.18.3. NULLIF

NULLIF(value1, value2) (NOT SUPPORTED)
The NULLIF function returns a null value if value1 equals value2; otherwise it returns value1. This can be used to perform the inverse operation of the COALESCE example given above:

```
SELECT NULLIF(value, '(none)') ...
```

In this example, if value is (none), null is returned, otherwise the value of value is returned.

The two arguments must be of comparable types. To be specific, they are compared exactly as if you had written value1 = value2, so there must be a suitable = operator available.

The result has the same type as the first argument — but there is a subtlety. What is actually returned is the first argument of the implied = operator, and in some cases that will have been promoted to match the second argument's type. For example, NULLIF(1, 2.2) yields numeric, because there is no integer = numeric operator, only numeric = numeric.

9.18.4. GREATEST and LEAST

GREATEST(value [, ...])
LEAST(value [, ...])

The GREATEST and LEAST functions select the largest or smallest value from a list of any number of expressions. The expressions must all be convertible to a common data type, which will be the type of the result (see Section 10.5 for details). NULL values in the list are ignored. The result will be NULL only if all the expressions evaluate to NULL. (NOT SUPPORTED)

Note that GREATEST and LEAST are not in the SQL standard, but are a common extension. Some other databases make them return NULL if any argument is NULL, rather than only when all are NULL.

9.19. Array Functions and Operators

Table 9.51 shows the specialized operators available for array types. In addition to those, the usual comparison operators shown in Table 9.1 are available for arrays. The comparison operators compare the array contents element-by-element, using the default B-tree comparison function for the element data type, and sort based on the first difference. In multidimensional arrays the elements are visited in row-major order (last subscript varies most rapidly). If the contents of two arrays are equal but the dimensionality is different, the first difference in the dimensionality information determines the sort order.

string like_regex string [flag string] → boolean	Tests whether the first operand matches the regular expression given by the second operand, optionally with

Operator	Description	Example(s)
anyarray @> anyarray → boolean	Does the first array contain the second, that is, does each element appearing in the second array equal some element of the first array? (Duplicates are not treated specially, thus ARRAY[1] and ARRAY[1,1] are each considered to contain the other.)	`ARRAY[1,4,3] @> ARRAY[3,1,3] → true`
anyarray <@ anyarray → boolean	Is the first array contained by the second?	`ARRAY[2,2,7] <@ ARRAY[1,7,4,2,6] → true`
anyarray && anyarray → boolean	Do the arrays overlap, that is, have any elements in common?	`ARRAY[1,4,3] && ARRAY[2,1] → true`
anycompatiblearray || anycompatiblearray → anycompatiblearray	Concatenates the two arrays. Concatenating a null or empty array is a no-op; otherwise the arrays must have the same number of dimensions (as illustrated by the first example) or differ in number of dimensions by one (as illustrated by the second). If the arrays are not of identical element types, they will be coerced to a common type (see Section 10.5). (NOT SUPPORTED)	`#ARRAY[1,2,3] || ARRAY[4,5,6,7] → {1,2,3,4,5,6,7}` `#ARRAY[1,2,3] || ARRAY[[4,5,6],[7,8,9.9]] → {{1,2,3},{4,5,6}, {7,8,9.9}}`

anycompatible || anycompatiblearray → anycompatiblearray	Concatenates an element onto the front of an array (which must be empty or one-dimensional). (NOT SUPPORTED)	`#3 || ARRAY[4,5,6] → {3,4,5,6}`
anycompatiblearray || anycompatible → anycompatiblearray	Concatenates an element onto the end of an array (which must be empty or one-dimensional). (NOT SUPPORTED)	`#ARRAY[4,5,6] || 7 → {4,5,6,7}`

operators support indexed

amples of the use of these

Function	Description	Example(s)
array_append (anycompatiblearray, anycompatible) → anycompatiblearray	Appends an element to the end of an array (same as the anycompatiblearray || anycompatible operator). (NOT SUPPORTED)	`#array_append(ARRAY[1,2], 3) → {1,2,3}`
array_cat (anycompatiblearray, anycompatiblearray) → anycompatiblearray	Concatenates two arrays (same as the anycompatiblearray || anycompatiblearray operator). (NOT SUPPORTED)	`#array_cat(ARRAY[1,2,3], ARRAY[4,5]) → {1,2,3,4,5}`
array_dims (anyarray) → text	Returns a text representation of the array's dimensions.	`array_dims(ARRAY[[1,2,3], [4,5,6]]) → [1:2][1:3]`

array_fill (anyelement, integer[] [, integer[]]) → anyarray	Returns an array filled with copies of the given value, having dimensions of the lengths specified by the second argument. The optional third argument supplies lower-bound values for each dimension (which default to all 1). (NOT SUPPORTED)	`#array_fill(11, ARRAY[2,3]) → {{11,11,11}, {11,11,11}}` `#array_fill(7, ARRAY[3], ARRAY[2]) → [2:4]={7,7,7}`

array_length (anyarray, integer) → integer	Returns the length of the requested array dimension. (Produces NULL instead of 0 for empty or missing array dimensions.)	`array_length(array[1,2,3], 1) → 3`
		`#array_length(array[]::int[], 1) → NULL`
		`array_length(array['text'], 2) → NULL`

array_lower (anyarray, integer) → integer	Returns the lower bound of the requested array dimension.	`array_lower('[0:2]={1,2,3}'::integer[], 1) → 0`
array_ndims (anyarray) → integer	Returns the number of dimensions of the array.	`array_ndims(ARRAY[[1,2,3], [4,5,6]]) → 2`

array_position (anycompatiblearray, anycompatible [, integer]) → integer	Returns the subscript of the first occurrence of the second argument in the array, or NULL if it's not present. If the third argument is given, the search begins at that subscript. The array must be one-dimensional. Comparisons are done using IS NOT DISTINCT FROM semantics, so it is possible to search for NULL. (NOT SUPPORTED)	`#array_position(ARRAY['sun', 'mon', 'tue', 'wed', 'thu', 'fri', 'sat'], 'mon') → 2`
array_positions (anycompatiblearray, anycompatible) → integer[]	Returns an array of the subscripts of all occurrences of the second argument in the array given as first argument. The array must be one-dimensional. Comparisons are done using IS NOT DISTINCT FROM semantics, so it is possible to search for NULL. NULL is returned only if the array is NULL; if the value is not found in the array, an empty array is returned. (NOT SUPPORTED)	`#array_positions(ARRAY['A','A','B','A'], 'A') → {1,2,4}`
array_prepend (anycompatible, anycompatiblearray) → anycompatiblearray	Prepends an element to the beginning of an array (same as the anycompatible || anycompatiblearray operator). (NOT SUPPORTED)	`#array_prepend(1, ARRAY[2,3]) → {1,2,3}`
array_remove (anycompatiblearray, anycompatible) → anycompatiblearray	Removes all elements equal to the given value from the array. The array must be one-dimensional. Comparisons are done using IS NOT DISTINCT FROM semantics, so it is possible to remove NULLs. (NOT SUPPORTED)	`#array_remove(ARRAY[1,2,3,2], 2) → {1,3}`
array_replace (anycompatiblearray, anycompatible, anycompatible) → anycompatiblearray	Replaces each array element equal to the second argument with the third argument. (NOT SUPPORTED)	`#array_replace(ARRAY[1,2,5,4], 5, 3) → {1,2,3,4}`

Note
There are two differences in the behavior of string_to_array from pre-9.1 versions of PostgreSQL. First, it will return an empty (zero-element) array rather than NULL when the input string is of zero length. Second, if the delimiter string is NULL, the function splits the input into individual characters, rather than returning NULL as before.

See also Section 9.21 about the aggregate function array_agg for use with arrays.

9.20. Range/Multirange Functions and Operators (NOT SUPPORTED)

9.21. Aggregate Functions

e aggregate functions are listed in Table 9.57
ons are listed in Table 9.59 while the built-in
d to aggregate functions, are listed in Table 9.61.
Section 2.7 for additional introductory information.

ch as parallel aggregation.

array_to_string (array anyarray, delimiter text [, null_string text]) → text	Converts each array element to its text representation, and concatenates those separated by the delimiter string. If null_string is given and is not NULL, then NULL array entries are represented by that string; otherwise, they are omitted.		`array_to_string(ARRAY[1, 2, 3, NULL, 5], ',', '*') → 1,2,3,*,5`

Function	Description	Partial Mode	Example
array_agg (anynonarray) → anyarray	Collects all the input values, including nulls, into an array.	No	`SELECT array_agg(x) FROM (VALUES (1),(2)) a(x) → {1,2}`
array_agg (anyarray) → anyarray	Concatenates all the input arrays into an array of one higher dimension. (The inputs must all have the same dimensionality, and cannot be empty or null.)	No	`SELECT array_agg(x) FROM (VALUES (Array[1,2]),(Array[3,4])) a(x) → {{1,2},{3,4}}`

avg (smallint) → numeric
avg (integer) → numeric
avg (bigint) → numeric
avg (numeric) → numeric
avg (real) → double precision
avg (double precision) → double precision
avg (interval) → interval
	Computes the average (arithmetic mean) of all the non-null input values.	Yes	`SELECT avg(x::smallint) FROM (VALUES (1),(2),(3)) a(x) → 2.0000000000000000`
			`SELECT avg(x::integer) FROM (VALUES (1),(2),(3)) a(x) → 2.0000000000000000`
			`SELECT avg(x::bigint) FROM (VALUES (1),(2),(3)) a(x) → 2.0000000000000000`
			`SELECT avg(x::numeric) FROM (VALUES (1),(2),(3)) a(x) → 2.0000000000000000`
			`SELECT avg(x::real) FROM (VALUES (1),(2),(3)) a(x) → 2`
			`SELECT avg(x::double precision) FROM (VALUES (1),(2),(3)) a(x) → 2`
			`SELECT avg(cast(x as interval day)) FROM (VALUES ('1'),('2'),('3')) a(x) → 2 days`

`array_upper(ARRAY[1,8,3,7], 1) → 4`

`cardinality(ARRAY[[1,2],[3,4]]) → 4`

`trim_array(ARRAY[1,2,3,4,5,6], 2) → {1,2,3,4}`

bit_and (smallint) → smallint
bit_and (integer) → integer
bit_and (bigint) → bigint
bit_and (bit) → bit
	Computes the bitwise AND of all non-null input values.	Yes	`SELECT bit_and(x::smallint) FROM (VALUES (5),(6),(7)) a(x) → 4`
			`SELECT bit_and(x::integer) FROM (VALUES (5),(6),(7)) a(x) → 4`
			`SELECT bit_and(x::bigint) FROM (VALUES (5),(6),(7)) a(x) → 4`
			`SELECT bit_and(x::bit(3)) FROM (VALUES ('101'),('110'),('111')) a(x) → 100`

bit_or (smallint) → smallint
bit_or (integer) → integer
bit_or (bigint) → bigint
bit_or (bit) → bit
	Computes the bitwise OR of all non-null input values.	Yes	`SELECT bit_or(x::smallint) FROM (VALUES (4),(5),(6)) a(x) → 7`
			`SELECT bit_or(x::integer) FROM (VALUES (4),(5),(6)) a(x) → 7`
			`SELECT bit_or(x::bigint) FROM (VALUES (4),(5),(6)) a(x) → 7`
			`SELECT bit_or(x::bit(3)) FROM (VALUES ('100'),('101'),('110')) a(x) → 111`

bit_xor (smallint) → smallint
bit_xor (integer) → integer
bit_xor (bigint) → bigint
bit_xor (bit) → bit
	Computes the bitwise exclusive OR of all non-null input values. Can be useful as a checksum for an unordered set of values.	Yes	`SELECT bit_xor(x::smallint) FROM (VALUES (5),(6),(6)) a(x) → 5`
			`SELECT bit_xor(x::integer) FROM (VALUES (5),(6),(6)) a(x) → 5`
			`SELECT bit_xor(x::bigint) FROM (VALUES (5),(6),(6)) a(x) → 5`
			`SELECT bit_xor(x::bit(3)) FROM (VALUES ('101'),('110'),('110')) a(x) → 101`

```
SELECT * FROM unnest(ARRAY[1,2]) as a → [
1
2
]

SELECT * FROM unnest(ARRAY[['foo','bar'],
['baz','quux']]) as a → [
foo
bar
baz
quux
]
```

```
#SELECT * FROM unnest(ARRAY[1,2],
ARRAY['foo','bar','baz']) as x(a,b) → [
1,foo
2,bar
,baz
]
```

It should be noted that except for count, these functions return a null value when no rows are selected. In particular, sum of no rows returns null, not zero as one might expect, and array_agg returns null rather than an empty array when there are no input rows. The coalesce function can be used to substitute zero or an empty array for null when necessary.

bool_and (boolean) → boolean	Returns true if all non-null input values are true, otherwise false.	Yes	`SELECT bool_and(x) FROM (VALUES (null),(false),(true)) a(x) → false`
			`SELECT bool_and(x) FROM (VALUES (null),(true),(true)) a(x) → true`
			`SELECT bool_and(x) FROM (VALUES (null::bool),(null::bool),(null::bool)) a(x) → NULL`

bool_or (boolean) → boolean	Returns true if any non-null input value is true, otherwise false.	Yes	`SELECT bool_or(x) FROM (VALUES (null),(false),(false)) a(x) → false`
			`SELECT bool_or(x) FROM (VALUES (null),(false),(true)) a(x) → true`
			`SELECT bool_or(x) FROM (VALUES (null::bool),(null::bool),(null::bool)) a(x) → NULL`

count (*) → bigint	Computes the number of input rows.	Yes	`SELECT count(*) FROM (VALUES (4),(5),(6)) a(x) → 3`
count (any) → bigint	Computes the number of input rows in which the input value is not null.	Yes	`SELECT count(x) FROM (VALUES (4),(null),(6)) a(x) → 2`

, json_object_agg, jsonb_object_agg, string_agg, and xmlagg, as well as similar user-
ent result values depending on the order of the input values. This ordering is unspecified by
ause within the aggregate call, as shown in Section 4.2.7. Alternatively, supplying the input
nple:

```
R BY y DESC) AS tab;
```

contains additional processing, such as a join, because that might cause the subquery's

nd to the standard SQL aggregates every and any or some. PostgreSQL supports every,
nto the standard syntax:

```
t1 ...;
```

uery, or as being an aggregate function, if the subquery returns one row with a Boolean
aggregates.

anagement systems might be disappointed by the performance of the count aggregate

tgreSQL will need to scan either the entire table or the entirety of an index that includes all

atistical analysis. (These are separated out merely to avoid cluttering the listing of more-
ing numeric_type are available for all the types smallint, integer, bigint, numeric, real, and double precision. Where the description mentions N, it means the number of input rows for which all the input expressions are non-null. In all cases, null is returned if the computation is meaningless, for example when N is zero.

Table 9.58. Aggregate Functions for Statistics

Function	Description	Partial Mode	Examples
corr (Y double precision, X double precision) → double precision	Computes the correlation coefficient.	Yes	`SELECT corr(x,y) FROM (VALUES (1,2),(2,3),(3,1)) a(x,y) → -0.5`
covar_pop (Y double precision, X double precision) → double precision	Computes the population covariance.	Yes	`SELECT covar_pop(x,y) FROM (VALUES (1,2),(2,3),(3,1)) a(x,y) → -0.3333333333333333`

covar_samp (Y double precision, X double precision) → double precision	Computes the sample covariance.	Yes	`SELECT covar_samp(x,y) FROM (VALUES (1,2),(2,3),(3,1)) a(x,y) → -0.5`
regr_avgx (Y double precision, X double precision) → double precision	Computes the average of the independent variable, sum(X)/N.	Yes	`SELECT regr_avgx(x,y) FROM (VALUES (1,2),(2,3),(3,1)) a(x,y) → 2`
regr_avgy (Y double precision, X double precision) → double precision	Computes the average of the dependent variable, sum(Y)/N.	Yes	`SELECT regr_avgy(x,y) FROM (VALUES (1,2),(2,3),(3,1)) a(x,y) → 2`
regr_count (Y double precision, X double precision) → bigint	Computes the number of rows in which both inputs are non-null.	Yes	`SELECT regr_count(x,y) FROM (VALUES (1,2),(2,3),(3,1)) a(x,y) → 3`

regr_intercept (Y double precision, X double precision) → double precision	Computes the y-intercept of the least-squares-fit linear equation determined by the (X, Y) pairs.	Yes	`SELECT regr_intercept(x,y) FROM (VALUES (1,2),(2,3),(3,1)) a(x,y) → 3`
regr_r2 (Y double precision, X double precision) → double precision	Computes the square of the correlation coefficient.	Yes	`SELECT regr_r2(x,y) FROM (VALUES (1,2),(2,3),(3,1)) a(x,y) → 0.25`
regr_slope (Y double precision, X double precision) → double precision	Computes the slope of the least-squares-fit linear equation determined by the (X, Y) pairs.	Yes	`SELECT regr_slope(x,y) FROM (VALUES (1,2),(2,3),(3,1)) a(x,y) → -0.5`
regr_sxx (Y double precision, X double precision) → double precision	Computes the "sum of squares" of the independent variable, `sum(X^2) - sum(X)^2/N`.	Yes	`SELECT regr_sxx(x,y) FROM (VALUES (1,2),(2,3),(3,1)) a(x,y) → 2`

. These functions are sometimes referred to as "inverse
ay also take a direct argument that is not aggregated, but is
r those that take a fraction parameter, the fraction value
y produces a null result.

Function	Description	Partial Mode
mode () WITHIN GROUP (ORDER BY anyelement) → anyelement	Computes the mode, the most frequent value of the aggregated argument (arbitrarily choosing the first one if there are multiple equally-frequent values). The aggregated argument must be of a sortable type.	No
percentile_cont (fraction double precision) WITHIN GROUP (ORDER BY double precision) → double precision percentile_cont (fraction double precision) WITHIN GROUP (ORDER BY interval) → interval	Computes the continuous percentile, a value corresponding to the specified fraction within the ordered set of aggregated argument values. This will interpolate between adjacent input items if needed.	No
percentile_cont (fractions double precision[]) WITHIN GROUP (ORDER BY double precision) → double precision[] percentile_cont (fractions double precision[]) WITHIN GROUP (ORDER BY interval) → interval[]	Computes multiple continuous percentiles. The result is an array of the same dimensions as the fractions parameter, with each non-null element replaced by the (possibly interpolated) value corresponding to that percentile.	No

percentile_disc (fraction double precision) WITHIN GROUP (ORDER BY anyelement) → anyelement	Computes the discrete percentile, the first value within the ordered set of aggregated argument values whose position in the ordering equals or exceeds the specified fraction. The aggregated argument must be of a sortable type.	No
percentile_disc (fractions double precision[]) WITHIN GROUP (ORDER BY anyelement) → anyarray	Computes multiple discrete percentiles. The result is an array of the same dimensions as the fractions parameter, with each non-null element replaced by the input value corresponding to that percentile. The aggregated argument must be of a sortable type.	No

			`SELECT regr_syy(x,y) FROM (VALUES (1,2),(2,3),(3,1)) a(x,y) → 2`
stddev (numeric_type) → double precision for real or double precision, otherwise numeric	This is a historical alias for stddev_samp.	Yes	`SELECT stddev(x) FROM (VALUES (1),(2),(3)) a(x) → 1.0000000000000000000`

ow function of the same name defined in Section 9.22. In
d have returned for the “hypothetical” row constructed from
ed_args. For each of these functions, the list of direct
ts given in sorted_args. Unlike most built-in aggregates,
l values sort according to the rule specified in the ORDER

Function	Description	Partial Mode
rank (args) WITHIN GROUP (ORDER BY sorted_args) → bigint	Computes the rank of the hypothetical row, with gaps; that is, the row number of the first row in its peer group.	No
dense_rank (args) WITHIN GROUP (ORDER BY sorted_args) → bigint	Computes the rank of the hypothetical row, without gaps; this function effectively counts peer groups.	No
percent_rank (args) WITHIN GROUP (ORDER BY sorted_args) → double precision	Computes the relative rank of the hypothetical row, that is (rank - 1) / (total rows - 1). The value thus ranges from 0 to 1 inclusive.	No
cume_dist (args) WITHIN GROUP (ORDER BY sorted_args) → double precision	Computes the cumulative distribution, that is (number of rows preceding or peers with hypothetical row) / (total rows). The value thus ranges from 1/N to 1.	No

Table 9.61. Grouping Operations

Function	Description
GROUPING (group_by_expression(s)) → integer	Returns a bit mask indicating which GROUP BY expressions are not included in the current grouping set. Bits are assigned with the rightmost argument corresponding to the least-significant bit; each bit is 0 if the corresponding expression is included in the grouping criteria of the grouping set generating the current result row, and 1 if it is not included. (NOT SUPPORTED)

The grouping operations shown in Table 9.61 are used in conjunction with grouping sets (see Section 7.2.4) to distinguish result rows. The arguments to the GROUPING function are not actually evaluated, but they must exactly match expressions given in the GROUP BY clause of the

stddev_samp (numeric_type) → double precision for real or double precision, otherwise numeric	Computes the sample standard deviation of the input values.	Yes	`SELECT stddev_samp(x) FROM (VALUES (1),(2),(3)) a(x) → 1.0000000000000000000`
variance (numeric_type) → double precision for real or double precision, otherwise numeric	This is a historical alias for var_samp.	Yes	`SELECT variance(x) FROM (VALUES (1),(2),(3)) a(x) → 1.0000000000000000000`
var_pop (numeric_type) → double precision for real or double precision, otherwise numeric	Computes the population variance of the input values (square of the population standard deviation).	Yes	`SELECT var_pop(x) FROM (VALUES (1),(2),(3)) a(x) → 0.66666666666666666667`
var_samp (numeric_type) → double precision for real or double precision, otherwise numeric	Computes the sample variance of the input values (square of the sample standard deviation).	Yes	`SELECT var_samp(x) FROM (VALUES (1),(2),(3)) a(x) → 1.0000000000000000000`

```
GROUP BY ROLLUP(make,model);
```

ormally, over both the grouping columns. The value 1
ndicates that neither make nor model was grouped by in the

e related to the current query row. See Section 3.5 for an

e invoked using window function syntax, i.e., an OVER

rdered-set or hypothetical-set aggregates) can be used as a
ns act as window functions only when an OVER clause
ntire set.

Table 9.62. General-Purpose Window Functions

Function	Description	Examples
row_number () → bigint	Returns the number of the current row within its partition, counting from 1.	`SELECT row_number() OVER (ORDER BY x) FROM (VALUES (4),(5),(6)) a(x) → [ 1 2 3 ]`

rank () → bigint	Returns the rank of the current row, with gaps; that is, the row_number of the first row in its peer group.	`SELECT rank() OVER (ORDER BY x) FROM (VALUES (4),(5),(5),(6)) a(x) → [ 1 2 2 4 ]`
dense_rank () → bigint	Returns the rank of the current row, without gaps; this function effectively counts peer groups.	`SELECT dense_rank() OVER (ORDER BY x) FROM (VALUES (4),(5),(5),(6)) a(x) → [ 1 2 2 3 ]`
percent_rank () → double precision	Returns the relative rank of the current row, that is (rank - 1) / (total partition rows - 1). The value thus ranges from 0 to 1 inclusive.	`SELECT percent_rank() OVER (ORDER BY x) FROM (VALUES (4),(5),(5),(6)) a(x) → [ 0 0.3333333333333333 0.3333333333333333 1 ]`

cume_dist () → double precision	Returns the cumulative distribution, that is (number of partition rows preceding or peers with current row) / (total partition rows). The value thus ranges from 1/N to 1.	`SELECT cume_dist() OVER (ORDER BY x) FROM (VALUES (4),(5),(5),(6)) a(x) → [ 0.25 0.75 0.75 1 ]`
ntile (num_buckets integer) → integer	Returns an integer ranging from 1 to the argument value, dividing the partition as equally as possible.	`SELECT ntile(2) OVER (ORDER BY x) FROM (VALUES (4),(5),(5),(6)) a(x) → [ 1 1 2 2 ]`
lag (value anycompatible [, offset integer [, default anycompatible]]) → anycompatible	Returns value evaluated at the row that is offset rows before the current row within the partition; if there is no such row, instead returns default (which must be of a type compatible with value). Both offset and default are evaluated with respect to the current row. If omitted, offset defaults to 1 and default to NULL.	`SELECT lag(x) OVER (ORDER BY x) FROM (VALUES (4),(5),(5),(6)) a(x) → [ NULL 4 5 5 ]`

All of the functions listed in Table 9.62 depend on the sort ordering specified by the ORDER BY clause of the associated window definition. Rows that are not distinct when considering only the ORDER BY columns are said to be peers. The four ranking functions (including cume_dist) are defined so that they give the same answer for all rows of a peer group.

Note that first_value, last_value, and nth_value consider only the rows within the “window frame”, which by default contains the rows from the start of the partition through the last peer of the current row. This is likely to give unhelpful results for last_value and sometimes also nth_value. You can redefine the frame by adding a suitable frame specification (RANGE, ROWS or GROUPS) to the OVER clause. See Section 4.2.8 for more information about frame specifications.

lead (value anycompatible [, offset integer [, default anycompatible]]) → anycompatible	Returns value evaluated at the row that is offset rows after the current row within the partition; if there is no such row, instead returns default (which must be of a type compatible with value). Both offset and default are evaluated with respect to the current row. If omitted, offset defaults to 1 and default to NULL.	`SELECT lead(x) OVER (ORDER BY x) FROM (VALUES (4),(5),(5),(6)) a(x) → [ 5 5 6 NULL ]`
first_value (value anyelement) → anyelement	Returns value evaluated at the row that is the first row of the window frame.	`SELECT first_value(x) OVER (ORDER BY x) FROM (VALUES (4),(5),(5),(6)) a(x) → [ 4 4 4 4 ]`
last_value (value anyelement) → anyelement	Returns value evaluated at the row that is the last row of the window frame.	`SELECT last_value(x) OVER (ORDER BY x) FROM (VALUES (4),(5),(5),(6)) a(x) → [ 4 5 5 6 ]`

window frame. An aggregate used
or may not be what's wanted. To
DING AND UNBOUNDED

and nth_value. This is not
LLS. Likewise, the standard's FROM
orted. (You can achieve the result of

forms documented in this section

ine whether it returns any rows. If it
"false".

luation of the subquery.

ot all the way to completion. It is
occur might be unpredictable.

tput list of the subquery is normally
...). There are exceptions to this rule

n if there are several matching tab2

y) WHERE x=y) → [

on is evaluated and compared to
" if no equal row is found (including

hand row yields null, the result of the
ull values.

The left-hand side of this form of IN is a row constructor, as described in Section 4.2.13. The right-hand side is a parenthesized subquery, which must return exactly as many columns as there are expressions in the left-hand row. The left-hand expressions are evaluated and compared row-wise to each row of the subquery result. The result of IN is "true" if any equal subquery row is found. The result is "false" if no equal row is found (including the case where the subquery returns no rows).

As usual, null values in the rows are combined per the normal rules of SQL Boolean expressions. Two rows are considered equal if all their corresponding members are non-null and equal; the rows are unequal if any corresponding members are non-null and unequal; otherwise the result of that row comparison is unknown (null). If all the per-row results are either unequal or null, with at least one null, then the result of IN is null.

Example

```
SELECT x FROM (VALUES (1),(2),(3)) a(x) WHERE x IN (SELECT y FROM (VALUES (3),(4),(5)) b(y)) → [
3
]
```

9.23.3. NOT IN

expression NOT IN (subquery)

nth_value (value anyelement, n integer) → anyelement	Returns value evaluated at the row that is the n'th row of the window frame (counting from 1); returns NULL if there is no such row.	`SELECT nth_value(x,2) OVER (ORDER BY x) FROM (VALUES (4),(5),(5),(6)) a(x) → [ NULL 5 5 5 ]`

on is evaluated and compared to
ng the case where the subquery

hand row yields null, the result of the
of null values.

de is a parenthesized subquery,
sions are evaluated and compared
und (including the case where the

considered equal if all their corresponding members are non-null and equal; the rows are unequal if any corresponding members are non-null and unequal; otherwise the result of that row comparison is unknown (null). If all the per-row results are either unequal or null, with at least one null, then the result of NOT IN is null.

Example

```
SELECT x FROM (VALUES (1),(2),(3)) a(x) WHERE x NOT IN (SELECT y FROM (VALUES (3),(4),(5)) b(y)) ORDER BY x → [
1
2
]
```

9.23.4. ANY/SOME

expression operator ANY (subquery)
expression operator SOME (subquery)

The right-hand side is a parenthesized subquery, which must return exactly one column. The left-hand expression is evaluated and compared to each row of the subquery result using the given operator, which must yield a Boolean result. The result of ANY is “true” if any true result is obtained. The result is “false” if no true result is found (including the case where the subquery returns no rows).

SOME is a synonym for ANY. IN is equivalent to = ANY.

Note that if there are no successes and at least one right-hand row yields null for the operator's result, the result of the ANY construct will be null, not false. This is in accordance with SQL's normal rules for Boolean combinations of null values.

As with EXISTS, it's unwise to assume that the subquery will be evaluated completely.

row_constructor operator ANY (subquery) (NOT SUPPORTED)
row_constructor operator SOME (subquery) (NOT SUPPORTED)

The left-hand side of this form of ANY is a row constructor, as described in Section 4.2.13. The right-hand side is a parenthesized subquery, which must return exactly as many columns as there are expressions in the left-hand row. The left-hand expressions are evaluated and compared row-wise to each row of the subquery result, using the given operator. The result of ANY is “true” if the comparison returns true for any subquery row. The result is “false” if the comparison returns false for every subquery row (including the case where the subquery returns no rows). The result is NULL if no comparison with a subquery row returns true, and at least one comparison returns NULL.

See Section 9.24.5 for details about the meaning of a row constructor comparison.

Example

```
SELECT x FROM (VALUES (1),(2),(3)) a(x) WHERE x = ANY (SELECT y FROM (VALUES (3),(4),(5)) b(y)) → [
3
]
```

9.23.5. ALL

expression operator ALL (subquery)

The right-hand side is a parenthesized subquery, which must return exactly one column. The left-hand expression is evaluated and compared to each row of the subquery result using the given operator, which must yield a Boolean result. The result of ALL is “true” if all rows yield true (including the case where the subquery returns no rows). The result is “false” if any false result is found. The result is NULL if no comparison with a subquery row returns false, and at least one comparison returns NULL.

NOT IN is equivalent to <> ALL.

As with EXISTS, it's unwise to assume that the subquery will be evaluated completely.

row_constructor operator ALL (subquery) (NOT SUPPORTED)

The left-hand side of this form of ALL is a row constructor, as described in Section 4.2.13. The right-hand side is a parenthesized subquery, which must return exactly as many columns as there are expressions in the left-hand row. The left-hand expressions are evaluated and compared row-wise to each row of the subquery result, using the given operator. The result of ALL is “true” if the comparison returns true for all subquery rows (including the case where the subquery returns no rows). The result is “false” if the comparison returns false for any subquery row. The result is NULL if no comparison with a subquery row returns false, and at least one comparison returns NULL.

See Section 9.24.5 for details about the meaning of a row constructor comparison.

Example

```
SELECT x FROM (VALUES (1),(2),(3)) a(x) WHERE x <> ALL (SELECT y FROM (VALUES (3),(4),(5)) b(y)) ORDER BY x → [
1
2
]
```

9.23.6. Single-Row Comparison

row_constructor operator (subquery) (NOT SUPPORTED)

The left-hand side is a row constructor, as described in Section 4.2.13. The right-hand side is a parenthesized subquery, which must return exactly as many columns as there are expressions in the left-hand row. Furthermore, the subquery cannot return more than one row. (If it returns zero rows, the result is taken to be null.) The left-hand side is evaluated and compared row-wise to the single subquery result row.

See Section 9.24.5 for details about the meaning of a row constructor comparison.

9.24. Row and Array Comparisons

This section describes several specialized constructs for making multiple comparisons between groups of values. These forms are syntactically related to the subquery forms of the previous section, but do not involve subqueries. The forms involving array subexpressions are PostgreSQL extensions; the rest are SQL-compliant. All of the expression forms documented in this section return Boolean (true/false) results.

9.24.1. IN

expression IN (value [, ...])
The right-hand side is a parenthesized list of expressions. The result is “true” if the left-hand expression's result is equal to any of the right-hand expressions. This is a shorthand notation for

expression = value1
OR
expression = value2
OR
...
Note that if the left-hand expression yields null, or if there are no equal right-hand values and at least one right-hand expression yields null, the result of the IN construct will be null, not false. This is in accordance with SQL's normal rules for Boolean combinations of null values.

Example

```
SELECT x FROM (VALUES (1),(2),(3)) a(x) WHERE x IN (1,2) ORDER BY x → [
1
2
]

SELECT x IN (y, z) FROM (VALUES (1,1,2),(2,3,null),(3,4,5),(4,null,null)) a(x,y,z) ORDER BY x → [
true
NULL
false
NULL
]
```

9.24.2. NOT IN

expression NOT IN (value [, ...])
The right-hand side is a parenthesized list of expressions. The result is “true” if the left-hand expression's result is unequal to all of the right-hand expressions. This is a shorthand notation for

expression <> value1
AND
expression <> value2
AND
...
Note that if the left-hand expression yields null, or if there are no equal right-hand values and at least one right-hand expression yields null, the result of the NOT IN construct will be null, not true as one might naively expect. This is in accordance with SQL's normal rules for Boolean combinations of null values.

Tip
x NOT IN y is equivalent to NOT (x IN y) in all cases. However, null values are much more likely to trip up the novice when working with NOT IN than when working with IN. It is best to express your condition positively if possible.

Example

```
SELECT x FROM (VALUES (1),(2),(3)) a(x) WHERE x NOT IN (1,2) ORDER BY x → [
3
]
```

```
SELECT x NOT IN (y, z) FROM (VALUES (1,1,2),(2,3,null),(3,4,5),(4,null,null)) a(x,y,z) ORDER BY x → [
false
NULL
true
NULL
]
```

9.24.3. ANY/SOME (array)

expression operator ANY (array expression)
expression operator SOME (array expression)

The right-hand side is a parenthesized expression, which must yield an array value. The left-hand expression is evaluated and compared to each element of the array using the given operator, which must yield a Boolean result. The result of ANY is “true” if any true result is obtained. The result is “false” if no true result is found (including the case where the array has zero elements).

If the array expression yields a null array, the result of ANY will be null. If the left-hand expression yields null, the result of ANY is ordinarily null (though a non-strict comparison operator could possibly yield a different result). Also, if the right-hand array contains any null elements and no true comparison result is obtained, the result of ANY will be null, not false (again, assuming a strict comparison operator). This is in accordance with SQL's normal rules for Boolean combinations of null values.

SOME is a synonym for ANY.

Example

```
SELECT x FROM (VALUES (1),(2),(3)) a(x) WHERE x = ANY (array[1,2]) ORDER BY x → [
1
2
]
]
```

9.24.4. ALL (array)

expression operator ALL (array expression)

The right-hand side is a parenthesized expression, which must yield an array value. The left-hand expression is evaluated and compared to each element of the array using the given operator, which must yield a Boolean result. The result of ALL is “true” if all comparisons yield true (including the case where the array has zero elements). The result is “false” if any false result is found.

If the array expression yields a null array, the result of ALL will be null. If the left-hand expression yields null, the result of ALL is ordinarily null (though a non-strict comparison operator could possibly yield a different result). Also, if the right-hand array contains any null elements and no false comparison result is obtained, the result of ALL will be null, not true (again, assuming a strict comparison operator). This is in accordance with SQL's normal rules for Boolean combinations of null values.

Example

```
SELECT x FROM (VALUES (1),(2),(3)) a(x) WHERE x <> ALL (array[1,2]) ORDER BY x → [
3
]
]
```

9.24.5. Row Constructor Comparison (NOT SUPPORTED)

row_constructor operator row_constructor

Each side is a row constructor, as described in Section 4.2.13. The two row constructors must have the same number of fields. The given operator is applied to each pair of corresponding fields. (Since the fields could be of different types, this means that a different specific operator could be selected for each pair.) All the selected operators must be members of some B-tree operator class, or be the negator of an = member of a B-tree operator class, meaning that row constructor comparison is only possible when the operator is =, <>, <, <=, >, or >=, or has semantics similar to one of these.

The = and <> cases work slightly differently from the others. Two rows are considered equal if all their corresponding members are non-null and equal; the rows are unequal if any corresponding members are non-null and unequal; otherwise the result of the row comparison is unknown (null).

For the <, <=, > and >= cases, the row elements are compared left-to-right, stopping as soon as an unequal or null pair of elements is found. If either of this pair of elements is null, the result of the row comparison is unknown (null); otherwise comparison of this pair of elements determines the result. For example, ROW(1,2,NULL) < ROW(1,3,0) yields true, not null, because the third pair of elements are not considered.

Note
Prior to PostgreSQL 8.2, the <, <=, > and >= cases were not handled per SQL specification. A comparison like ROW(a,b) < ROW(c,d) was implemented as a < c AND b < d whereas the correct behavior is equivalent to a < c OR (a = c AND b < d).

row_constructor IS DISTINCT FROM row_constructor

This construct is similar to a <> row comparison, but it does not yield null for null inputs. Instead, any null value is considered unequal to (distinct from) any non-null value, and any two nulls are considered equal (not distinct). Thus the result will either be true or false, never null.

row_constructor IS NOT DISTINCT FROM row_constructor

This construct is similar to a = row comparison, but it does not yield null for null inputs. Instead, any null value is considered unequal to (distinct from) any non-null value, and any two nulls are considered equal (not distinct). Thus the result will always be either true or false, never null.

9.24.6. Composite Type Comparison (NOT SUPPORTED)

record operator record

The SQL specification requires row-wise comparison to return NULL if the result depends on comparing two NULL values or a NULL and a non-NULL. PostgreSQL does this only when comparing the results of two row constructors (as in Section 9.24.5) or comparing a row constructor to the output of a subquery (as in Section 9.23). In other contexts where two composite-type values are compared, two NULL field values are considered equal, and a NULL is considered larger than a non-NULL. This is necessary in order to have consistent sorting and indexing behavior for composite types.

Each side is evaluated and they are compared row-wise. Composite type comparisons are allowed when the operator is =, <>, <, <=, > or >=, or has semantics similar to one of these. (To be specific, an operator can be a row comparison operator if it is a member of a B-tree operator class, or is the negator of the = member of a B-tree operator class.) The default behavior of the above operators is the same as for IS [NOT] DISTINCT FROM for row constructors (see Section 9.24.5).

To support matching of rows which include elements without a default B-tree operator class, the following operators are defined for composite type comparison: `*=`, `*<>`, `*<`, `*<=`, `*>`, and `*>=`. These operators compare the internal binary representation of the two rows. Two rows might have a different binary representation even though comparisons of the two rows with the equality operator is true. The ordering of rows under these comparison operators is deterministic but not otherwise meaningful. These operators are used internally for materialized views and might be useful for other specialized purposes such as replication and B-Tree deduplication (see Section 64.4.3). They are not intended to be generally useful for writing queries, though.

9.25. Set Returning Functions

This section describes functions that possibly return more than one row. The most widely used functions in this class are series generating functions, as detailed in Table 9.63 and Table 9.64. Other, more specialized set-returning functions are described elsewhere in this manual. See Section 7.2.1.4 for ways to combine multiple set-returning functions.

Table 9.63. Series Generating Functions

Function	Description
generate_series (start integer, stop integer [, step integer]) → setof integer generate_series (start bigint, stop bigint [, step bigint]) → setof bigint generate_series (start numeric, stop numeric [, step numeric]) → setof numeric	Generates a series of values from start to stop, with a step size of step. step defaults to 1.
generate_series (start timestamp, stop timestamp, step interval) → setof timestamp generate_series (start timestamp with time zone, stop timestamp with time zone, step interval) → setof timestamp with time zone	Generates a series of values from start to stop, with a step size of step.

When step is positive, zero rows are returned if start is greater than stop. Conversely, when step is negative, zero rows are returned if start is less than stop. Zero rows are also returned if any input is NULL. It is an error for step to be zero. Some examples follow:

```
SELECT * FROM generate_series(2,4) a → [
2
3
4
]

SELECT * FROM generate_series(5,1,-2) a → [
5
3
1
]

SELECT * FROM generate_series(4,3) a → [
]

SELECT * FROM generate_series(1.1, 4, 1.3) a → [
1.1
2.4
```

```
3.7
]

-- this example relies on the date-plus-integer operator:
SELECT date '2004-02-05' + s.a AS dates FROM generate_series(0,14,7) AS s(a) → [
2004-02-05
2004-02-12
2004-02-19
]

SELECT * FROM generate_series('2008-03-01 00:00'::timestamp, '2008-03-04 12:00', '10 hours') a → [
2008-03-01 00:00:00
2008-03-01 10:00:00
2008-03-01 20:00:00
2008-03-02 06:00:00
2008-03-02 16:00:00
2008-03-03 02:00:00
2008-03-03 12:00:00
2008-03-03 22:00:00
2008-03-04 08:00:00
]
```

Table 9.64. Subscript Generating Functions (NOT SUPPORTED)

Function	Description
generate_subscripts (array anyarray, dim integer) → setof integer	Generates a series comprising the valid subscripts of the dim'th dimension of the given array.
generate_subscripts (array anyarray, dim integer, reverse boolean) → setof integer	Generates a series comprising the valid subscripts of the dim'th dimension of the given array. When reverse is true, returns the series in reverse order.

```
-- basic usage:
#SELECT generate_subscripts('{NULL,1,NULL,2}'::int[], 1) AS s → [
1
2
3
4
]

SELECT a AS array, s AS subscript, a[s] AS value FROM (SELECT generate_subscripts(a, 1) AS s, a FROM (VALUES
(array[-1,-2]),(array[100,200,300])) s(a)) foo;
```

9.26. System Information Functions and Operators (NOT SUPPORTED)

9.27. System Administration Functions (NOT SUPPORTED)

9.28. Trigger Functions (NOT SUPPORTED)

9.29. Event Trigger Functions (NOT SUPPORTED)

9.30. Statistics Information Functions (NOT SUPPORTED)

Импорт данных из PostgreSQL

Важно

Поддержка синтаксиса PostgreSQL в YDB **находится в разработке**. Использовать её в production окружениях **не рекомендуется**. Запросы в PostgreSQL могут исполняться до нескольких раз медленнее по сравнению с аналогичными запросами на YQL.

Основной сценарий, который можно тестировать — выполнение аналитических запросов к хранимым в YDB данным.

Данные из PostgreSQL в YDB можно импортировать различными способами:

- С помощью pg-dump.
- С помощью импорта данных из файлов.
- С помощью утилиты ydb-importer.

Способ импорта	Способ работы	Сценарии использования
pg-dump	Создание всей необходимой структуры таблиц и данных	Импорт баз данных PostgreSQL целиком
Импорт данных из файлов	Импорт файлов с данными в заранее созданные таблицы базы данных YDB	Импорт данных из баз данных Greenplum или любых других баз данных со сменой структуры хранения данных
ydb-importer	Импорт данных из другой базы данных в заранее созданные таблицы базы данных YDB	Импорт данных из любых баз данных, поддерживающих JDBC-протокол

pg-dump

Данные из PostgreSQL в YDB можно перенести с помощью утилит pg_dump, psql и YDB CLI. Утилиты pg_dump и psql устанавливаются вместе с PostgreSQL, YDB CLI — консольный клиент YDB, который устанавливается отдельно.

Для этого нужно:

1. Сделать дамп данных через pg_dump со следующими параметрами:
   - `--inserts` — для добавления данных через команду INSERT, вместо использования протокола COPY.
   - `--column-inserts` — для добавления данных через команду INSERT с именами колонок.
   - `--rows-per-insert=1000` — для вставки данных пачками и ускорения процесса.
   - `--encoding=utf_8` — YDB поддерживает строковые данные только в UTF-8.
2. Привести дамп к виду, который поддерживается YDB командой `ydb tools pg-convert` YDB CLI.
3. Результат загрузить в YDB в режиме postgres-совместимости.

Команда pg-convert

Команда `ydb tools pg-convert` считывает из файла или stdin'а дамп, полученный утилитой pg_dump, выполняет преобразования и выводит в stdout дамп, который можно отправить в PostgreSQL-совместимую прослойку YDB.

`ydb tools pg-convert` выполняет следующие преобразования:

- Перенос создания первичного ключа в тело команды CREATE TABLE.
- Вырезание схемы `public` из имен таблиц.
- Удаление секции `WITH (...)` в `CREATE TABLE`
- Комментирование неподдерживаемых конструкций (опционально):
  - `SELECT pg_catalog.set_config.*`
  - `ALTER TABLE`

Если CLI не сможет найти первичный ключ таблицы, он автоматически создаст BIGSERIAL колонку `__ydb_stub_id` в качестве первичного ключа.

Общий вид команды:

```
ydb [global options...] tools pg-convert [options...]
```

- `global options` — глобальные параметры.

- `options` — параметры подкоманды.

Параметры подкоманды

Имя	Описание
`-i`	Имя файла, в котором находится изначальный дамп. Если опция не указана, дамп считывается из stdin'a.
`--ignore-unsupported`	При указании этой опции, неподдерживаемые конструкции будут закомментированы в итоговом дампе и продублированы в stderr. По умолчанию, при обнаружении неподдерживаемых конструкций, команда возвращает ошибку. Не относится к выражениям `ALTER TABLE`, задающим первичный ключ таблицы, они комментируются в любом случае.

Важно

При загрузке больших дампов считывание из stdin'a не рекомендуется, поскольку в таком случае весь дамп будет сохранен в оперативной памяти. Рекомендуется использовать опцию с файлом, в таком случае CLI будет держать в памяти небольшую часть дампа.

Пример импорта дампа в YDB

В качестве примера будут загружены данные, сгенерированные pgbench.

1. Поднять докер-контейнеры с PostgreSQL и YDB:

```
docker run --name postgres_container \
    -e POSTGRES_USER=pgroot -e POSTGRES_PASSWORD=1234 \
    -e POSTGRES_DB=local \
    -p 5433:5433 -d postgres:14 -c 'port=5433'
docker run --name ydb-postgres -d --pull always -p 5432:5432 -p 8765:8765 \
    -e POSTGRES_USER=ydbroot -e POSTGRES_PASSWORD=4321 \
    -e YDB_FEATURE_FLAGS=enable_temp_tables,enable_table_pg_types \
    -e YDB_USE_IN_MEMORY_PDISKS=true \
    ghcr.io/ydb-platform/local-ydb:latest
```

Информация по поднимаемым docker-контейнерам:

База данных	PostgreSQL	YDB
Имя контейнера	postgres_container	ydb-postgres
Адрес	postgres://pgroot:1234@localhost:5433/local	postgresql://ydbroot:4321@localhost:5432/local
Порт	5433	5432
Имя пользователя	pgroot	ydbroot
Пароль	1234	4321

2. Сгенерировать данные через pgbench:

```
docker exec postgres_container pgbench postgres://pgroot:1234@localhost:5433/local -i
```

3. Сделать дамп базы через pg_dump:

```
docker exec postgres_container pg_dump postgres://pgroot:1234@localhost:5433/local --inserts \
```

```
    --column-inserts --encoding=utf_8 --rows-per-insert=1000 > dump.sql
```

4. Загрузить дамп базы в YDB:

```
ydb tools pg-convert --ignore-unsupported -i dump.sql | psql postgresql://ydbroot:4321@localhost:5432/local
```

Эта команда использует YDB CLI для преобразования файла `dump.sql` в формат, поддерживаемый YDB в режиме совместимости с PostgreSQL. Затем преобразованный файл перенаправляется в утилиту `psql` для загрузки данных в YDB по протоколу PostgreSQL.

CREATE TABLE (создание таблицы)

Важно

Поддержка синтаксиса PostgreSQL в YDB **находится в разработке**. Использовать её в production окружениях **не рекомендуется**. Запросы в PostgreSQL могут исполняться до нескольких раз медленнее по сравнению с аналогичными запросами на YQL.

Основной сценарий, который можно тестировать — выполнение аналитических запросов к хранимым в YDB данным.

Инструкция `CREATE TABLE` предназначена для создания пустой таблицы в текущей базе данных. Синтаксис команды:

```
CREATE [TEMPORARY | TEMP] TABLE <table name> (

<column name> <column data type> [COLLATE][PRIMARY KEY]

[CONSTRAINT  <constraint name> [PRIMARY KEY <column name>],
...]

);
```

При создании таблицы можно задать:

1. **Тип таблицы**: `TEMPORARY` / `TEMP` – временная таблица, которая автоматически удаляется при завершении сессии. Если параметр не задан (оставлен пустым) – создается постоянная таблица. Любые индексы, созданные во временных таблицах, также будут удалены при завершении сессиии, следовательно они тоже являются временными. Допускается существование временной таблицы и постоянной таблицы с одинаковым именем, в этом случае будет выбрана временная таблица.
2. **Имя таблицы**: `<table name>` – можно использовать английские буквы в нижнем регистре, цифры, нижнее подчёркивание и знак доллара ($). Например, название таблицы "People" будет сохранено как "people";
3. **Имя столбца/колонки**: – действую такие же правила нейминга как и для имен таблиц;
4. **Тип данных**: – указываются стандартные типы данных PostgreSQL;
5. **Правило сортировки**: `COLLATE` – правила сортировки позволяют устанавливать порядок сортировки и особенности классификации символов в отдельных столбцах или даже при выполнении отдельных операций. К сортируемым типам относятся: `text`, `varchar` и `char`. Можно указать локализацию (`ru_RU`, `en_US`), используемую для определения правил сортировки и сравнения строк в указанных столбцах.
6. Первичный ключ таблицы: `PRIMARY KEY` – обязательное условие при создании таблицы в режиме совместимости YDB с PostgreSQL;
7. Ограничения на уровне таблицы (может быть множество, перечисляются через запятую): `CONSTRAINT` – данный тип ограничения используется как альтернативный синтаксис записи поколоночным ограничениям, или когда нужно задать одинаковые условия ограничения на несколько колонок. Для указания ограничения необходимо указать:
   - Ключевое слово `CONSTRAINT`;
   - Имя ограничения . Правила создания идентификатора для ограничения такие же, как у названия таблиц и названия колонок;
   - Ограничение. Например, `PRIMARY KEY (<column name>)`.

Создание двух таблиц с первичным ключом и автоинкрементом

Таблица `people`	Таблица `social_card`

ть автоинкремент, который

```
CREATE TABLE people (
    id                Serial PRIMARY KEY,
    name              Text,
    lastname          Text,
    age               Int,
    country           Text,
    state             Text,
    city              Text,
    birthday          Date,
    sex               Text,
    social_card_number Int
);
```

```
CREATE TABLE social_card (
    id                  Serial PRIMARY KEY,
    social_card_number  Int,
    card_holder_name    Text,
    card_holder_lastname Text,
    issue               Date,
    expiry              Date,
    issuing_authority   Text,
    category            Text
);
```

В этом примере мы создали таблицу "people" с ограничением (блоком `CONSTRAINT`), в котором задали первичный ключ (`PRIMARY KEY`) для колонки "id". Альтернативная запись может выглядеть так: `PRIMARY KEY(id)` без указания ключевого слова `CONSTRAINT`.

Создание временной таблицы

```
CREATE TEMPORARY TABLE people (
    id serial PRIMARY KEY,
    name TEXT NOT NULL
);
```

Временная таблица задается через ключевые слова `TEMPORARY` или `TEMP`.

Создание таблицы с условиями сортировки

```
CREATE TABLE people (
    id                  Serial PRIMARY KEY,
    name                Text COLLATE "en_US",
    lastname            Text COLLATE "en_US",
    age                 Int,
    country             Text,
    state               Text,
    city                Text,
    birthday            Date,
    sex                 Text,
    social_card_number  Int
);
```

В этом примере колонки "name" и "lastname" используют сортировку с `en_US` локализацией.

Примечание

В отличии от PostgreSQL, в YDB с используются оптимистические блокировки. Это значит, что транзакции проверяют условия выполнения необходимых блокировок в конце своей работы, а не в начале. Если за время выполнения транзакции блокировка была нарушена – такая транзакция завершится ошибкой `Transaction locks invalidated`. В этом случае можно попробовать выполнить аналогичную транзакцию снова.

DROP TABLE (удаление таблицы)

Важно

Поддержка синтаксиса PostgreSQL в YDB **находится в разработке**. Использовать её в production окружениях **не рекомендуется**. Запросы в PostgreSQL могут исполняться до нескольких раз медленнее по сравнению с аналогичными запросами на YQL.

Основной сценарий, который можно тестировать — выполнение аналитических запросов к хранимым в YDB данным.

Синтаксис инструкции `DROP TABLE`:

```
DROP TABLE [IF EXISTS] <table name>;
```

Инструкция `DROP TABLE <table name>;` предназначена для удаления таблицы. Например: `DROP TABLE people;`. Если удаляемая таблица отсутствует – будет выведено сообщение об ошибки:

```
Error: Cannot find table '...' because it does not exist or you do not have access permissions. Please check correctness of table path and user permissions., code: 2003
```

В ряде сценариев такое поведение не требуется. Например, если мы хотим гарантированно создать новую таблицу с удалением предыдущий в рамках одного SQL-скрипта или последовательности SQL-команд. В таких случаях применяется инструкция `DROP TABLE IF EXIST <table name>`. В случае отсутствия таблицы инструкция вернет сообщение `DROP TABLE`, а не ошибку.

Примечание

В отличии от PostgreSQL, в YDB с используются оптимистические блокировки. Это значит, что транзакции проверяют условия выполнения необходимых блокировок в конце своей работы, а не в начале. Если за время выполнения транзакции блокировка была нарушена – такая транзакция завершится ошибкой `Transaction locks invalidated`. В этом случае можно попробовать выполнить аналогичную транзакцию снова.

DELETE FROM (удаление строк из таблицы)

Важно

Поддержка синтаксиса PostgreSQL в YDB **находится в разработке**. Использовать её в production окружениях **не рекомендуется**. Запросы в PostgreSQL могут исполняться до нескольких раз медленнее по сравнению с аналогичными запросами на YQL.

Основной сценарий, который можно тестировать — выполнение аналитических запросов к хранимым в YDB данным.

Синтаксис инструкции `DELETE FROM`:

```
DELETE FROM <table name>
WHERE <column name><condition><value/range>;
```

Для удаления строки из таблицы по конкретному значению столбца используется конструкция `DELETE FROM <table name> WHERE <column name><condition><value/range>`.

Важно

Обратите внимание, что использование оператора `WHERE ...` опционально, поэтому при работе с `DELETE FROM` очень важно случайно не выполнить команду раньше указания оператора `WHERE ...`.

Примечание

В отличии от PostgreSQL, в YDB с используются оптимистические блокировки. Это значит, что транзакции проверяют условия выполнения необходимых блокировок в конце своей работы, а не в начале. Если за время выполнения транзакции блокировка была нарушена – такая транзакция завершится ошибкой `Transaction locks invalidated`. В этом случае можно попробовать выполнить аналогичную транзакцию снова.

INSERT INTO (вставка строк в таблицу)

Важно

Поддержка синтаксиса PostgreSQL в YDB **находится в разработке**. Использовать её в production окружениях **не рекомендуется**. Запросы в PostgreSQL могут исполняться до нескольких раз медленнее по сравнению с аналогичными запросами на YQL.

Основной сценарий, который можно тестировать — выполнение аналитических запросов к хранимым в YDB данным.

Синтаксис инструкции `INSERT INTO`:

```
INSERT INTO <table name> (<column name>, ...)
VALUES (<value>);
```

Инструкция `INSERT INTO` предназначена для добавления строк в таблицу. Она может добавить одну или несколько строк за одно исполнение. Пример вставки одной строки в таблицу "people":

```
INSERT INTO people (name, lastname, age, country, state, city, birthday, sex)
VALUES ('John', 'Doe', 30, 'USA', 'California', 'Los Angeles', CAST('1992-01-15' AS Date), 'Male');
```

В этой записи мы не указали столбец `id` и не задали ему значение – это сделано намеренно, так как в таблице "people" у столбца "id" задан тип данных `Serial`. При выполнении инструкции `INSERT INTO` значение столбца "id" будет присвоено автоматически с учетом предыдущих значений – будет выполнен инкремент текущего значения "id".

Для множественной вставки строк в таблицу используется та же конструкция с перечислением групп данных для вставки через запятую:

```
INSERT INTO people (name, lastname, age, country, state, city, birthday, sex)
VALUES
    ('Jane', 'Smith', 25, 'Canada', 'Ontario', 'Toronto', CAST('1997-08-23' AS Date), 'Female'),
    ('Alice', 'Johnson', 28, 'UK', 'England', 'London', CAST('1994-05-05' AS Date), 'Female'),
    ('Bob', 'Brown', 40, 'USA', 'Texas', 'Dallas', CAST('1982-12-10' AS Date), 'Male'),
    ('Charlie', 'Davis', 35, 'Canada', 'Quebec', 'Montreal', CAST('1987-02-17' AS Date), 'Male'),
    ('Eve', 'Martin', 29, 'UK', 'Scotland', 'Edinburgh', CAST('1993-11-21' AS Date), 'Female'),
    ('Frank', 'White', 45, 'USA', 'Florida', 'Miami', CAST('1977-03-14' AS Date), 'Male'),
    ('Grace', 'Clark', 50, 'Canada', 'British Columbia', 'Vancouver', CAST('1972-04-26' AS Date), 'Female'),
    ('Hank', 'Miller', 33, 'UK', 'Wales', 'Cardiff', CAST('1989-07-30' AS Date), 'Male'),
    ('Ivy', 'Garcia', 31, 'USA', 'Arizona', 'Phoenix', CAST('1991-09-05' AS Date), 'Female'),
    ('Jack', 'Anderson', 22, 'Canada', 'Manitoba', 'Winnipeg', CAST('2000-06-13' AS Date), 'Male'),
    ('Kara', 'Thompson', 19, 'UK', 'Northern Ireland', 'Belfast', CAST('2003-10-18' AS Date), 'Female'),
    ('Liam', 'Martinez', 55, 'USA', 'New York', 'New York City', CAST('1967-01-29' AS Date), 'Male'),
    ('Molly', 'Robinson', 40, 'Canada', 'Alberta', 'Calgary', CAST('1982-12-01' AS Date), 'Female'),
    ('Noah', 'Lee', 47, 'UK', 'England', 'Liverpool', CAST('1975-05-20' AS Date), 'Male'),
    ('Olivia', 'Gonzalez', 38, 'USA', 'Illinois', 'Chicago', CAST('1984-03-22' AS Date), 'Female'),
    ('Paul', 'Harris', 23, 'Canada', 'Saskatchewan', 'Saskatoon', CAST('1999-08-19' AS Date), 'Male'),
    ('Quinn', 'Lewis', 34, 'UK', 'England', 'Manchester', CAST('1988-07-25' AS DATE), 'Female'),
    ('Rachel', 'Young', 42, 'USA', 'Ohio', 'Cleveland', CAST('1980-02-03' AS Date), 'Female');
```

В обоих примерах для указания даты выпуска фильма, мы использовали функцию `CAST()`, которая используется для преобразования одного типа данных в другой. В данном случае мы с помощью ключевого слова `AS` и типа данных `Date` явно указали, что хотим преобразовать строковое представление даты в формате ISO8601.

Указать нужный тип данных, например, `DATE` можно и альтернативным способом с помощью оператора приведения типов `::`, который используется для явного приведения значения одного типа данных к другому. Пример использования оператора `::` может выглядеть так:

```
INSERT INTO people (name, lastname, age, country, state, city, birthday, sex)
VALUES ('Sam', 'Walker', 60, 'Canada', 'Nova Scotia', 'Halifax', '1962-04-15'::Date, 'Male');
```

Примечание

В отличии от PostgreSQL, в YDB с используются оптимистические блокировки. Это значит, что транзакции проверяют условия выполнения необходимых блокировок в конце своей работы, а не в начале. Если за время выполнения транзакции блокировка была нарушена – такая транзакция завершится ошибкой `Transaction locks invalidated`. В этом случае можно попробовать выполнить аналогичную транзакцию снова.

SELECT (Чтение строк из таблицы)

> **Важно**
>
> Поддержка синтаксиса PostgreSQL в YDB **находится в разработке**. Использовать её в production окружениях **не рекомендуется**. Запросы в PostgreSQL могут исполняться до нескольких раз медленнее по сравнению с аналогичными запросами на YQL.
>
> Основной сценарий, который можно тестировать — выполнение аналитических запросов к хранимым в YDB данным.

Синтаксис инструкции `SELECT`:

```
SELECT [<table column>, ... | *]
FROM [<table name> | <sub query>] AS <table name alias>
LEFT | RIGHT | CROSS | INNER JOIN <another table> AS <table name alias> ON <join condition>
WHERE <condition>
GROUP BY <table column>
HAVING <condition>
UNION | UNION ALL | EXCEPT | INTERSECT
ORDER BY <table column> [ASC | DESC]
LIMIT [<limit value>]
OFFSET <offset number>
```

Вызов SELECT без указания целевой таблицы

`SELECT` используется для возврата вычислений на клиентскую сторону, в случае если он вызван без дополнительных конструкций, так как `FROM ...`, `INSERT INTO ...` и т.д. Например, `SELECT` можно использовать для работы с датами, преобразования чисел или подсчета длины строки:

```
SELECT CURRENT_DATE + INTERVAL '1 day';  -- Возвращает завтрашнюю дату
SELECT LENGTH('Hello');  -- Возвращает длину строки 'Hello'
SELECT CAST('123' AS INTEGER);  -- Преобразует строки в числа
```

Такое применение `SELECT` бывает полезно при тестировании, отладки выражений или SQL-функций без обращения к реальной таблице, но чаще `SELECT` используется для получения строк из одной или множества таблиц.

Выборка значений из одного или нескольких столбцов

Для возвращения значений из одного или нескольких столбцов таблицы применяется `SELECT` в следующем виде:

```
SELECT <column name> , <column name>
FROM <table name>;
```

Чтобы прочитать все данные из таблицы, например, таблицы `people` – нужно выполнить команду `SELECT * FROM people;`, где `*` – это оператор выбора данных по всем столбцам. При такой записи будут возвращены все строки из таблицы с данными по всем столбцам.

Вывести столбцы "id", "name" и "lastname" для всех строк таблицы `people` можно так:

```
SELECT id, name, lastname
FROM people;
```

Ограничение получаемых результатов выборки с помощью WHERE

Для выборки только части строк - используется оператор `WHERE` с условиями выборки: `WHERE <column name> <condition> <column value>;`:

```
SELECT id, name, lastname
FROM people
WHERE age > 30;
```

`WHERE` позволяет использовать несколько операторов условного выбора (`AND` (И), `OR` (ИЛИ)) для создания сложных условий выборок, например, диапазонов:

```
SELECT id, name, lastname
FROM people
WHERE age > 30 AND age < 45;
```

Получение части строк по условиям LIMIT и OFFSET

Для ограничения количества строк в результатах выборки используется `LIMIT` с указанием количества строк:

```
SELECT id, name, lastname
FROM people
WHERE age > 30 AND age < 45
LIMIT 5;
```

Так на печать будет выведено 5 первых строк из выборки. С `OFFSET` можно указать сколько нужно пропустить строк, прежде чем начать выдавать строки на печать:

```
SELECT id, name, lastname
FROM people
WHERE age > 30 AND age < 45
OFFSET 3
LIMIT 5;
```

При указании `OFFSET 3` первые 3 строки результирующей выборки из таблицы `people` будут пропущены.

Сортировка результатов выборки с помощью ORDER BY

По умолчанию база данных не гарантирует порядок возврата строк, и он может отличаться от запроса к запросу. Если требуется сохранить определенный порядок строк – используется инструкция `ORDER BY` с указанием столбца для сортировки и направлением сортировки:

```
SELECT id, name, lastname, age
FROM people
WHERE age > 30 AND age < 45
ORDER BY age DESC;
```

Сортировка происходит по результатам, которые возвращает (`SELECT`), а не по исходным столбцам таблицы (`FROM`). Сортировать можно в прямом порядке – `ASC` (от меньшего к большему - вариант по умолчанию, можно не указывать) и в обратном – `DESC` (от большего к меньшему). Как сортировка будет выполняться, зависит от типа данных столбца. Например, строки хранятся в utf-8 и сравниваются по "unicode collate" (по кодам символов).

Группировка результатов выборки из одной или нескольких таблиц с помощью GROUP BY

`GROUP BY` используется для сбора данных по нескольким записям и группировки результатов по одному или нескольким столбцам. Синтаксис использования `GROUP BY` :

```
SELECT <column name>, <column name>, ...
FROM <table name>
[WHERE <column name> = <value>]
GROUP BY <column name>, <column name>, ...;
[HAVING <column name> = <limit column value>]
[LIMIT <value>]
[OFFSET <value>]
```

Пример группировки данных из таблицы "people" по полу ("sex") и возрасту ("age") с ограничением выборки (`WHERE`) по возрасту:

```
SELECT sex, age
FROM people
WHERE age > 40
GROUP BY sex, age;
```

В предыдущем примере мы использовали `WHERE` – необязательный параметр фильтрации результата, который фильтрует отдельные строки до применения `GROUP BY` . В следующем примере мы используем `HAVING` для исключения из результата строки групп, не удовлетворяющих условию. `HAVING` фильтрует строки групп, созданных `GROUP BY` . При использовании `HAVING` запрос превращается в группируемый, даже если `GROUP BY` отсутствует. Все выбранные строки считаются формирующими одну группу, а в списке SELECT и предложении HAVING можно обращаться к столбцам таблицы только из агрегатных функций. Такой запрос будет выдавать единственную строку, если результат условия HAVING — true, и ноль строк в противном случае.

Примеры использования HAVING

`HAVING` + `GROUP BY`	`HAVING` + `WHERE` + `GROUP BY`

```sql
SELECT sex, country, age
FROM people
GROUP BY sex, country, age
HAVING sex = 'Female';
```

```sql
SELECT sex, name,age
FROM people
WHERE age > 40
GROUP BY sex,name,age
HAVING sex = 'Female';
```

`SELECT` можно применять к нескольким таблицам с указанием типа соединения таблиц. Объединение таблиц задается через оператор `JOIN`, который бывает пяти типов: `LEFT JOIN`, `RIGHT JOIN`, `INNER JOIN`, `CROSS JOIN`, `FULL JOIN`. Когда выполняется `JOIN` по определенному условию, например, по ключу, и в одной из таблиц есть несколько строк с одинаковым значением этого ключа, получается декартово произведение. Это означает, что каждая строка из одной таблицы будет соединена с каждой соответствующей строкой из другой таблицы.

Объединение таблиц с помощью LEFT JOIN, RIGHT JOIN или INNER JOIN

Синтаксис `SELECT` с использованием `LEFT JOIN`, `RIGHT JOIN`, `INNER JOIN`, `FULL JOIN` одинаков:

```sql
SELECT <table name left>.<column name>, ... ,
FROM <table name left>
LEFT | RIGHT | INNER | FULL JOIN <table name right> AS <table name right alias>
ON <table name left>.<column name> = <table name right>.<column name>;
```

Все операторы `JOIN`, кроме `CROSS JOIN` использую для присоединения таблиц ключевое слово `ON`. В случае `CROSS JOIN`, синтаксис его использования будет следующем: `CROSS JOIN <table name> AS <table name alias>;`. Рассмотрим пример использования каждого оператора `JOIN` в отдельности.

LEFT JOIN (или LEFT OUTER JOIN)

Возвращает все строки из левой таблицы и соответствующие строки из правой таблицы. Если нет совпадений, возвращает `NULL` (в выводе будет пустота) для всех колонок правой таблицы.

Пример использования `LEFT JOIN`:

```sql
SELECT people.name, people.lastname, card.social_card_number
FROM people
LEFT JOIN social_card AS card
ON people.name = card.card_holder_name AND people.lastname = card.card_holder_lastname;
```

Результат выполнения SQL запроса с использованием `LEFT JOIN` без одной записи в правой таблице `social_card`:

```
 name    | lastname | social_card_number
---------+----------+--------------------
 John    | Doe      |          123456789
 Jane    | Smith    |          223456789
 Alice   | Johnson  |          323456789
 Bob     | Brown    |          423456789
 Charlie | Davis    |          523456789
 Eve     | Martin   |          623456789
 Frank   | White    |
```

RIGHT JOIN (или RIGHT OUTER JOIN)

Возвращает все строки из правой таблицы и соответствующие строки из левой таблицы. Если не существует совпадений, возвращает `NULL` для всех колонок левой таблицы. Этот тип `JOIN` редко используется, так как его функциональность можно заменить `LEFT JOIN`, меняя местами таблицы. Пример использования `RIGHT JOIN`:

```sql
SELECT people.name, people.lastname, card.social_card_number
FROM people
RIGHT JOIN social_card AS card
ON people.name = card.card_holder_name AND people.lastname = card.card_holder_lastname;
```

Результат выполнения SQL запроса с использованием `RIGHT JOIN` без одной записи в левой таблице `people`:

```
 name   | lastname | social_card_number
--------+----------+--------------------
John    | Doe      |          123456789
Jane    | Smith    |          223456789
Alice   | Johnson  |          323456789
Bob     | Brown    |          423456789
Charlie | Davis    |          523456789
Eve     | Martin   |          623456789
        |          |          723456789
```

INNER JOIN (или просто JOIN)

Возвращает строки, когда есть соответствующие значения в обеих таблицах. Исключает из результатов те строки, для которых нет совпадений в соединяемых таблицах. Пример использования `INNER JOIN`:

```
SELECT people.name, people.lastname, card.social_card_number
FROM people
RIGHT JOIN social_card AS card
ON people.name = card.card_holder_name AND people.lastname = card.card_holder_lastname;
```

Такой SQL запрос вернет только те строки, для которых есть совпадения в обеих таблицах:

```
 name   | lastname | social_card_number
--------+----------+--------------------
John    | Doe      |          123456789
Jane    | Smith    |          223456789
Alice   | Johnson  |          323456789
Bob     | Brown    |          423456789
Charlie | Davis    |          523456789
Eve     | Martin   |          623456789
```

CROSS JOIN

Возвращает комбинированный результат каждой строки левой таблицы с каждой строкой правой таблицы. Обычно `CROSS JOIN` используется, когда необходимо получить все возможные комбинации строк из двух таблиц. `CROSS JOIN` просто комбинирует каждую строку одной таблицы с каждой строкой другой без какого-либо условия, поэтому в его синтаксисе отсутствуют ключевые слова `ON` или: `CROSS JOIN <table name> AS <table name alias>;`.

Пример использования `CROSS JOIN` с ограничением вывода результата `LIMIT 5`:

```
SELECT people.name, people.lastname, card.social_card_number
FROM people
CROSS JOIN social_card AS card
LIMIT 5;
```

Пример выше вернет все возможные комбинации столбцов, участвующих в выборке из двух таблиц:

```
name | lastname | social_card_number
------+----------+--------------------
 John | Doe      |          123456789
 John | Doe      |          223456789
 John | Doe      |          323456789
 John | Doe      |          423456789
 John | Doe      |          523456789
```

FULL JOIN (или FULL OUTER JOIN)

Возвращает как совпавшие, так и не совпавшие строки в обеих таблицах, при этом возвращает `NULL` в колонках из таблицы, для которой не найдено совпадение. Пример выполнения SQL запроса с использованием `FULL JOIN`:

```
SELECT people.name, people.lastname, card.social_card_number
FROM people
FULL JOIN social_card AS card
ON people.name = card.card_holder_name AND people.lastname = card.card_holder_lastname;
```

В результате выполнения SQL запроса будет возвращен следующей вывод:

```
 name   | lastname | social_card_number
--------+----------+--------------------
 Liam    | Martinez |         1323456789
 Eve     | Martin   |          623456789
 Hank    | Miller   |          923456789
 Molly   | Robinson |         1423456789
 Sam     | Walker   |
 Paul    | Harris   |         1723456789
 Kara    | Thompson |         1223456789
         |          |         1923456789
...
```

UPDATE (изменения строк таблицы)

> **Важно**
>
> Поддержка синтаксиса PostgreSQL в YDB **находится в разработке**. Использовать её в production окружениях **не рекомендуется**. Запросы в PostgreSQL могут исполняться до нескольких раз медленнее по сравнению с аналогичными запросами на YQL.
>
> Основной сценарий, который можно тестировать — выполнение аналитических запросов к хранимым в YDB данным.

Синтаксис инструкции `UPDATE`:

```
UPDATE <table name>
SET <column name> = [<new value>, CASE ... END]
WHERE <search column name> = [<search value>, IN]
```

Конструкция `UPDATE ... SET ... WHERE` работает так:

1. **Задаётся название таблицы** – `UPDATE`` `, в которой будет произведено обновление данных;
2. **Указывается название столбца** – `SET` `<column name>`, где следует обновить данные. Может применяться конструкция для указания набора данных `CASE ... END`;
3. **Задаётся новое значение** – `<new value>`;
4. **Указываются критерии поиска** – `WHERE` с указанием колонки для поиска `<search column name>` и значения, которому должен соответствовать критерий поиска . Если применяется `CASE`, тогда указывается оператор `IN` с перечислением значений .

Обновление одной строки в таблице с условиями

Обновление без условий	Обновление с условиями

Обновление с применением выражений	Обновление с применением функций

```
UPDATE people
SET age = age + 1
WHERE country = 'Canada';
```

```
UPDATE people
SET name = UPPER(name)
WHERE country = 'USA';
```

```
            WHEN name = 'John' THEN 32
            WHEN name = 'Jane' THEN 26
        END
WHERE name IN ('John', 'Jane');
```

Примечание

В отличии от PostgreSQL, в YDB с используются оптимистические блокировки. Это значит, что транзакции проверяют условия выполнения необходимых блокировок в конце своей работы, а не в начале. Если за время выполнения транзакции блокировка была нарушена – такая транзакция завершится ошибкой `Transaction locks invalidated`. В этом случае можно попробовать выполнить аналогичную транзакцию снова.

BEGIN, COMMIT, ROLLBACK (работа с транзакциями)

Важно

Поддержка синтаксиса PostgreSQL в YDB **находится в разработке**. Использовать её в production окружениях **не рекомендуется**. Запросы в PostgreSQL могут исполняться до нескольких раз медленнее по сравнению с аналогичными запросами на YQL.

Основной сценарий, который можно тестировать — выполнение аналитических запросов к хранимым в YDB данным.

Транзакции – это способ группировки одной или нескольких операций с базой данных в единую единицу работы. Транзакция может состоять из одной или нескольких SQL-операций и использоваться для обеспечения согласованности данных. Транзакция обеспечивает гарантированное выполнение либо всех входящих в неё SQL-операций, либо ни одной. Транзакции управляются командами `BEGIN`, `COMMIT`, `ROLLBACK`.

Транзакции выполняются внутри сессий. Сессия — это одно соединение с базой данных, которое начинается при подключении к базе данных и завершается при её отключении. Транзакция начинается с команды `BEGIN` и завершается командой `COMMIT` (успешное завершение) или `ROLLBACK` (откат). Указывать `BEGIN`, `COMMIT` и `ROLLBACK` не обязательно, часто их использование подразумевается неявно. В случае если сессия неожиданно прерывается, тогда все транзакции, которые были начаты в текущей сессии – автоматически откатываются.

Рассмотрим каждую из команд:

- `BEGIN` – инициирует новую транзакцию. После выполнения этой команды все последующие операции с базой данных будут выполняться в рамках этой транзакции.
- `COMMIT` – завершает текущую транзакцию, применяя все её операции. Если все операции в транзакции были успешными, результаты этих операций фиксируются (становятся постоянными). Изменения становятся видны последующим транзакциям.
- `ROLLBACK` – откатывает текущую транзакцию, отменяя все её операции, если в процессе выполнения транзакции возникли ошибки или отмена транзакции производится приложением исходя из внутренней логики работы. Когда вызывается `ROLLBACK`, только изменения, сделанные в рамках текущей транзакции отменяются. Изменения, сделанные другими транзакциями (даже если они были запущены и завершены во время выполнения текущей транзакции), остаются нетронутыми. В случае если в процессе выполнения транзакции произошла ошибка, то дальнейшая работа с такой транзакцией становится невозможна – нужно делать `ROLLBACK`, так как выполнение `COMMIT` вернет ошибку. Если произойдет разрыв сессии во время активной транзакции – автоматически будет выполнен `ROLLBACK`. Более подробную информацию про управление конкурентным доступом (MVCC) можно найти в этой статье.

Предположим, нужно внести изменения в разные строки таблицы для разных столбцов так, чтобы транзакция была объединена в одну рабочую единицу и имела гарантии ACID. Такая запись может выглядеть так:

```
--Начало транзакции
BEGIN;

-- Инструкция обновления данных
UPDATE people
SET name = 'Ivan'
WHERE id = 1;

-- Инструкция обновления данных
UPDATE people
SET lastname = 'Gray'
WHERE id = 10;
```

Если все данные верны – нужно выполнить инструкцию подтверждения транзакции:

```
COMMIT;
```

Примечание

В отличии от PostgreSQL, в YDB с используются оптимистические блокировки. Это значит, что транзакции проверяют условия выполнения необходимых блокировок в конце своей работы, а не в начале. Если за время выполнения транзакции блокировка была нарушена – такая транзакция завершится ошибкой `Transaction locks invalidated`. В этом случае можно попробовать выполнить аналогичную транзакцию снова.

Блокировки

Блокировки используются, чтобы упорядочить одновременный (конкурентный) доступ нескольких процессов к одним и тем же ресурсам. Ресурсом может быть объект, с которым работает СУБД:

1. Списки первичных ключей, условия или даташард целиком (если блокировок становится очень много);
2. Структура в памяти, такая как хеш-таблица, буфер и т. п. (идентифицируется заранее присвоенным номером);
3. Абстрактные ресурсы, не имеющие никакого физического смысла (идентифицируются просто уникальным числом).

Чтобы процесс мог обратиться к ресурсу – он обязан захватить (acquire) блокировку, ассоциированную с этим ресурсом. Блокировка – это участок разделяемой памяти, в котором хранится информация о том свободна ли блокировка или захвачена. Блокировка сама по себе тоже является ресурсом, к которому возможен конкурентный доступ. Для доступа к блокировкам используются специальные примитивы синхронизации: семафоры или мьютексы. Смысл их в том, чтобы код, обращающийся к разделяемому ресурсу, одновременно выполнялся только в одном процессе. Если ресурс занят и блокировка не возможна – процесс завершится ошибкой.

Блокировки можно классифицировать по принципу их реализации:

- **Пессимистическая блокировка** до модификации данных накладывается блокировка на все потенциально затрагиваемые строки. Это исключает возможность их изменения другими сессиями до завершения текущей операции. После модификации гарантировано, что запись данных будет непротиворечивой;
- **Оптимистическая блокировка** не ограничивает доступ к данным в процессе работы, но использует специальный атрибут (например: `VERSION`) для контроля изменений. Атрибут - это поле из метаданных о строке, которое не видно пользователям, оно относится к деталям реализации механизма блокировок. Перед фиксацией изменений проверяется установленный атрибут. Если он не изменился – изменения зафиксируются (`COMMIT`), иначе транзакция будет откатана (`ROLLBACK`).

В совместимости YDB с PostgreSQL используются оптимистические блокировки – это значит, что транзакции в конце своей работы проверяют условия выполнения блокировок. Если за время выполнения транзакции блокировка была нарушена – такая транзакция завершится ошибкой:

```
Error: Transaction locks invalidated. Table: <table name>, code: 2001
```

С ошибкой могут завершаться транзакции, в которых исполняются SQL-инструкции чтения и записи. Транзакции, где исполняются только SQL-инструкции чтения или записи завершаются корректно. Приведём пример транзакции, которая завершится с ошибкой в случае внесения изменений в данные таблице параллельной транзакцией:

```
BEGIN;
SELECT * FROM people;
-- Если тут будет выполнен INSERT в другой транзакции, эта транзакция завершится ошибкой
UPDATE people SET age = 27
WHERE name = 'JOHN';
COMMIT;
```

В результате транзакция завершится ошибкой `Error: Transaction locks invalidated` и будет откачена (`ROLLBACK`). В случае ошибки `Error: Transaction locks invalidated` – можно попытаться выполнить транзакцию снова.

Использование встроенного web-интерфейса

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.

Страница перемещена в новое место размещения, обновите ссылки.

YDB Monitoring

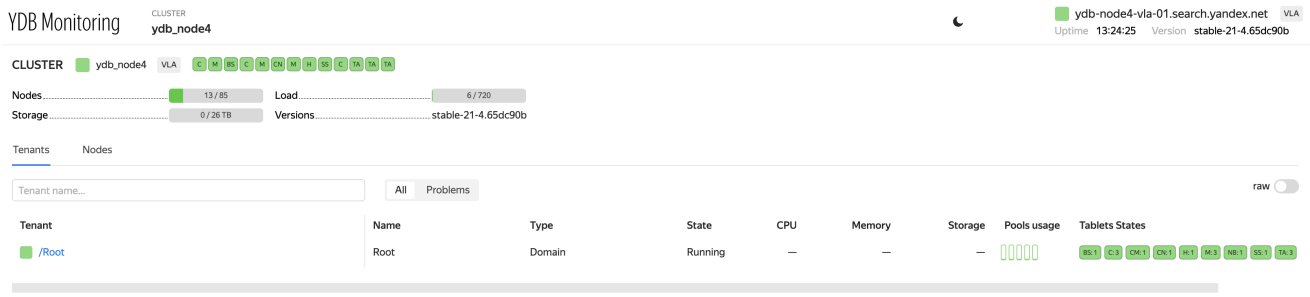
YDB Monitoring представлен набором web-страниц, отображающих состояние системы в разрезе по нескольким аспектам. На страницах приведены перечни компонент и их мгновенные параметры. Для многих компоненты приведен цветовой индикатор состояния перед названием.

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 Embedded 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 Embedded 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.

`Default log settings`.

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.

Графические пользовательские интерфейсы

Среда	Инструкция	Уровень поддержки
Встроенный 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
Jupyter Notebook	Инструкция	С помощью YDB-SQLAlchemy

Визуализация данных (Business Intelligence, BI)

Среда	Уровень поддержки	Инструкция
Apache Superset	Через PostgreSQL-совместимость	Инструкция
DataLens	Полный	Инструкция
FineBI	Через PostgreSQL-совместимость	Инструкция
Grafana	Полный	Инструкция

Оркестрация

Среда	Инструкция
Apache Airflow™	Инструкция

Поставка данных

Система поставки	Инструкция
Apache Kafka API	Инструкция
Apache Kafka Connect	Инструкция
Произвольные JDBC-источники данных	Инструкция

Система поставки	Инструкция
Apache Kafka API	Инструкция
Apache Kafka Connect	Инструкция
Произвольные JDBC-источники данных	Инструкция

Миграции данных

Среда	Инструкция
goose	Инструкция
Liquibase	Инструкция
Flyway	Инструкция

Объектно-реляционное отображение (ORM)

Система	Инструкция
Hibernate	Инструкция
Spring Data JDBC	Инструкция
JOOQ	Инструкция
Dapper	Инструкция
Entity Framework	Инструкция

Векторный поиск

Инструмент	Инструкция
LangChain	Инструкция

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.

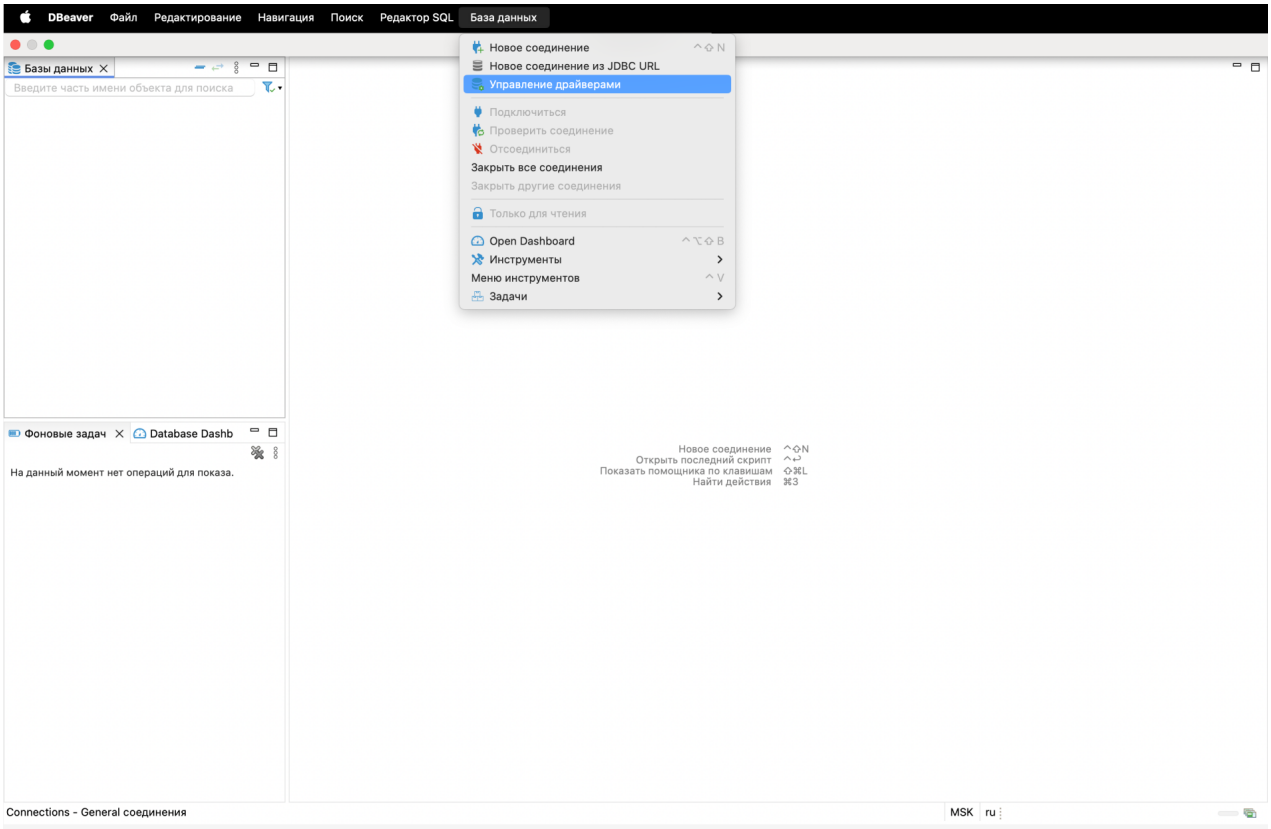
Подключение JDBC-драйвера YDB к DBeaver

Для подключения к YDB из DBeaver понадобится JDBC-драйвер. Для загрузки JDBC-драйвера выполните следующие шаги:

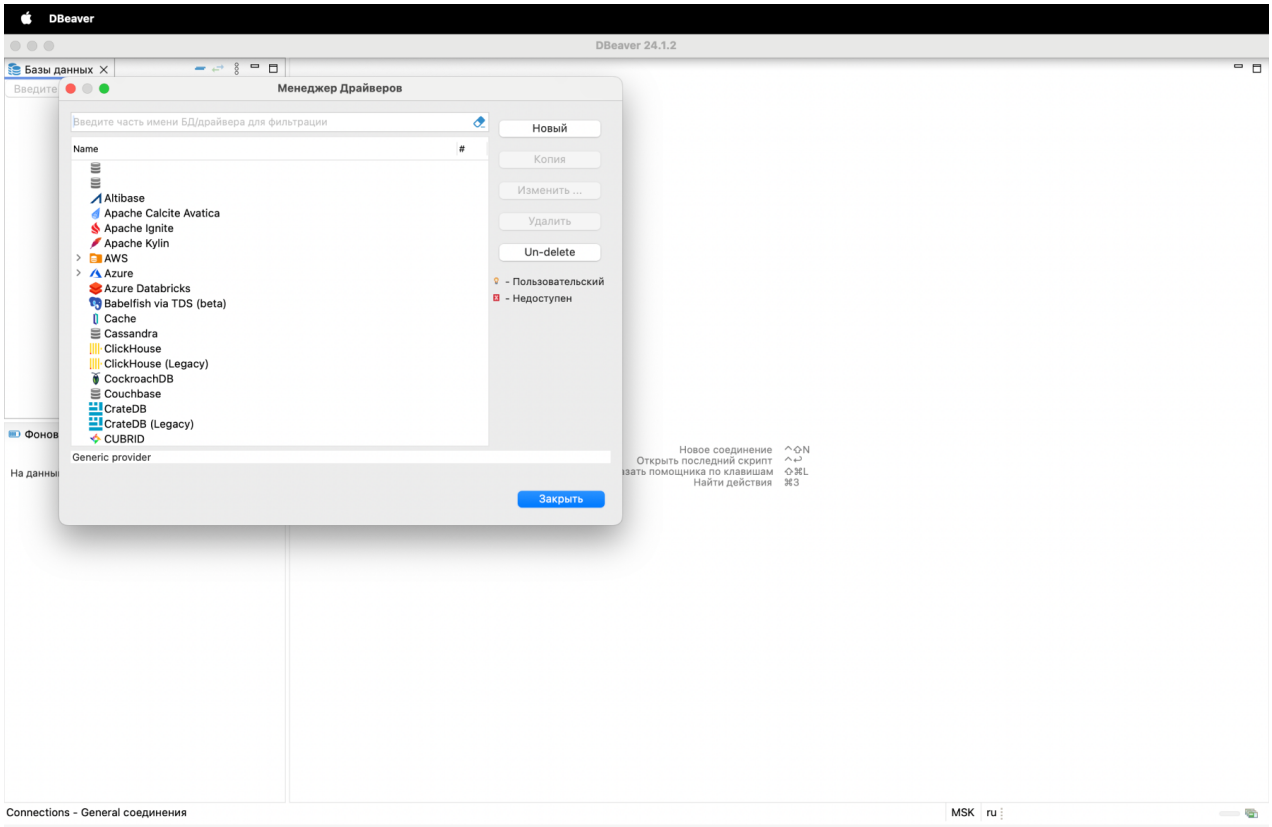
1. Перейдите в репозиторий ydb-jdbc-driver.
2. Выберите последний релиз (отмечен тегом `Latest`) и сохраните файл `ydb-jdbc-driver-shaded-<driver-version>.jar`.

Для подключения загруженного JDBC-драйвера выполните следующие шаги:

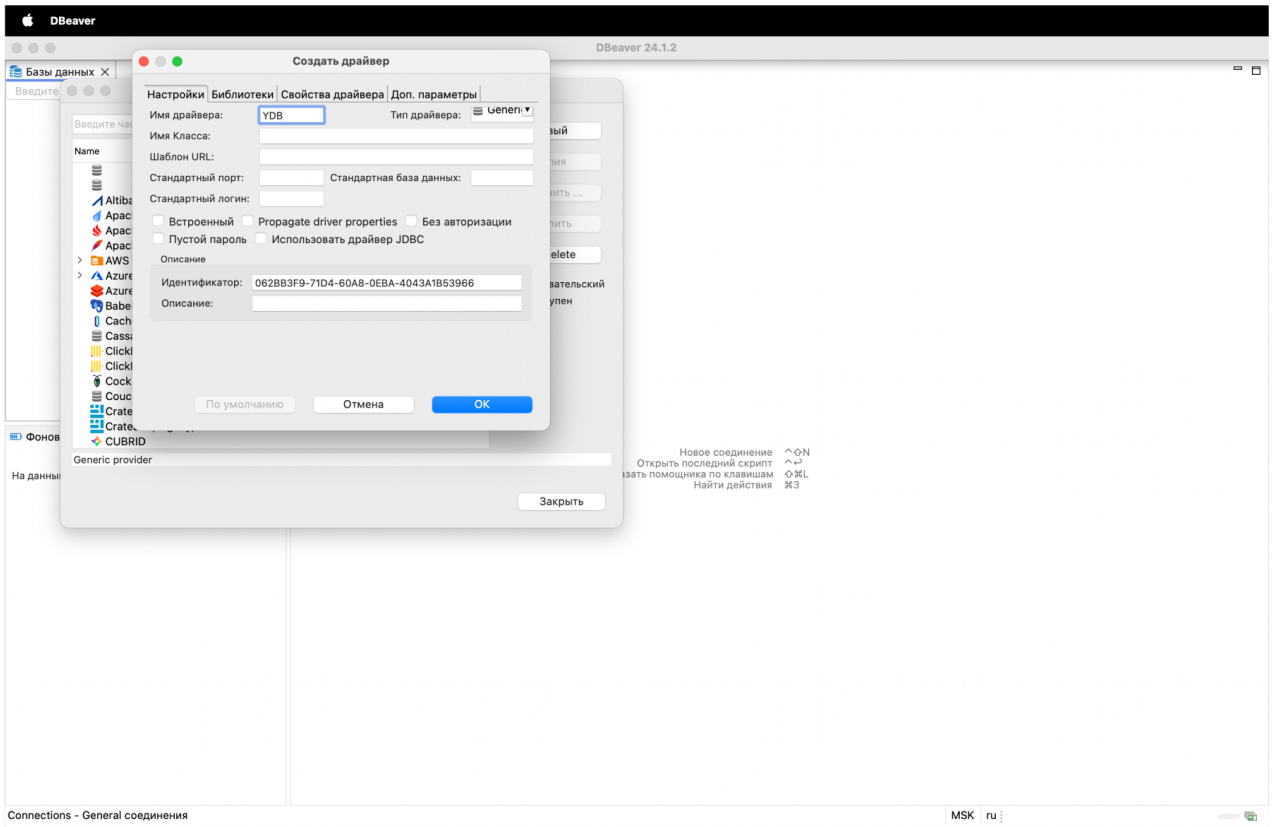
1. Выберите в верхнем меню DBeaver пункт **База данных**, а затем подпункт **Управление драйверами**:



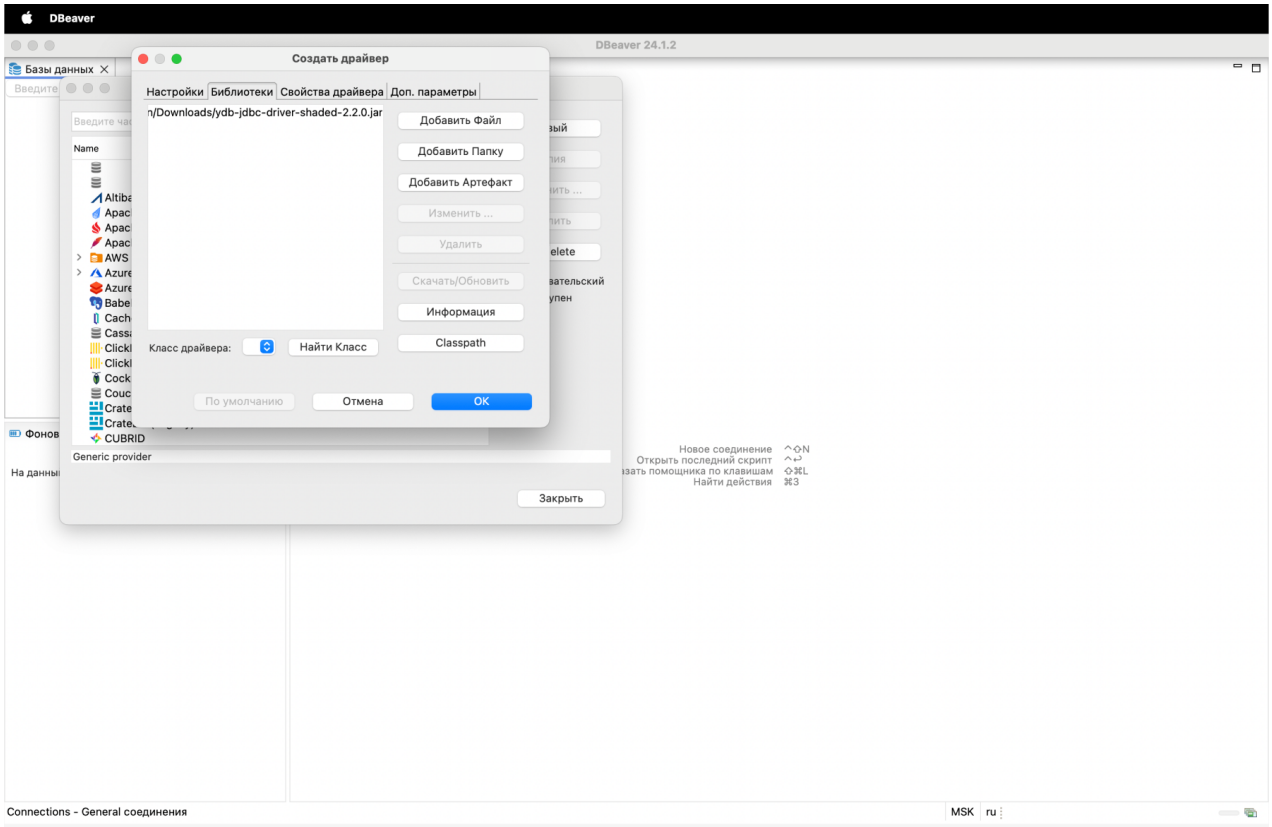
2. Чтобы создать новый драйвер, в открывшемся окне **Менеджер Драйверов** нажмите кнопку **Новый**:



3. В открывшемся окне **Создать драйвер**, в поле **Имя драйвера**, укажите `YDB`:

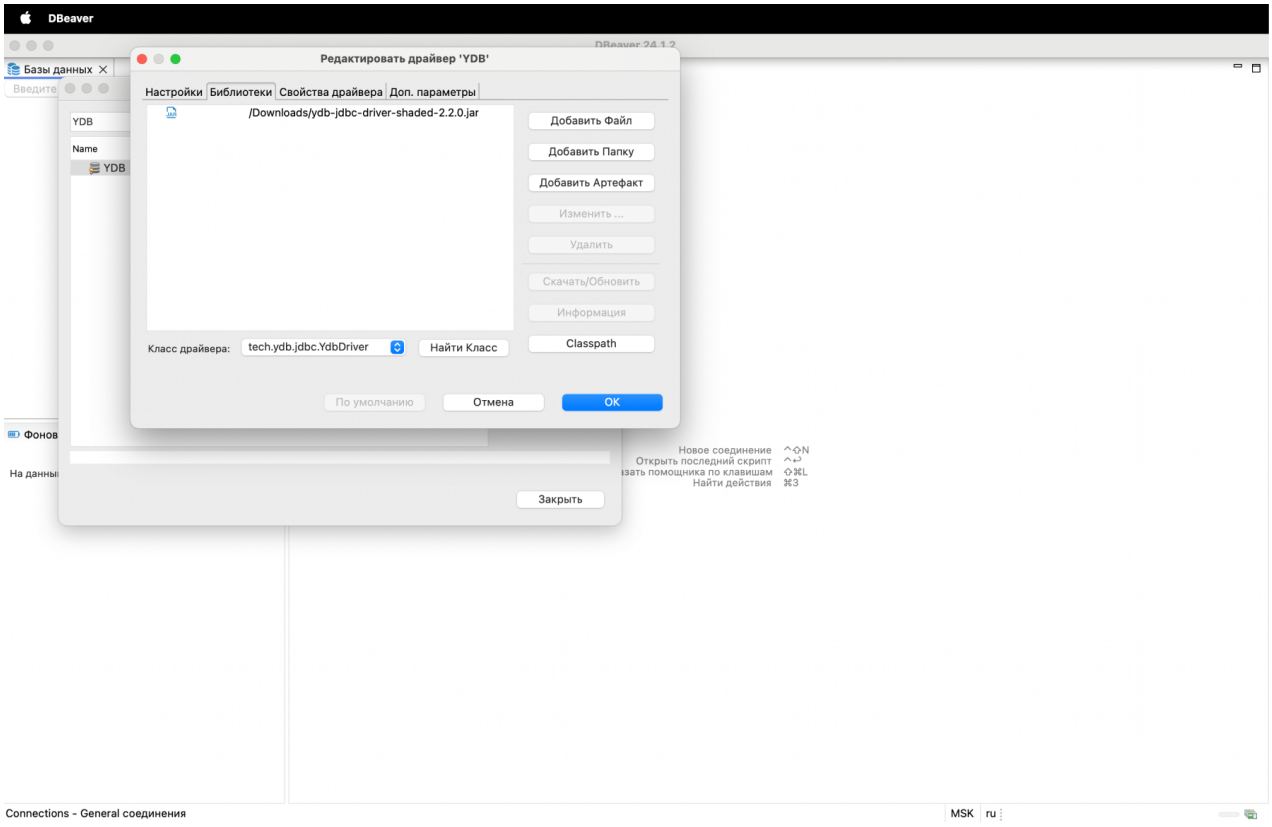


4. Перейдите в раздел **Библиотеки**, нажмите кнопку **Добавить файл**, укажите путь к скачанному ранее JDBC-драйверу YDB (файлу `ydb-jdbc-driver-shaded-<driver-version>.jar`) и нажмите кнопку **OK**:



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:

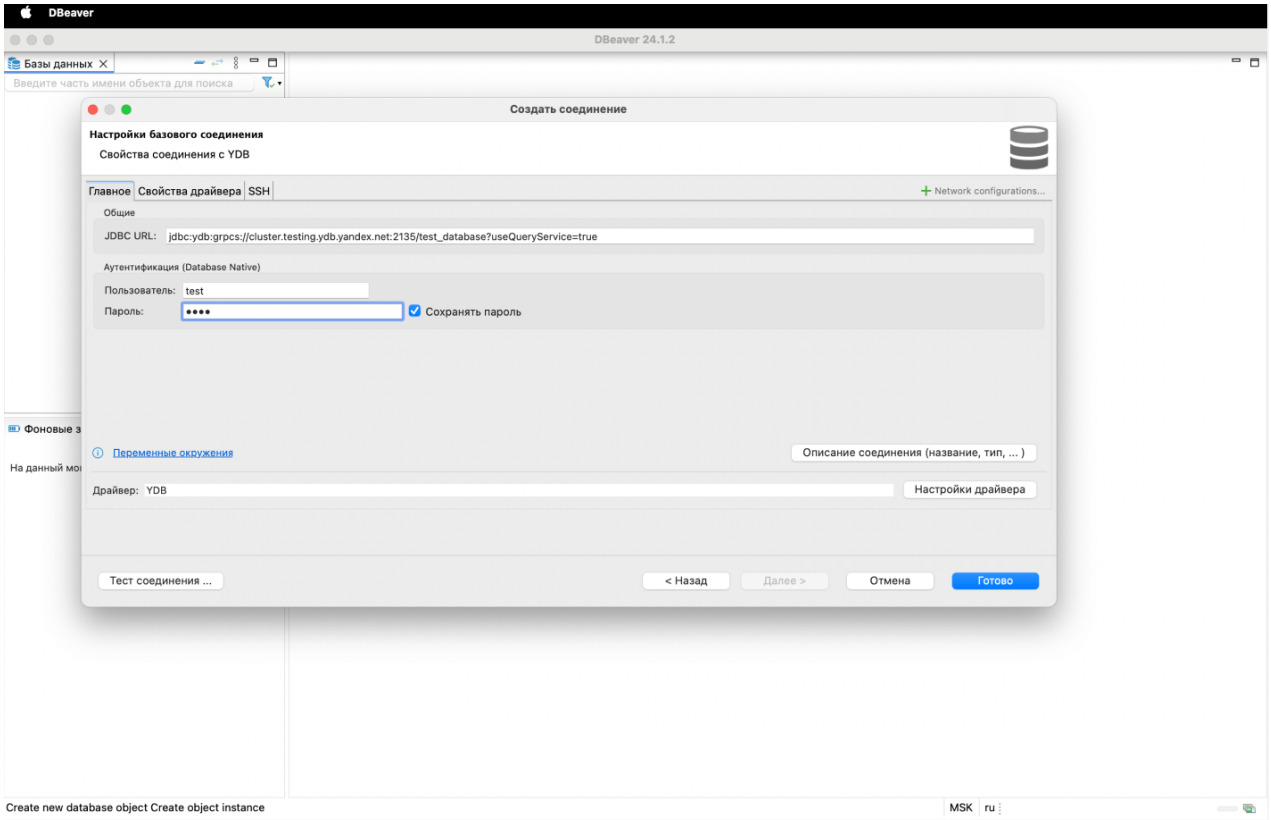
1. В DBeaver создайте новое соединение, указав тип соединения `YDB`.

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
```

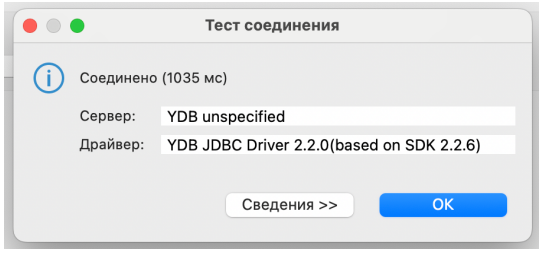
Где:

- `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.
5. Нажмите кнопку **Тест соединения ...** для проверки настроек.

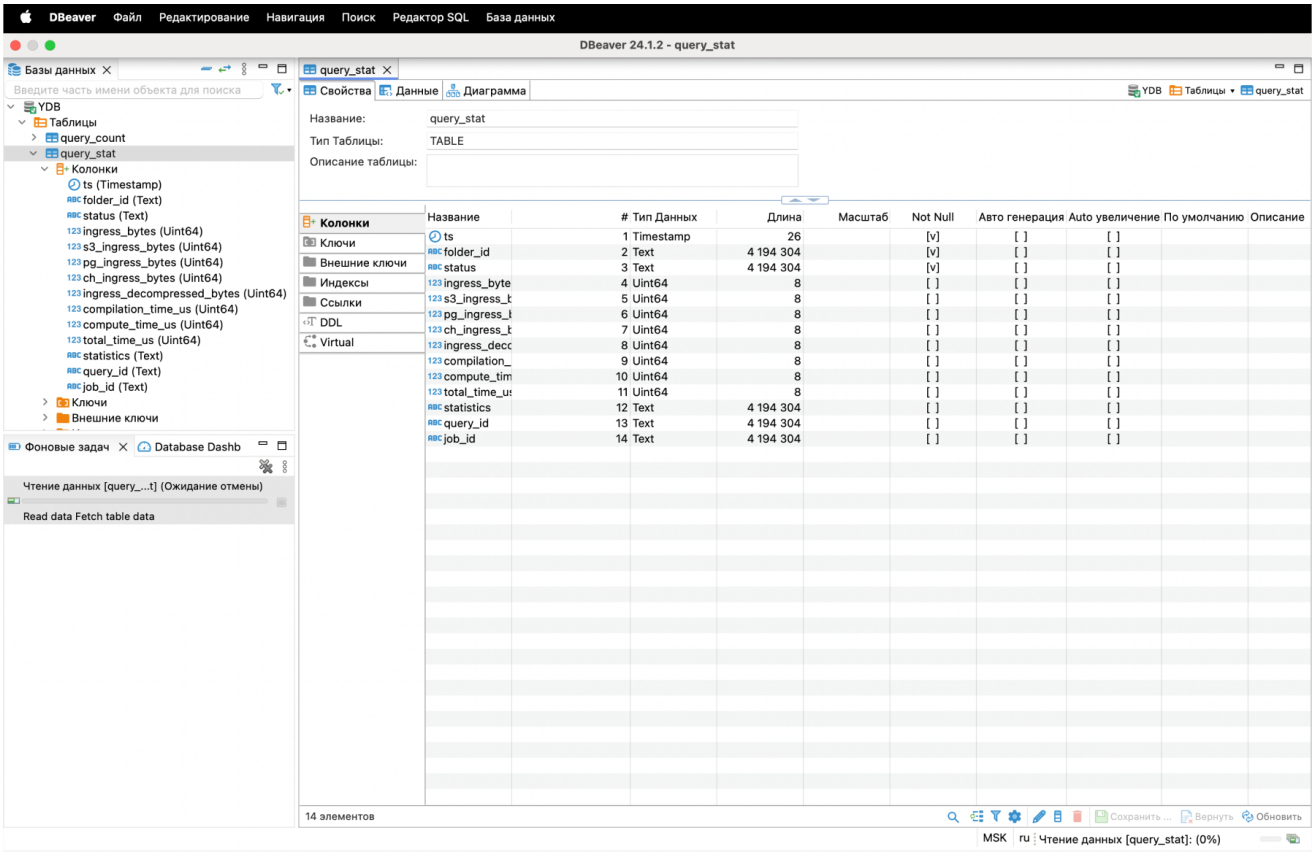
   Если все настройки указаны верно, то появится сообщение об успешном тестировании соединения:



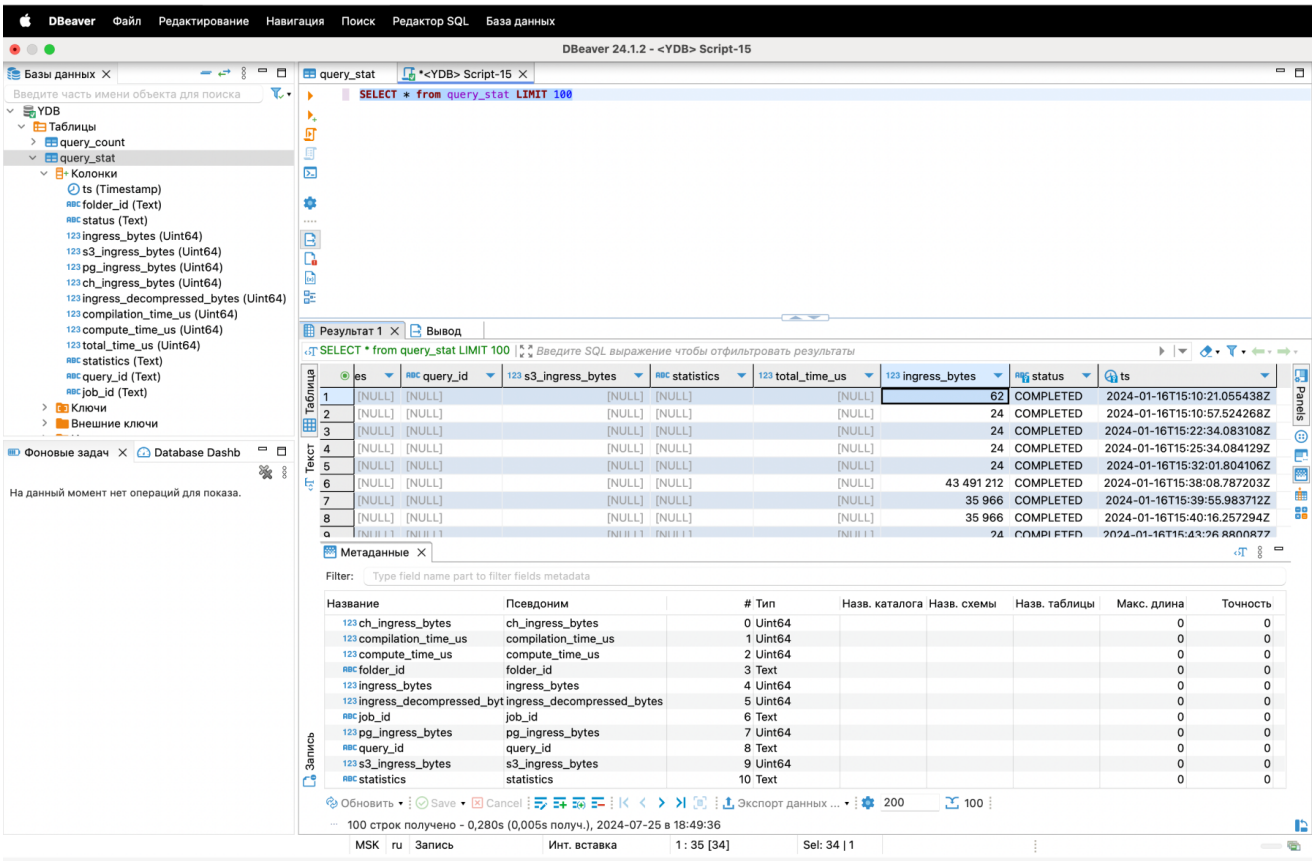
6. Нажмите кнопку **Готово** для сохранения соединения.

Работа с YDB

С помощью DBeaver можно просматривать список и структуру таблиц:



А также выполнять запросы к данным:



Подключение к YDB с помощью 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

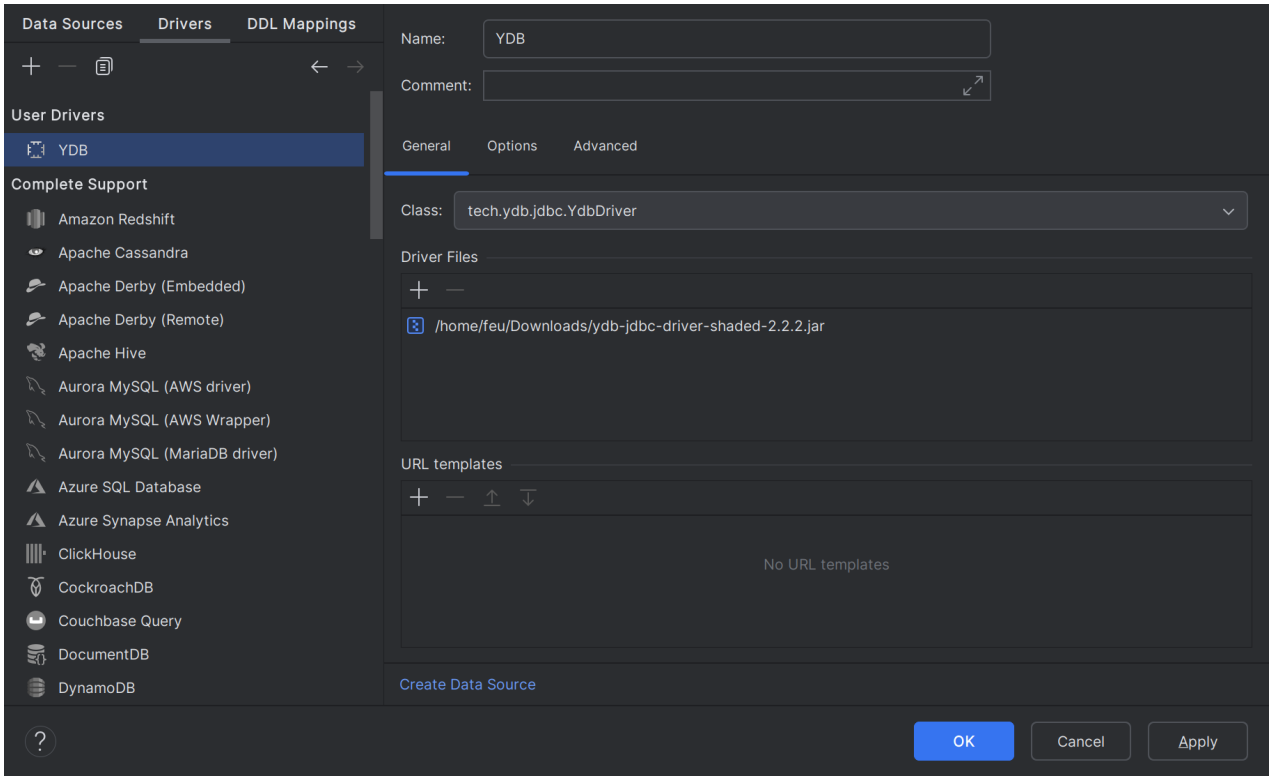
Для подключения к YDB из DataGrip понадобится JDBC-драйвер. Для загрузки JDBC-драйвера выполните следующие шаги:

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
   ```

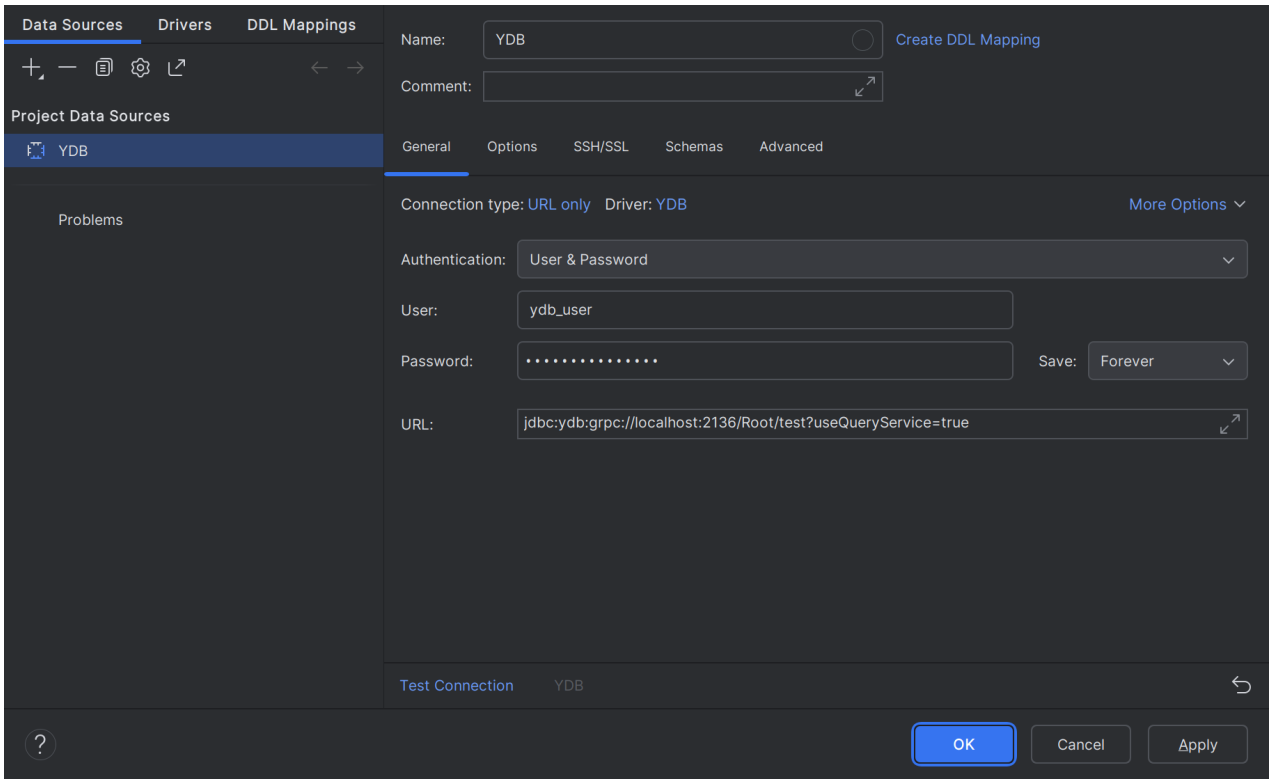
   Где:

   - `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.

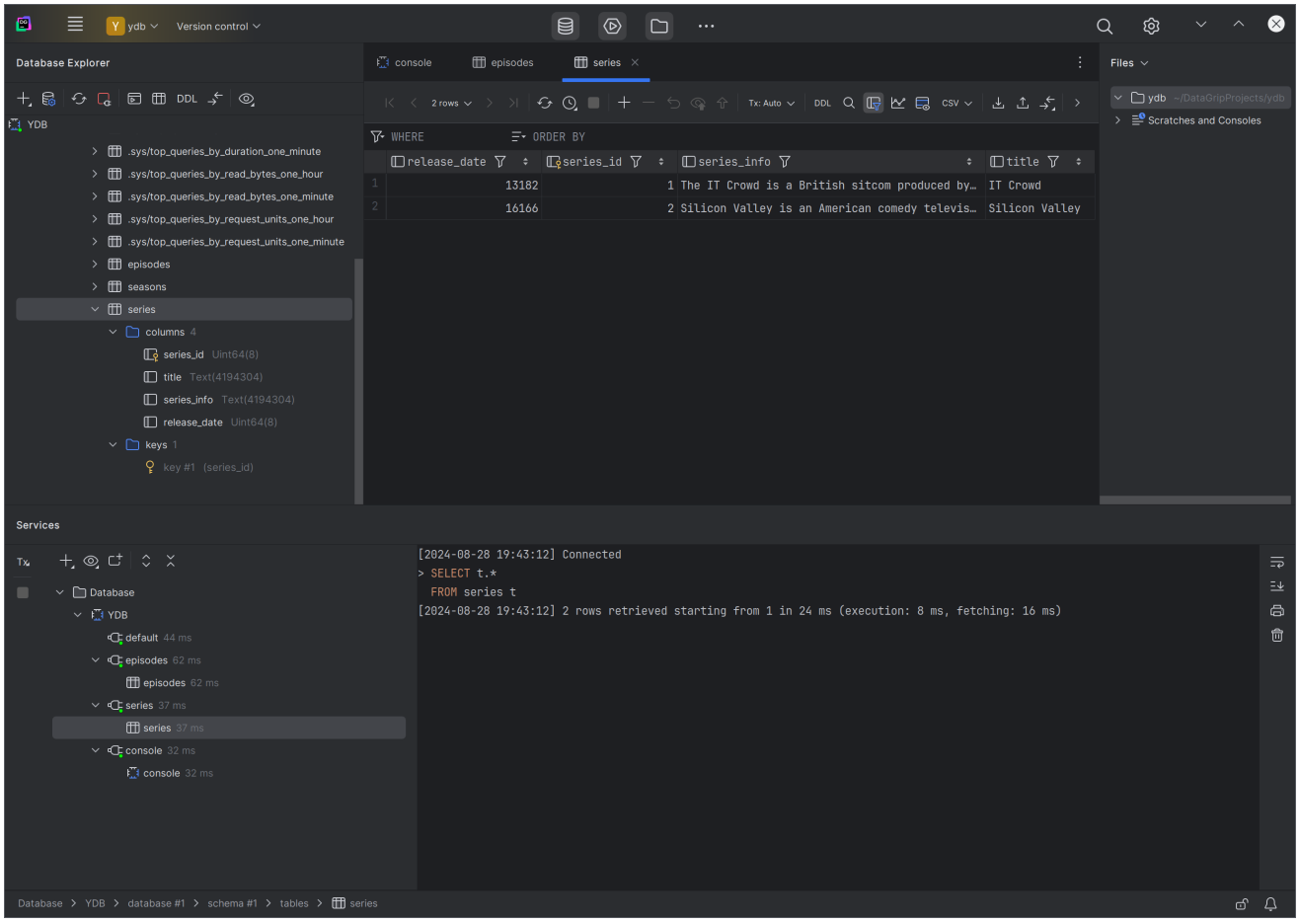
   Если все настройки указаны верно, то появится сообщение об успешном тестировании соединения.



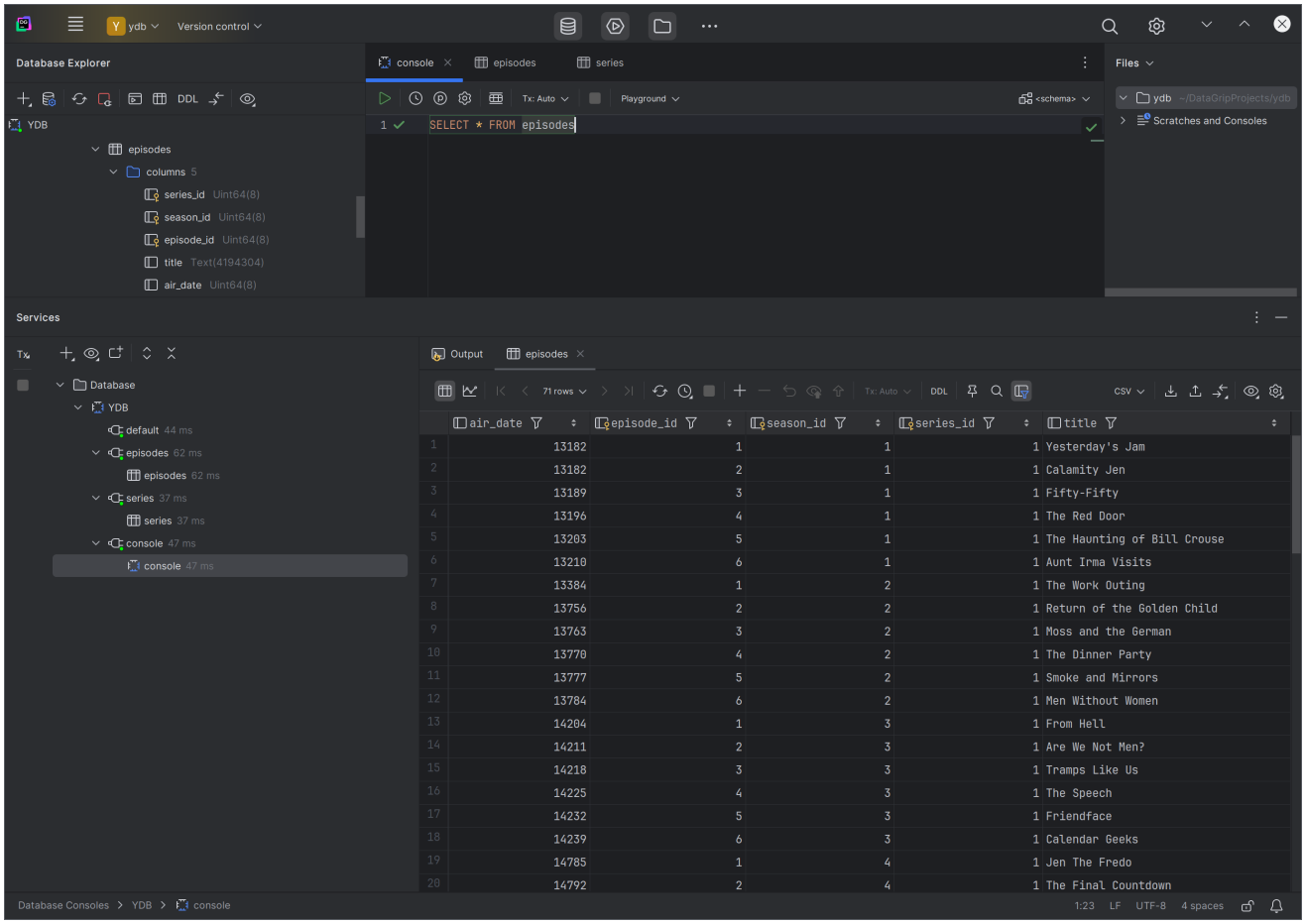
7. Нажмите кнопку **OK** для сохранения соединения.

Работа с YDB

С помощью DataGrip можно просматривать список и структуру таблиц:



А также выполнять запросы к данным:


ydb
Version control
Database Explorer
DDL
YDB
episodes
columns 5
series_id Uint64(8)
season_id Uint64(8)
episode_id Uint64(8)
title Text(4194304)
air_date Uint64(8)
console
episodes
series
Tx: Auto
Playground
<schema>
1
SELECT * FROM episodes
Files
ydb ~/DataGripProjects/ydb
Scratches and Consoles
Services
Database
YDB
default 44 ms
episodes 62 ms
episodes 62 ms
series 37 ms
series 37 ms
console 47 ms
console 47 ms
Output
episodes
71 rows
Tx: Auto
DDL
CSV
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
Database Consoles > YDB > console
1:23 LF UTF-8 4 spaces

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 {% file src="https://raw.githubusercontent.com/ydb-platform/ydb-sqlalchemy/refs/heads/main/examples/jupyter_notebook/YDB%20SQLAlchemy%20%2B%20Jupyter%20Notebook%20Example.ipynb" name="YDB SQLAlchemy - Jupyter Notebook Example.ipynb" %}, откройте его в Jupyter и последовательно пройдитесь по каждой ячейке, выполняя код по мере необходимости.

Apache Superset

Apache Superset — это современная платформа для анализа и визуализации данных.

Режим совместимости с PostgreSQL в YDB позволяет использовать Apache Superset для выполнения запросов и визуализации данных из YDB. В этом случае Apache Superset работает с YDB как с PostgreSQL.

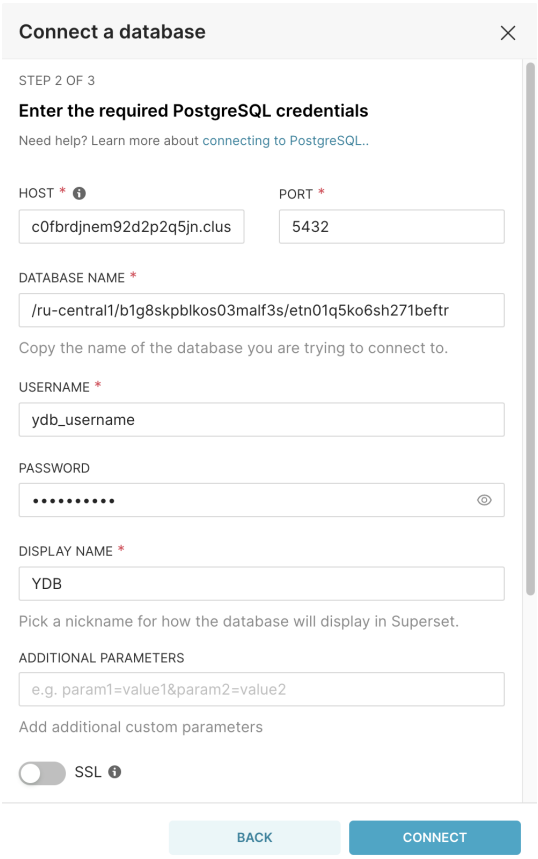
> **Важно**
>
> Поддержка синтаксиса PostgreSQL в YDB **находится в разработке**. Использовать её в production окружениях **не рекомендуется**. Запросы в PostgreSQL могут исполняться до нескольких раз медленнее по сравнению с аналогичными запросами на YQL.
>
> Основной сценарий, который можно тестировать — выполнение аналитических запросов к хранимым в YDB данным.

Создание подключения к YDB

Чтобы создать подключение к YDB из Apache Superset с использованием сетевого протокола PostgreSQL, 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 нажмите кнопку **PostgreSQL**.
4. На втором шаге мастера введите данные для подключения к YDB в следующие поля:
   - **HOST** — эндпоинт кластера YDB, к которому осуществляется подключение.
   - **PORT** — порт эндпоинта YDB.
   - **DATABASE NAME** — путь к базе данных в кластере YDB, к которой будут выполняться запросы.
   - **USERNAME** — логин для подключения к базе данных YDB.
   - **PASSWORD** — пароль для подключения к базе данных YDB.
   - **DISPLAY NAME** — наименование соединения с YDB в Apache Superset.



5. Нажмите кнопку **CONNECT**.
6. Нажмите кнопку **FINISH**, чтобы сохранить подключение.

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**.

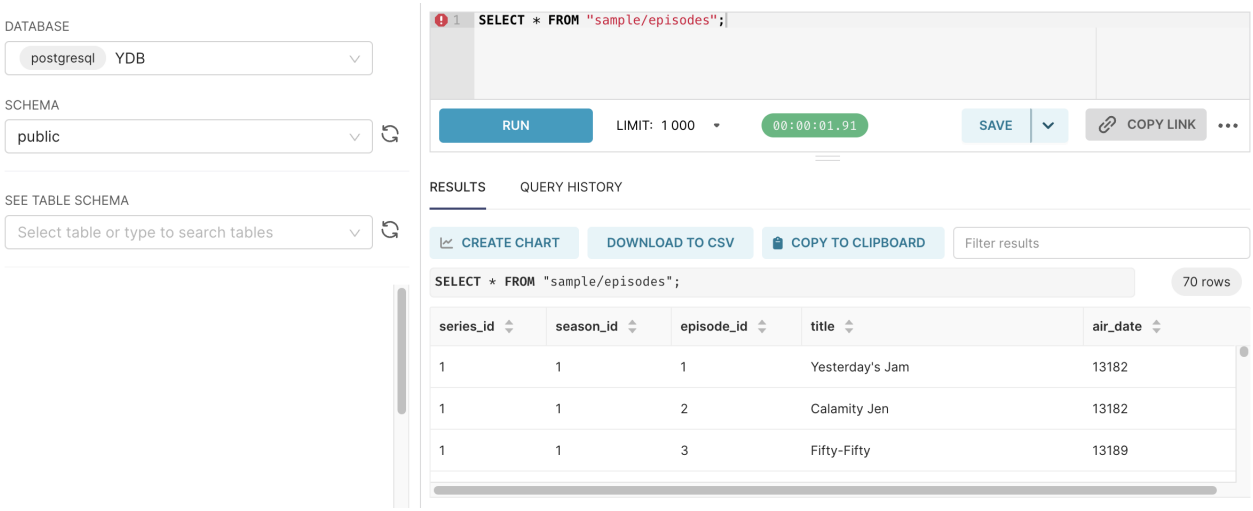
2. В выпадающем списке **DATABASE** выберите подключение к YDB.
3. В выпадающем меню **SCHEMA** выберите `public`.

> **Внимание**
>
> В настоящее время YDB не предоставляет схемы таблиц через сетевой протокол PostgreSQL. Поэтому выбор таблицы в выпадающем списке **SEE TABLE SCHEMA** можно пропустить.

4. 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>";
> ```

5. Нажмите кнопку **RUN**, чтобы проверить SQL-запрос.



6. Нажмите на стрелку рядом с кнопкой **SAVE** и выберите **Save dataset** в выпадающем списке.

   Откроется диалоговое окно **Save or Overwrite Dataset**.
7. 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.

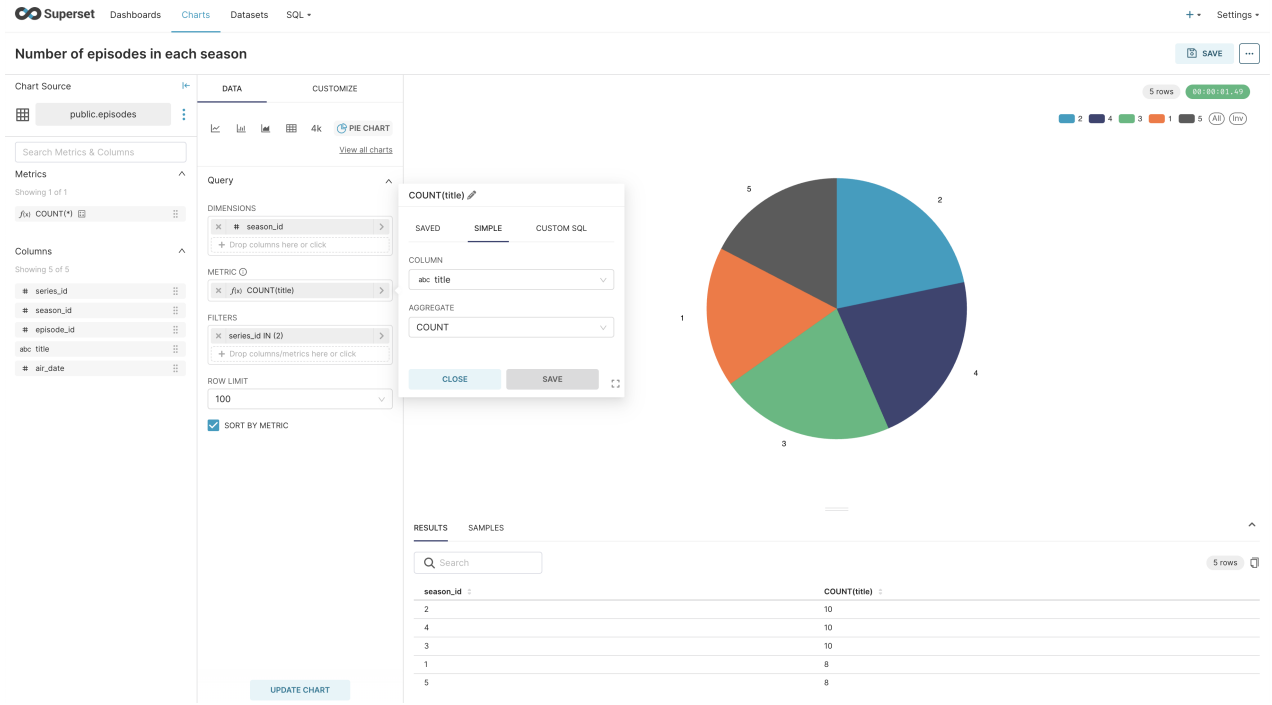
Чтобы создать диаграмму, выполните следующие шаги:

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

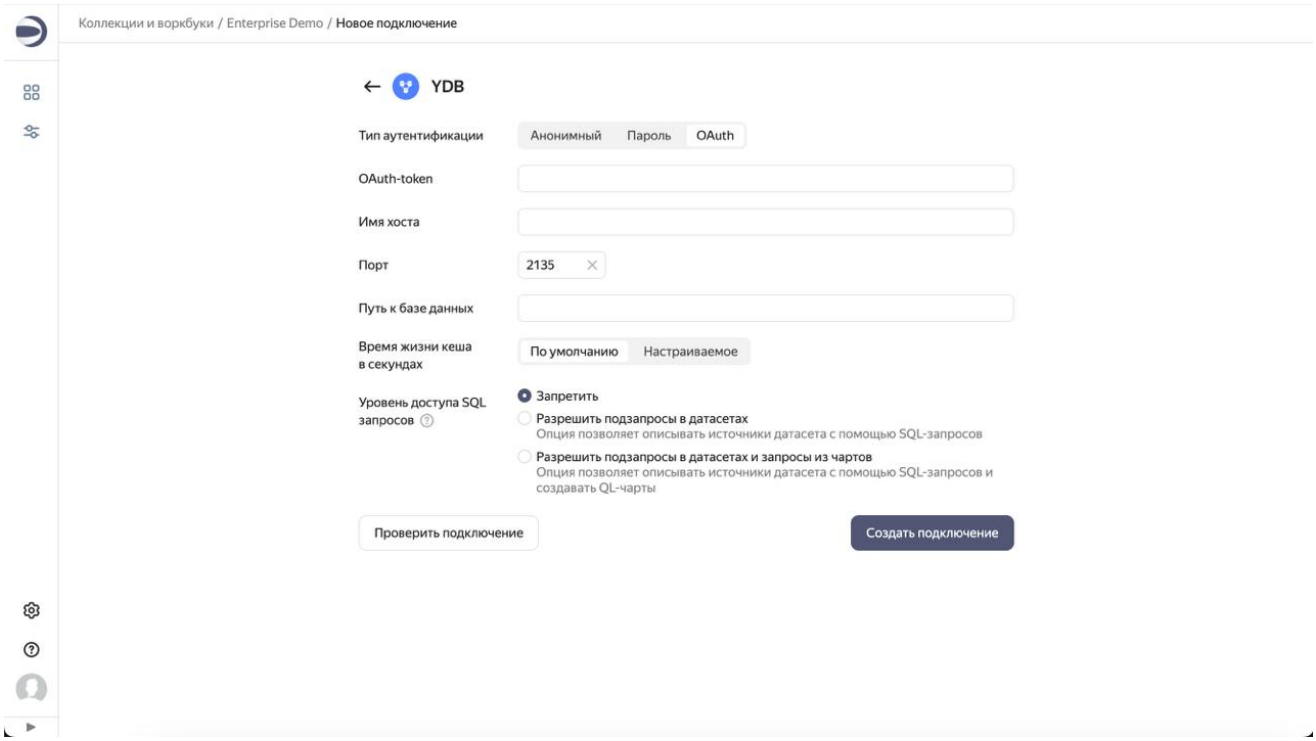
   - **Имя хоста** — хостнейм для подключения к YDB.
   - **Порт** — порт подключения к YDB. По умолчанию — 2135.
   - **Путь к базе данных** — имя базы данных для подключения.
   - **Имя пользователя** — имя пользователя для подключения к YDB.
   - **Пароль** — пароль пользователя.

   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


Коллекции и воркбуки / Enterprise Demo / Новое подключение
YDB
Тип аутентификации
Анонимный
Пароль
OAuth
OAuth-token
Имя хоста
Порт
2135
Путь к базе данных
Время жизни кеша в секундах
По умолчанию
Настраиваемое
Уровень доступа SQL запросов
Запретить
Разрешить подзапросы в датасетах
Опция позволяет описывать источники датасета с помощью SQL-запросов
Разрешить подзапросы в датасетах и запросы из чартов
Опция позволяет описывать источники датасета с помощью SQL-запросов и создавать QL-чарты
Проверить подключение
Создать подключение

FineBI

FineBI — это мощный инструмент для анализа больших данных. FineBI дает возможность организациям анализировать данные и делиться ими для принятия информированных решений. Он позволяет визуализировать необработанные данные, отслеживать ключевые показатели эффективности (KPI), идентифицировать тренды и прогнозировать будущие результаты.

Режим совместимости с PostgreSQL в YDB позволяет использовать FineBI для выполнения запросов и визуализации данных из YDB. В этом случае FineBI работает с YDB как с PostgreSQL.

Важно

Поддержка синтаксиса PostgreSQL в YDB **находится в разработке**. Использовать её в production окружениях **не рекомендуется**. Запросы в PostgreSQL могут исполняться до нескольких раз медленнее по сравнению с аналогичными запросами на YQL.

Основной сценарий, который можно тестировать — выполнение аналитических запросов к хранимым в YDB данным.

Пререквизиты

Для работы понадобятся:

- FineBI;
- PostgreSQL JDBC-драйвер, загруженный в FineBI.

Примечание

Скачать актуальный PostgreSQL JDBC-драйвер можно со страницы загрузки сайта PostgreSQL. Чтобы загрузить JDBC-драйвер для PostgreSQL в FineBI, следуйте инструкции в документации FineBI.

Создание подключения к YDB

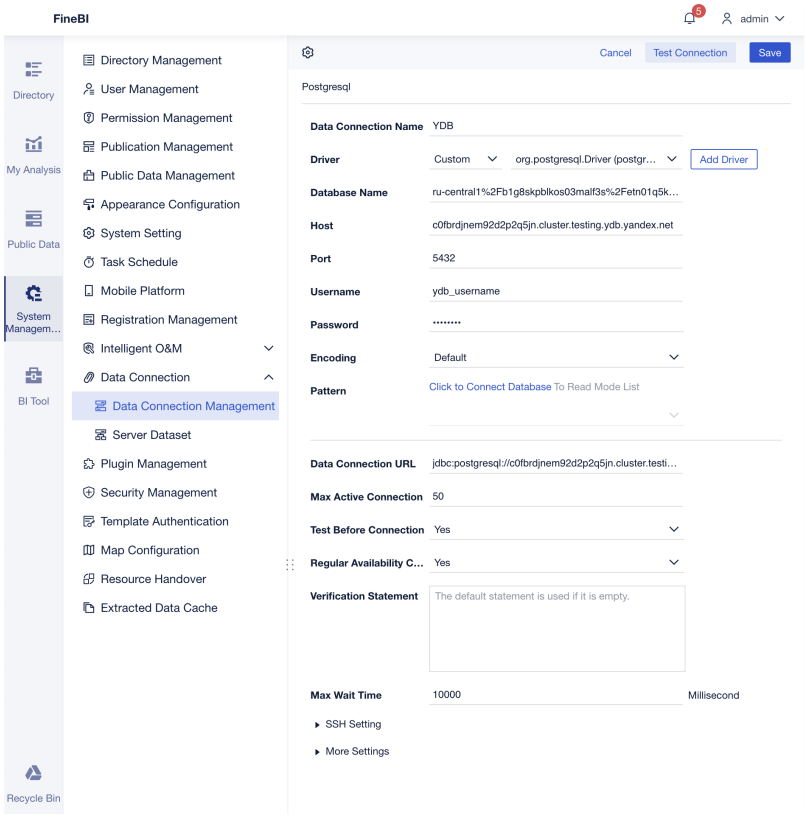
Чтобы создать подключение к YDB из FineBI с использованием сетевого протокола PostgreSQL, выполните следующие шаги:

1. Войдите в аккаунт `admin` на FineBI.
2. Перейдите в **System Management** > **Data Connection** > **Data Connection Management**.
3. Нажмите кнопку **New Data Connection**.
4. Чтобы найти иконку PostgreSQL, в поле **Search** введите `postgresql`.
5. Нажмите иконку PostgreSQL.
6. Введите данные для подключения к YDB в следующие поля:
   - **Data Connection Name** — наименование соединения с YDB в FineBI.
   - **Driver** — драйвер для подключения FineBI к YDB.

     Выберите `Custom` и установленный JDBC-драйвер `org.postgresql.Driver`.
   - **Database Name** — путь к базе данных в кластере YDB, к которой будут выполняться запросы.

     Внимание

     Специальные символы в пути к базе данных должны быть закодированы. Например, убедитесь, что все слэши (`/`) заменены на `%2F`.
   - **Host** — эндпоинт кластера YDB, к которому осуществляется подключение.
   - **Port** — порт эндпоинта YDB.
   - **Username** — логин для подключения к базе данных YDB.
   - **Password** — пароль для подключения к базе данных YDB.



7. Нажмите кнопку **Test Connection**.

   Если данные для подключения указаны верно, появится сообщение об удачном установлении соединения.
8. Чтобы сохранить подключение, нажмите кнопку **Save**.

   Созданное подключение к YDB появится в списке **Data Connection**.

Создание набора данных (SQL dataset)

Чтобы создать набор данных из таблицы YDB, выполните следующие шаги:

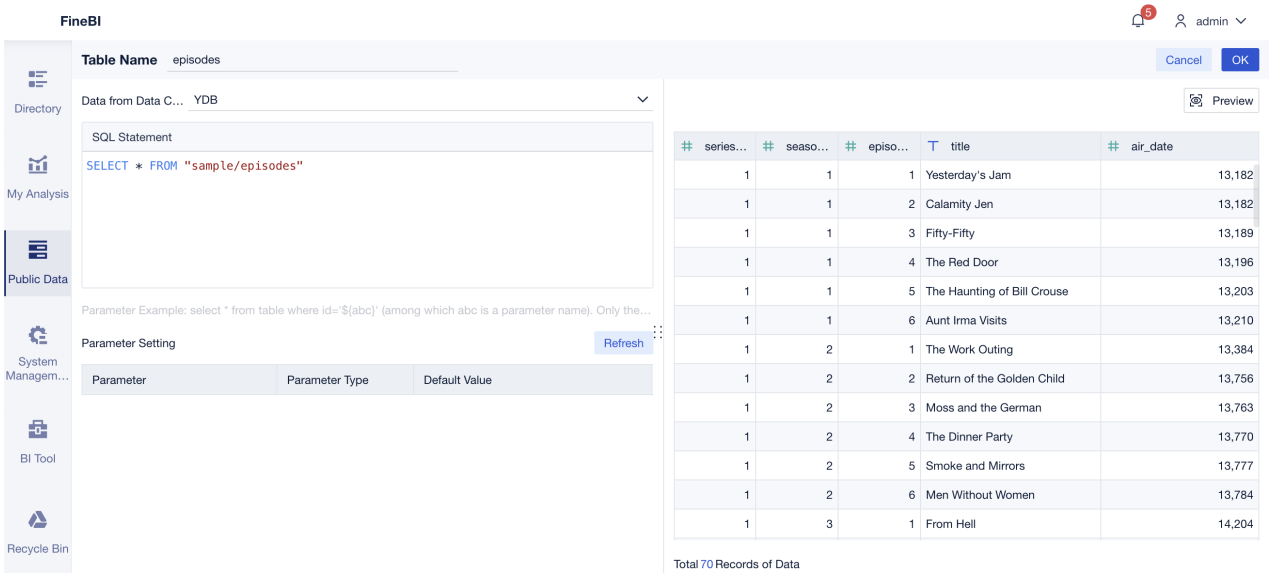
1. В FineBI откройте вкладку **Public Data**.
2. Выберите папку, в которой вы хотите создать набор данных.

> **Важно**
>
> У вас должны быть права Public Data Management для выбранной папки в FineBI.

3. Нажмите кнопку **Add Dataset** и выберите **SQL Dataset** в выпадающем списке.
4. В поле **Table Name** введите наименование набора данных.
5. В выпадающем списке **Data from Data Connection** выберите подключение к YDB.
6. В поле **SQL Statement** введите текст SQL-запроса на получение всех колонок из таблицы YDB. Например, `SELECT * FROM <YDB_table_name>`.

> **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 "<path/to/subdirectory/table_name>";
> ```

7. Нажмите кнопку **Preview**, чтобы проверить SQL-запрос. Если SQL-запрос составлен правильно, появится таблица с данными в правой части страницы.



8. Чтобы сохранить набор данных, нажмите кнопку **OK**.

После создания наборов данных вы можете использовать данные из YDB для создания диаграмм в FineBI. См. документацию FineBI.

Создание диаграммы

Теперь давайте создадим пример диаграммы с использованием набора данных из таблицы `episodes`, создание и наполнение данными которой описано в Туториале по YQL. Мы создадим круговую диаграмму, в которой будет видно, сколько серий содержит каждый сезон сериала.

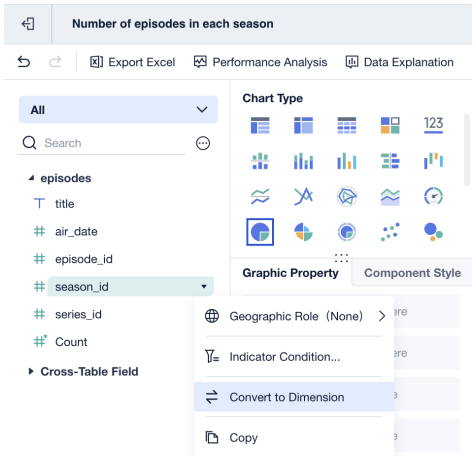
Таблица `episodes` содержит следующие колонки:

- `series_id`;
- `season_id`;
- `episode_id`;
- `title`;
- `air_date`.

Чтобы создать диаграмму, выполните следующие шаги:

1. В FineBI откройте вкладку **My Analysis**.
2. Нажмите кнопку **New Subject**.

   Откроется диалоговое окно **Select Data**.
3. В открывшемся окне **Select Data** выберите набор данных для таблицы `episodes` и нажмите кнопку **OK**.
4. Откройте вкладку **Component** внизу страницы.
5. На панели **Chart Type** нажмите иконку **Pie Chart**.
6. В списке колонок набора данных `episodes` нажмите стрелку рядом с колонкой `episode_id` и выберите пункт **Convert to Dimension** в выпадающем списке.



7. Перетащите колонку `season_id` в поле **Color**.
8. Перетащите колонку `title` в поле **Label**.
9. Перетащите колонку `series_id` в поле **Filter**.

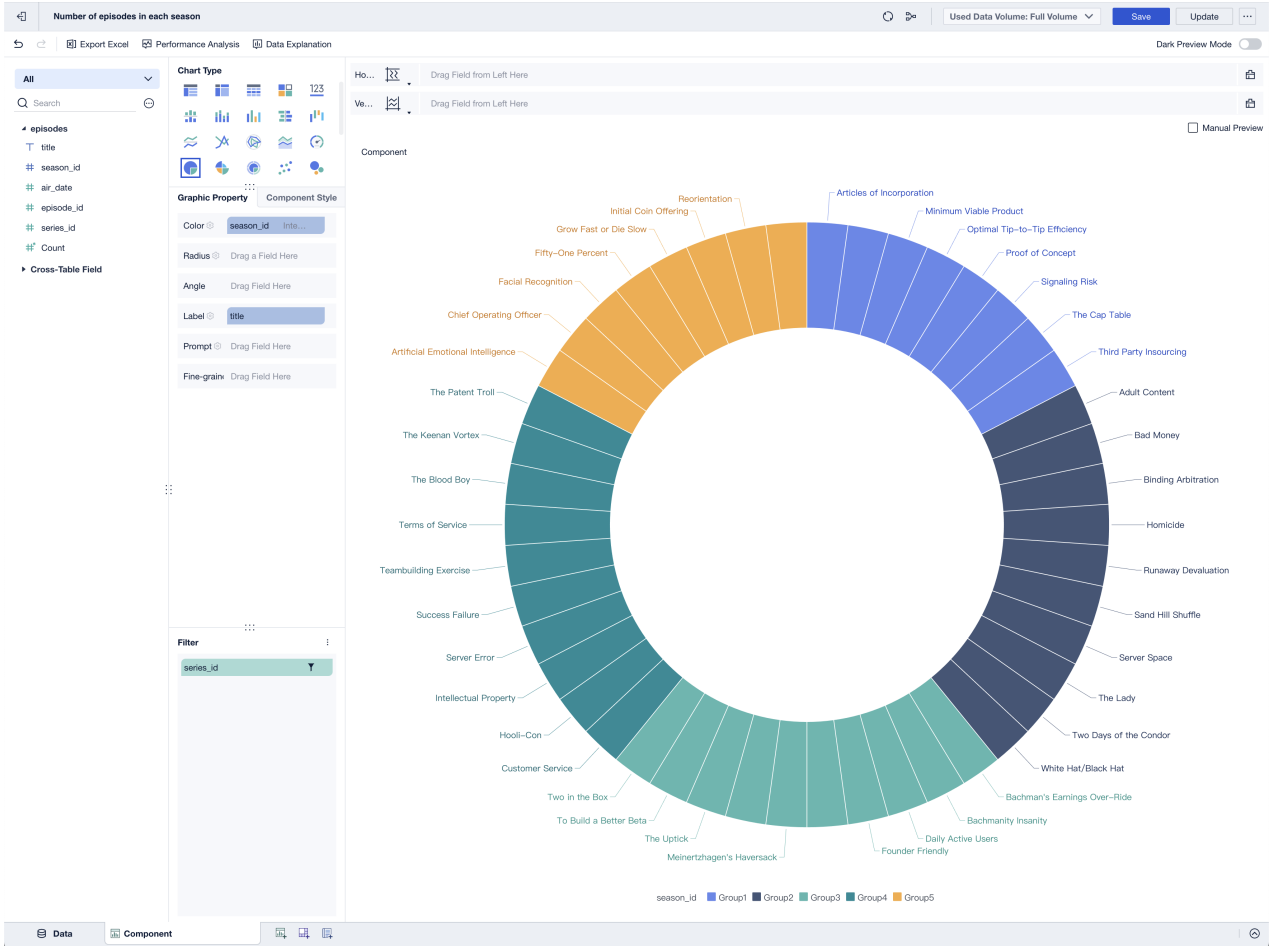
   Откроется диалоговое окно **Add Filter to episodes.series_id**.
10. В открывшемся окне **Add Filter to episodes.series_id** выберите пункт `Detailed Value` и нажмите кнопку **Next Step**.

11. Укажите следующее условие:

    `series_id` `Equal To` `Fixed Value` `2`

12. Нажмите кнопку **OK**.

    Диаграмма будет использовать данные только для сериала с идентификатором `2`.



13. Нажмите кнопку **Save**.

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.

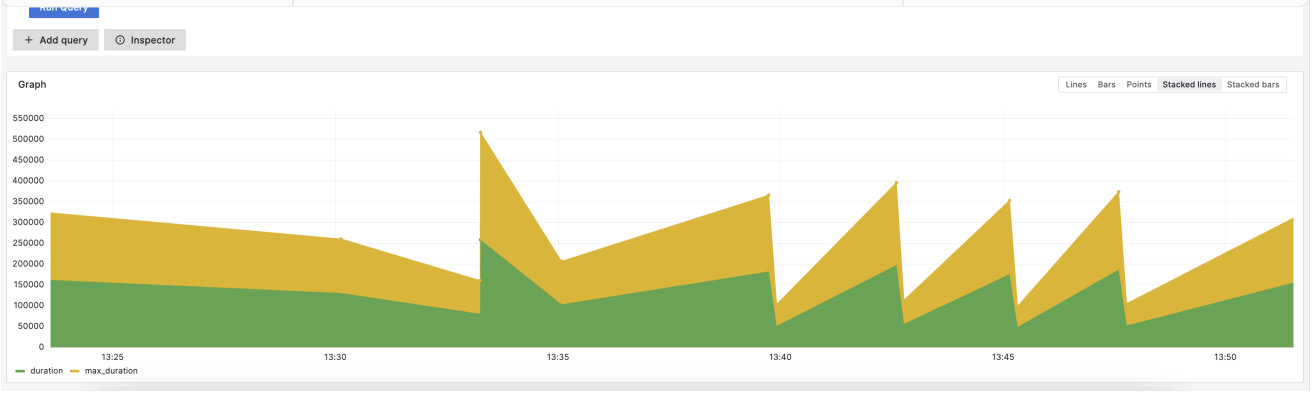
Пример настройки источника данных YDB с использованием аутентификации в через логин/пароль:

```
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)



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

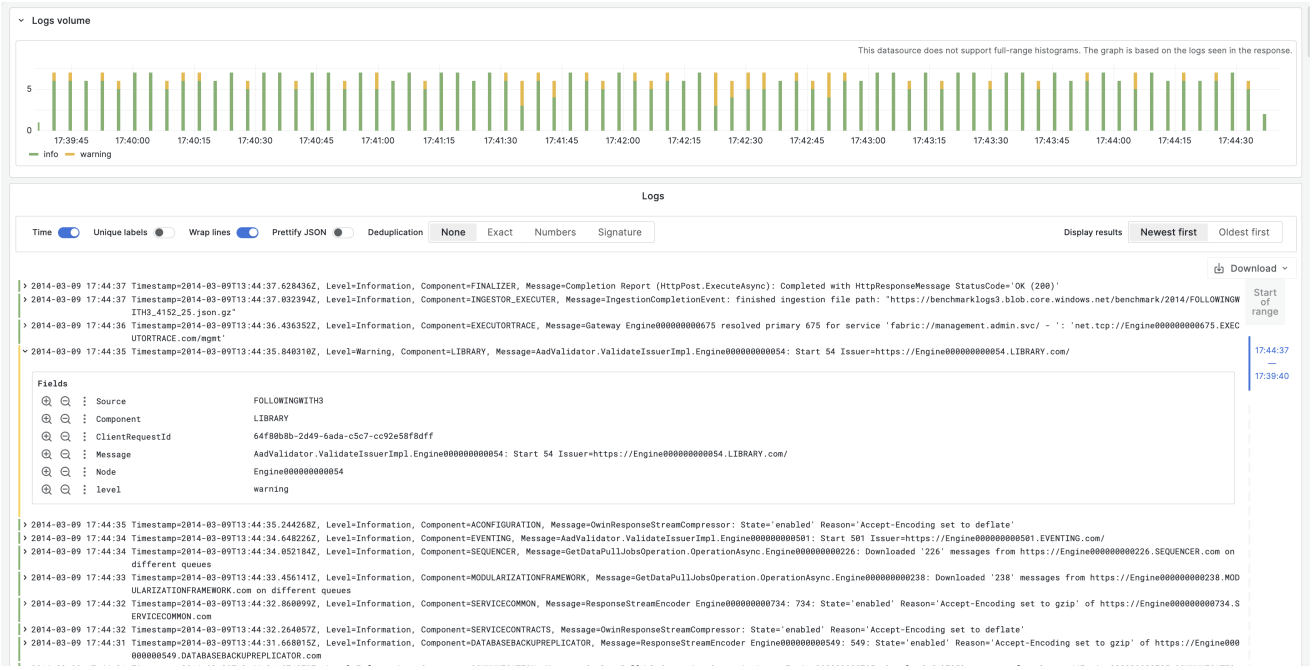
Для таких запросов использование колоночных таблиц может быть полезным с точки зрения производительности.

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
`$__timeFilter(columnName)`	Заменяется условием, которое фильтрует данные в указанной колонке или результате выражения на основании временного диапазона, заданного на панели в микросекундах	`foo >= CAST(1636717526371000 AS Timestamp) AND foo <= CAST(16682535263`
`$__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', $bar)` `foo` если переменная `bar` не определена, ил `$bar`

```
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.

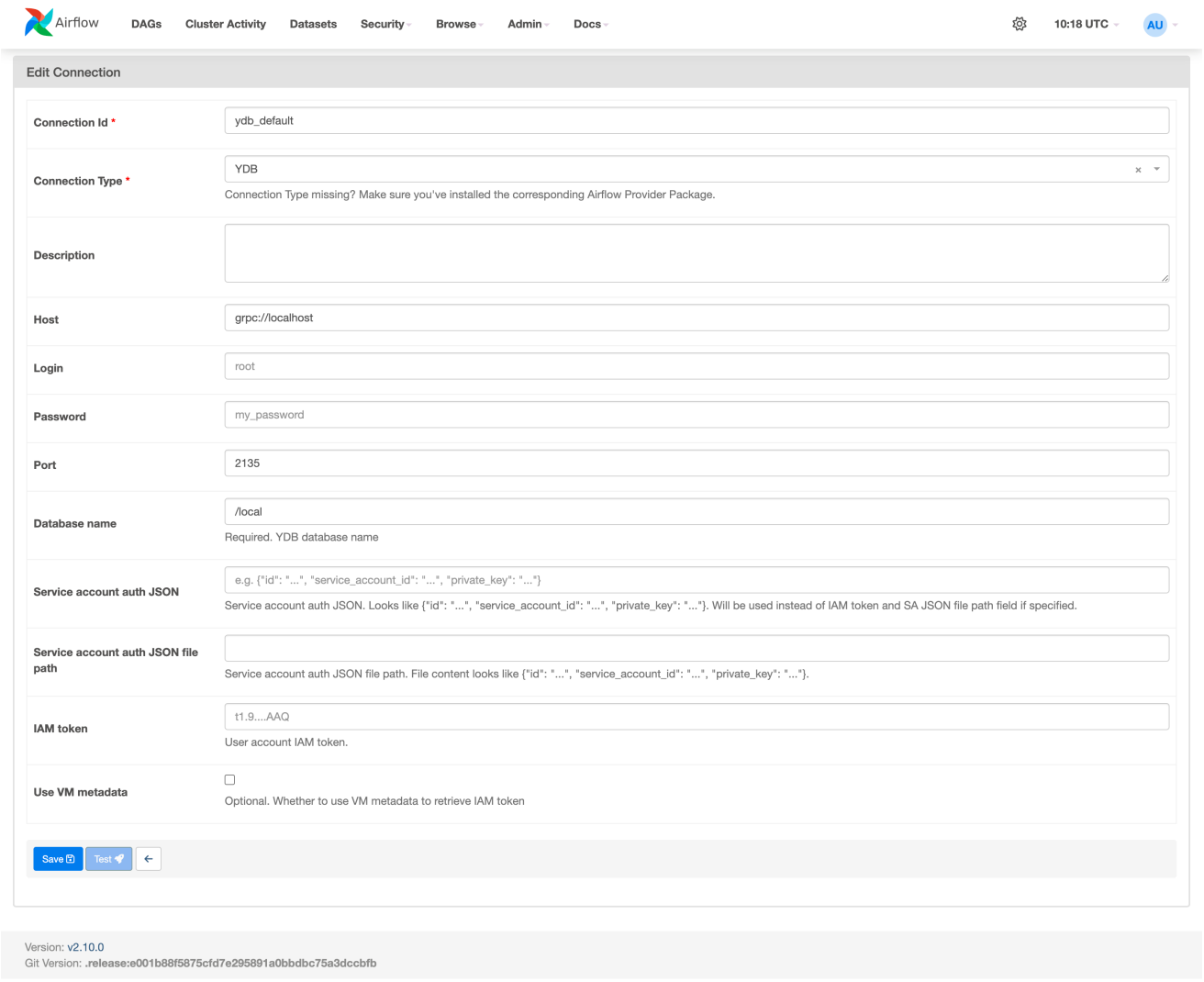
Пример:

```
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

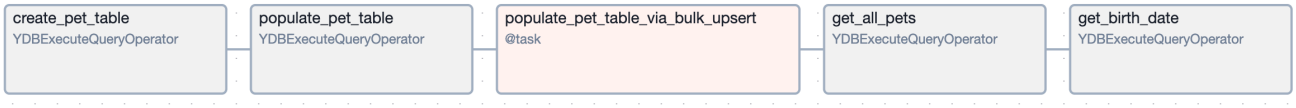
YQL-тип	Python-тип	Пример в Python
		`647713`
		True
		`7.88731023`
		`None`

YQL-тип	Python-тип	Пример в Python
		`{"a":[1,2,3]}`

YQL-тип	Python-тип	Пример в Python
		`list` `[1,2,3,4]`
		`dict` `{key1: "value1", key2: "value2"}`

YQL-тип	Python-тип
`Void`, `Null`	`None`

`EmptyList`	`[]`	`set`	`set(key_value1, key_value2)`		`45.23410083`
`EmptyDict`	`{}`	`tuple`	`(element1, element2)`		`Текст строки`
`Struct<Name:Utf8,Age:Int32>`		`dict`	`{ "Name": "value1", "Age": value2 }`		`Текст строки`



Для выполнения запросов к базе данных YDB используется предварительно созданное соединение с YDB типа YDB Connection с именем `test_ydb_connection`.

```
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="test_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
    )
```

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.

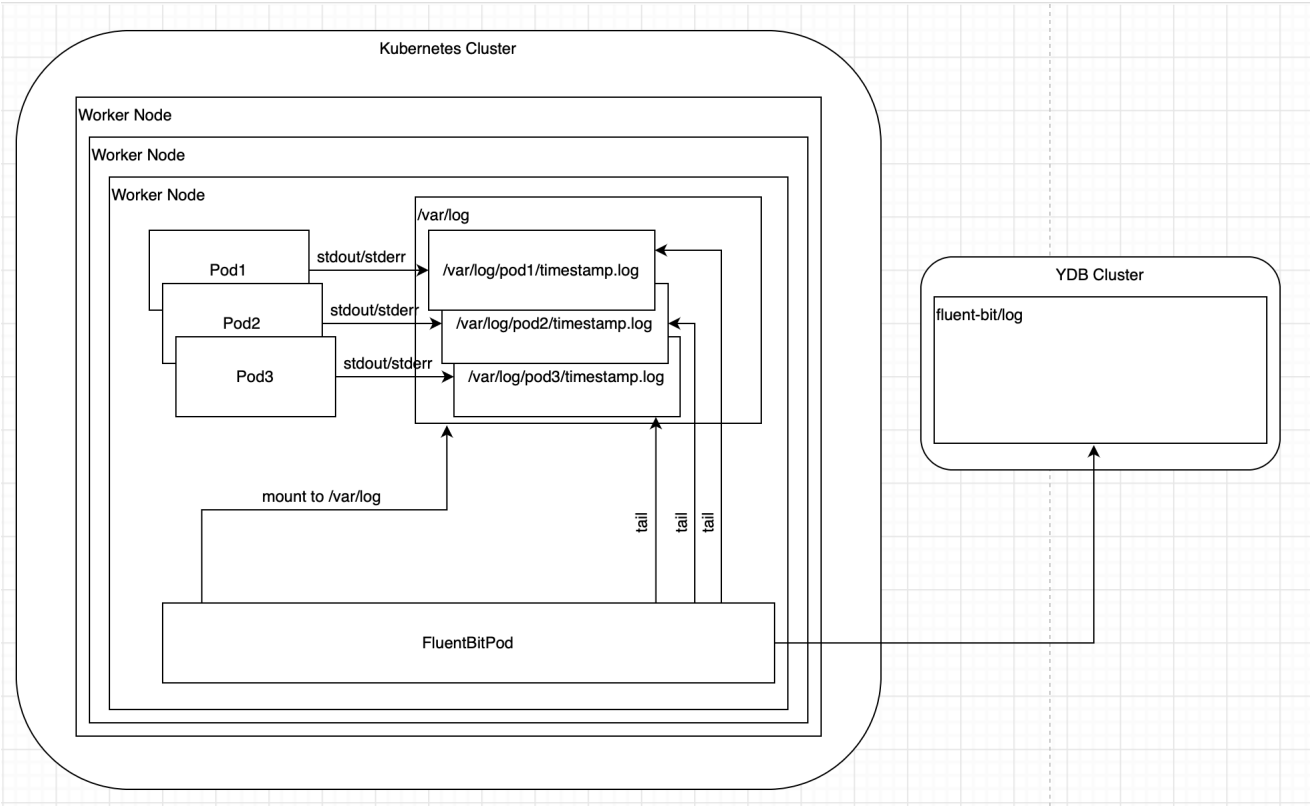
Настройка записи логов в YDB

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@`
`CredentialsYcServiceAccountKeyFile`	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`



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":"message_structured","stream":"pi
        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", "level": 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 (не забывая что чтение из колоночной таблицы возможно только в режиме ScanQuery):

```
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_topic
```

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_topic
ydb -e grpc://localhost:2136 -d /local topic read /local/logstash_demo_topic --consumer logstash-consumer --commit true
```

Чтение вернет содержимое отправленного сообщения:

```
{"level":4,"message":"demo message","timestamp":1716470292640,"user":"demo_user"}
```

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=AdventureWorks2022;
          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/etn63999hrinbapmef6g
        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>
```

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

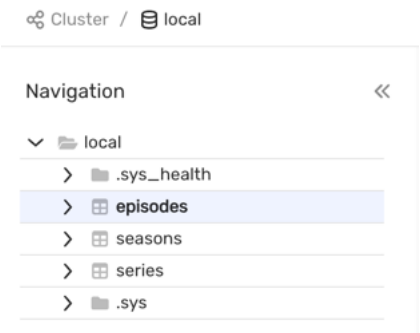
Flyway Desktop в настоящее время не поддерживается.

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:

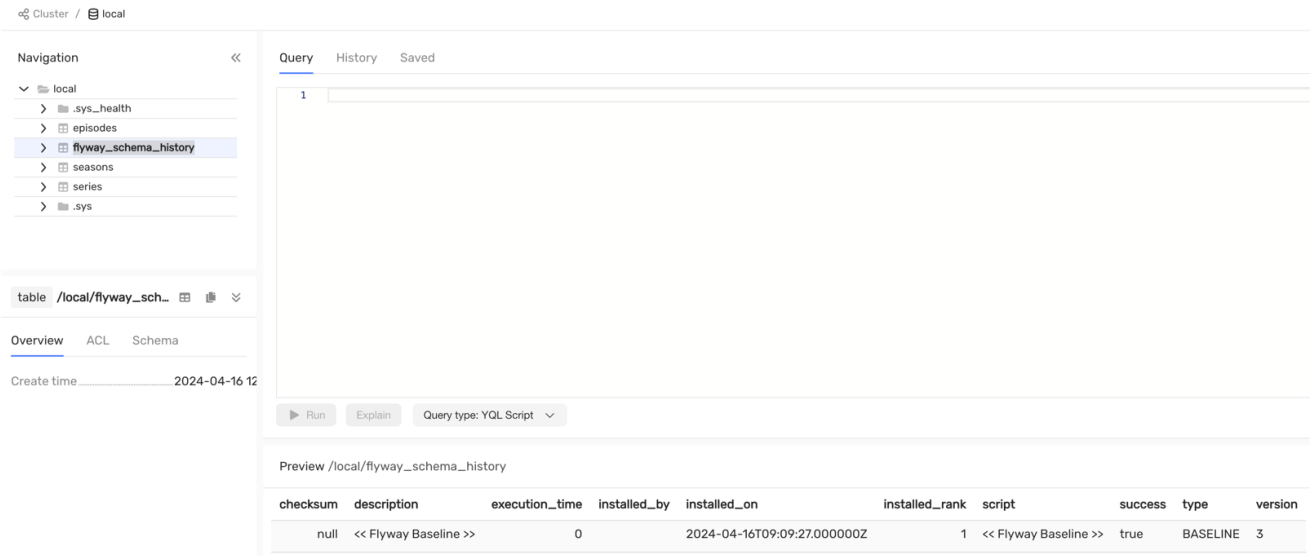
V1__create_series.sql

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`

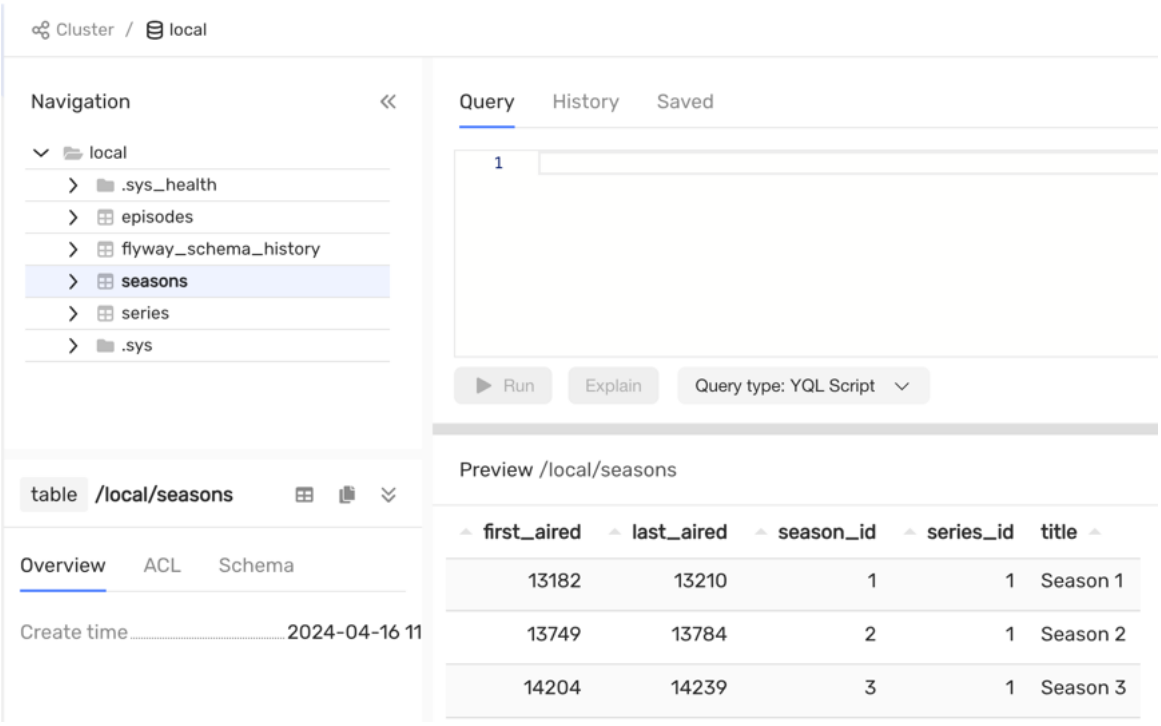
```
INSERT INTO series (series_id, title, release_date, series_info)
VALUES
    -- By default, numeric literals have type Int32
    -- if the value is within the range.
    -- Otherwise, they automatically expand to Int64.
    (1,
     "IT Crowd",
     CAST(Date ("2006-02-03") AS Uint64), -- CAST converts 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 Dave
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
```

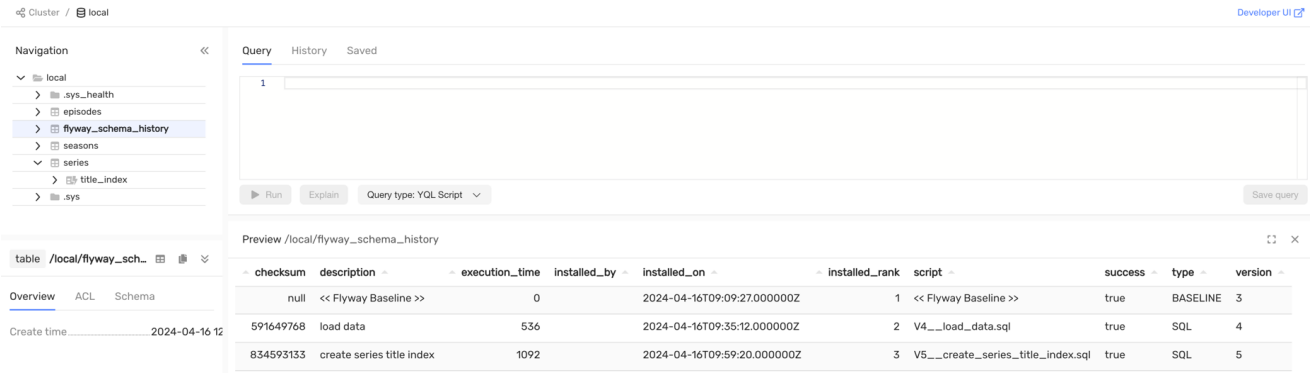
Содержимое файла `V5__create_series_title_index.sql`

```
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              | Undoable |
+-----------+---------+---------------------------+----------+---------------------+--------------------+----------+
| 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
```

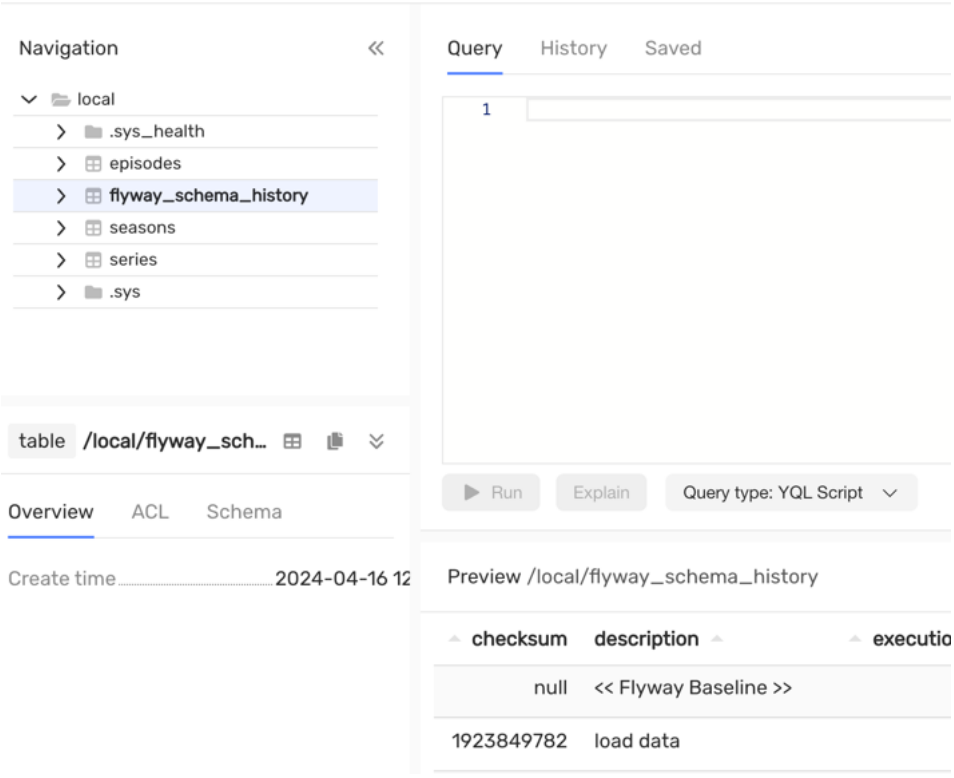
repair

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 подлкючения 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=declare,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_query_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`:

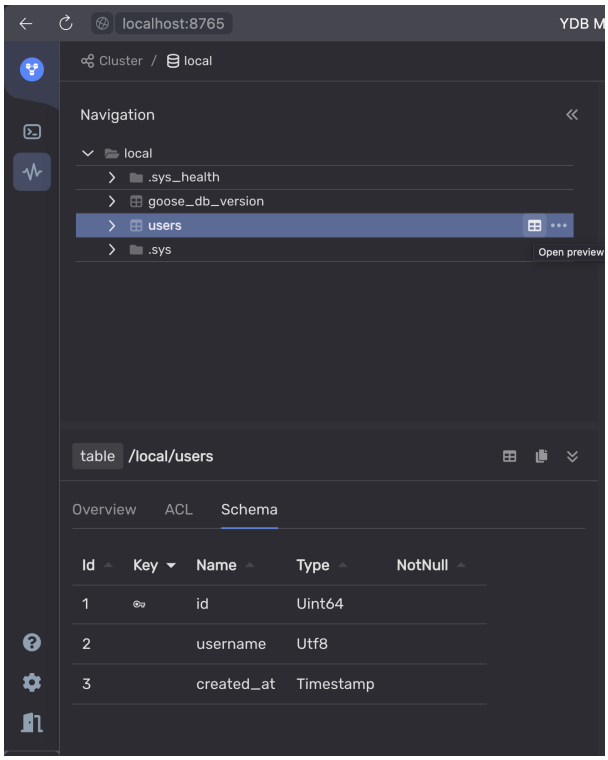
```
$ goose ydb $YDB_CONNECTION_STRING up
2024/01/12 11:55:18 OK   20240112115229_00001_create_first_table.sql (93.58ms)
2024/01/12 11:55:18 goose: successfully migrated database to version: 20240112115229
```

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):

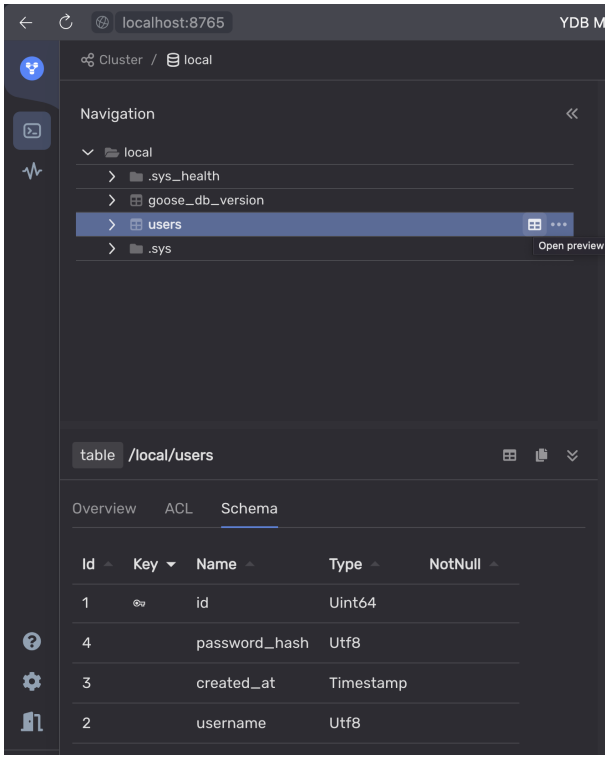
```
$ goose ydb $YDB_CONNECTION_STRING up-by-one
2024/01/12 12:04:56 OK   20240112120057_00002_add_column_password_hash_into_table_users.sql (59.93ms)
```

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)
```

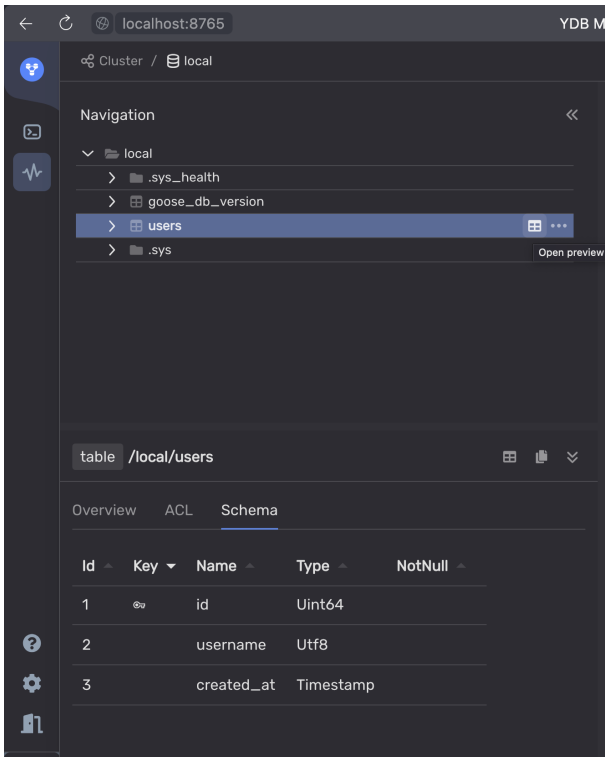
Проверим статус миграций `goose status`:

```
$ 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`

```
    }
  }
]
```

yaml

```
changes:
- addColumn:
    tableName: seasons
    columns:
    - column:
        name: is_deleted
        type: bool
```

`createIndex` - создание вторичного индекса. Пример:

xml

```
<createIndex tableName="episodes" indexName="episodes_index" unique="false">
    <column name="title"/>
</createIndex>
```

json

```
"changes": [
    {
      "createIndex": {
        "tableName": "episodes",
        "indexName": "episodes_index",
        "unique": "false",
        "columns": {
          "column": {
            "name": "title"
          }
        }
      }
    }
```

yaml

```
changes:
- createIndex:
    tableName: episodes
    indexName: episodes_index
    unique: false
    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

```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

```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

```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

```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)

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-dialect/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
```

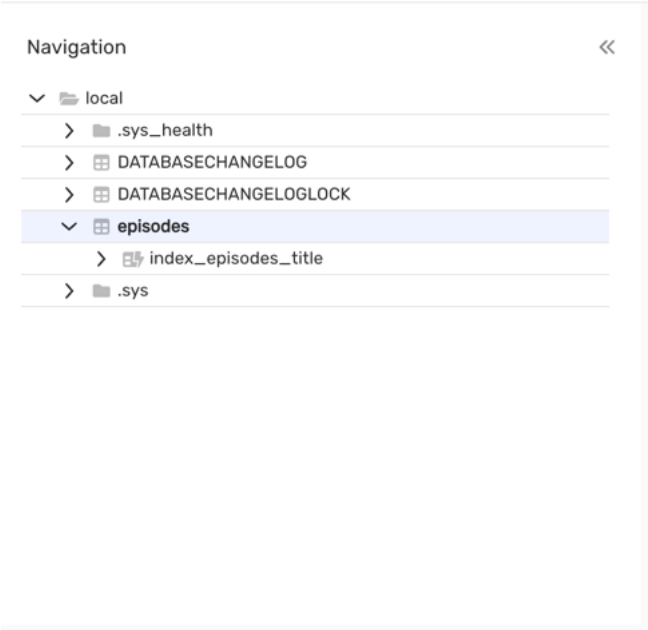
Применим к пустой базе данных следующий changeset:

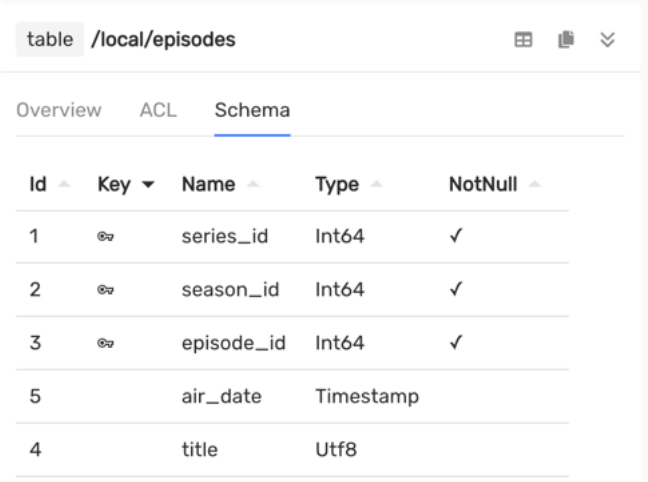
```
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog
        xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
                      http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.8.xsd">

    <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" unique="false">
            <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:





Id	Key	Name	Type	NotNull
1		series_id	Int64	✓
2		season_id	Int64	✓
3		episode_id	Int64	✓
5		air_date	Timestamp	
4		title	Utf8	

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
kurdyukov-kir	Table episodes.		12:53:27	1544007500	createTable tableName=episodes	EXECUTED
kurdyukov-kir	""		12:53:28	1544007500	createIndex indexName=index_episodes_title, tableName=episodes	EXECUTED

```
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`:

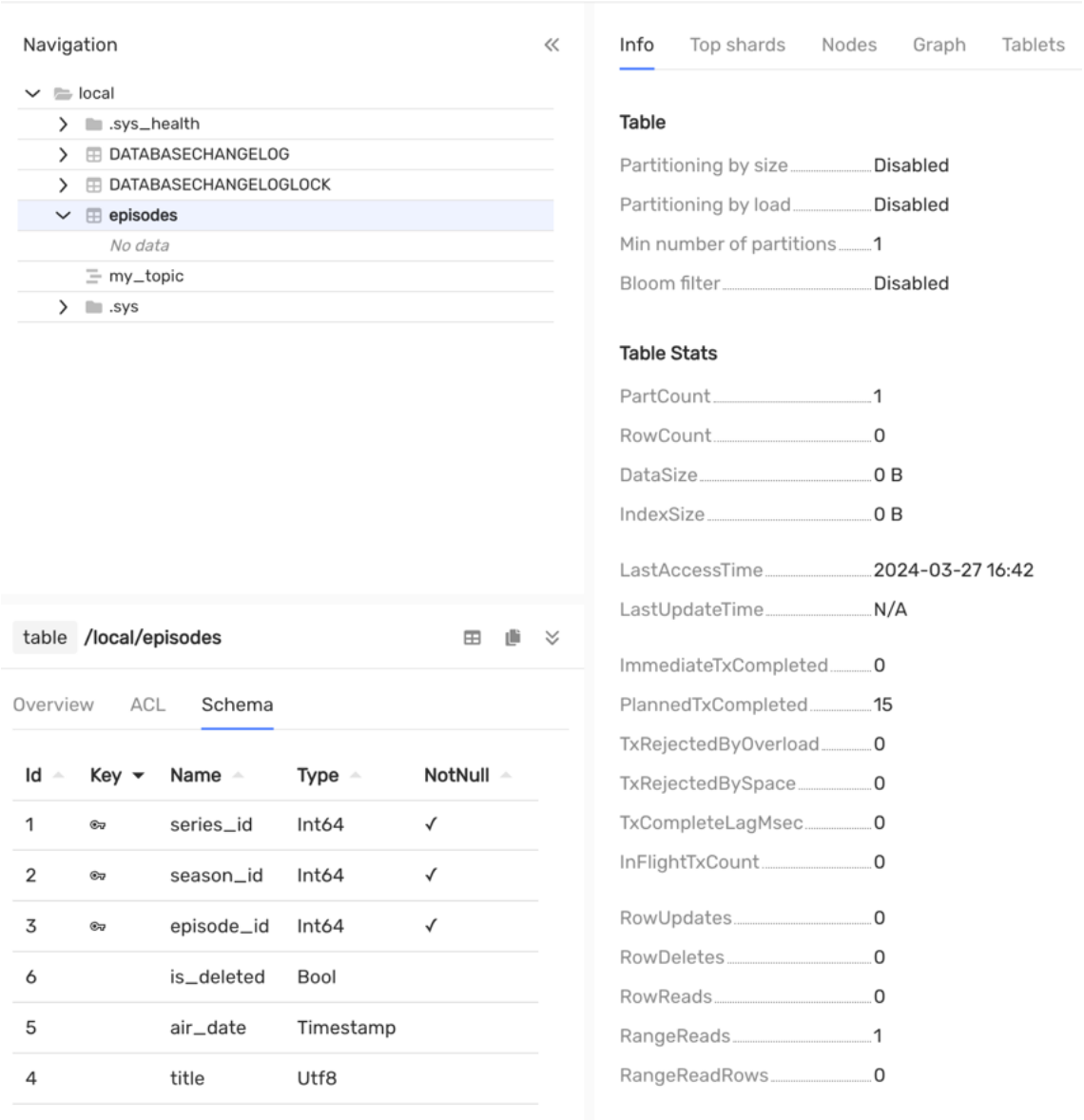
```
<changeSet id="alter-episodes" author="kurdyukov-kir">
    <comment>Alter table episodes.</comment>

    <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:

table /local/all_types_table

Overview ACL Schema

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:

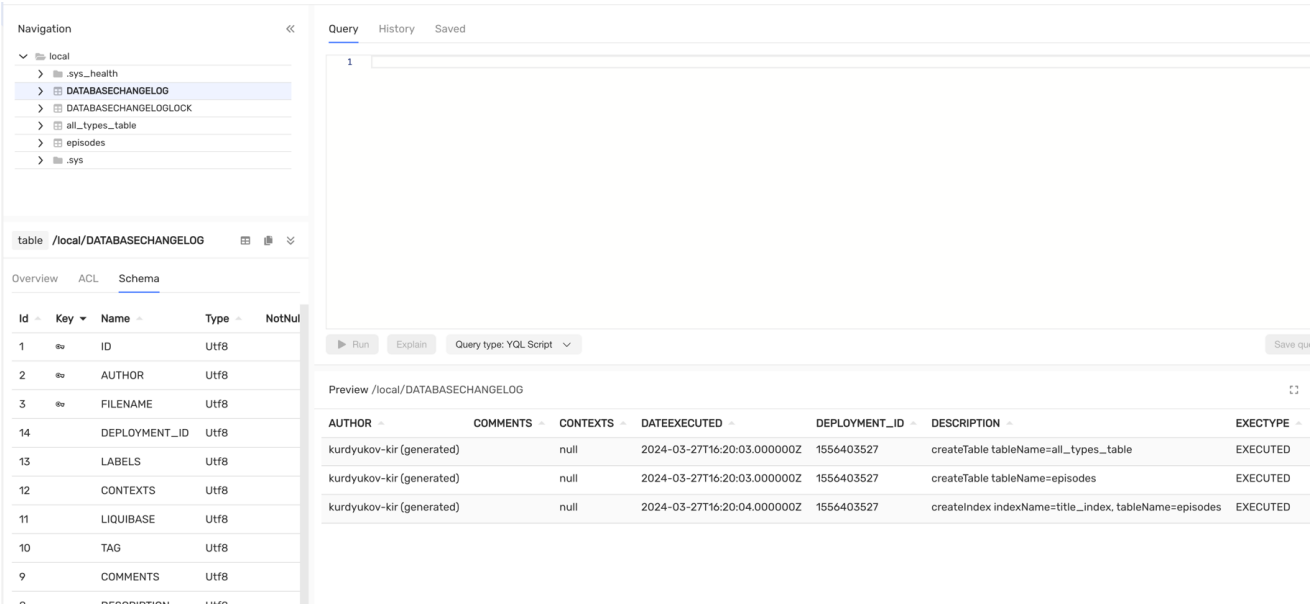
```
<changeSet author="kurdyukov-kir (generated)" id="1711556283305-1">
    <createTable tableName="all_types_table">
        <column name="id" type="INT32">
            <constraints nullable="false" primaryKey="true"/>
        </column>
        <column name="bool_column" type="BOOL"/>
        <column 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.

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`

```
    @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.

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.

Установка

Для использования этой интеграции установите следующее программное обеспечение:

- `langchain-ydb`

  Для установки `langchain-ydb`, полните следующую команду:

  ```
  pip install -qU langchain-ydb
  ```

- Модель ембеддингов

  В этом руководстве используется `HuggingFaceEmbeddings`. Чтобы установить этот пакет, выполните следующую команду:

  ```
  pip install -qU langchain-huggingface
  ```

- Локальный YDB

  Для получения дополнительной информации см. Установите и запустите YDB.

Инициализация

Для создания векторного хранилища YDB, требуется указать модель эмбеддингов. В данном примере используется `HuggingFaceEmbeddings`:

```
from langchain_huggingface import HuggingFaceEmbeddings

embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/all-mpnet-base-v2")
```

После создания модели эмбеддингов создаётся векторное хранилище YDB:

```
from langchain_ydb.vectorstores import YDB, YDBSearchStrategy, YDBSettings

settings = YDBSettings(
    host="localhost",
    port=2136,
    database="/local",
    table="ydb_example",
    strategy=YDBSearchStrategy.COSINE_SIMILARITY,
)
vector_store = YDB(embeddings, config=settings)
```

Управление векторным хранилищем

Создав векторное хранилище, можно взаимодействовать с ним, добавляя и удаляя различные элементы.

Добавление элементов

Подготавливаются документы для работы:

```
from uuid import uuid4

from langchain_core.documents import Document

document_1 = Document(
    page_content="I had chocalate chip pancakes and scrambled eggs for breakfast this morning.",
    metadata={"source": "tweet"},
)

document_2 = Document(
    page_content="The weather forecast for tomorrow is cloudy and overcast, with a high of 62 degrees.",
    metadata={"source": "news"},
)
```

```
document_3 = Document(
    page_content="Building an exciting new project with LangChain - come check it out!",
    metadata={"source": "tweet"},
)

document_4 = Document(
    page_content="Robbers broke into the city bank and stole $1 million in cash.",
    metadata={"source": "news"},
)

document_5 = Document(
    page_content="Wow! That was an amazing movie. I can't wait to see it again.",
    metadata={"source": "tweet"},
)

document_6 = Document(
    page_content="Is the new iPhone worth the price? Read this review to find out.",
    metadata={"source": "website"},
)

document_7 = Document(
    page_content="The top 10 soccer players in the world right now.",
    metadata={"source": "website"},
)

document_8 = Document(
    page_content="LangGraph is the best framework for building stateful, agentic applications!",
    metadata={"source": "tweet"},
)

document_9 = Document(
    page_content="The stock market is down 500 points today due to fears of a recession.",
    metadata={"source": "news"},
)

document_10 = Document(
    page_content="I have a bad feeling I am going to get deleted :(",
    metadata={"source": "tweet"},
)

documents = [
    document_1,
    document_2,
    document_3,
    document_4,
    document_5,
    document_6,
    document_7,
    document_8,
    document_9,
    document_10,
]
uuids = [str(uuid4()) for _ in range(len(documents))]
```

Элементы добавляются в векторное хранилище с помощью функции `add_documents`.

```
vector_store.add_documents(documents=documents, ids=uuids)
```

Результат:

```
Inserting data...: 100%|██████████| 10/10 [00:00<00:00, 14.67it/s]
['947be6aa-d489-44c5-910e-62e4d58d2ffb',
 '7a62904d-9db3-412b-83b6-f01b34dd7de3',
 'e5a49c64-c985-4ed7-ac58-5ffa31ade699',
 '99cf4104-36ab-4bd5-b0da-e210d260e512',
 '5810bcd0-b46e-443e-a663-e888c9e028d1',
 '190c193d-844e-4dbb-9a4b-b8f5f16cfae6',
 'f8912944-f80a-4178-954e-4595bf59e341',
 '34fc7b09-6000-42c9-95f7-7d49f430b904',
 '0f6b6783-f300-4a4d-bb04-8025c4dfd409',
 '46c37ba9-7cf2-4ac8-9bd1-d84e2cb1155c']
```

Удаление элементов

Элементы удаляются из векторного хранилища по идентификатору с использованием функции `delete`.

```
vector_store.delete(ids=[uuids[-1]])
```

Результат:

```
True
```

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:

```
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}]")
```

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:

```
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}]")
```

Результат:

```
* [SIM=0.595] The weather forecast for tomorrow is cloudy and overcast, with a high of 62 degrees. [{'source': 'news'}]
* [SIM=0.212] I had chocalate chip pancakes and scrambled eggs for breakfast this morning. [{'source': 'tweet'}]
* [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:

```
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}]")
```

Результат:

```
* I had chocalate 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:

```
retriever = vector_store.as_retriever(
    search_kwargs={"k": 2},
)
results = retriever.invoke(
    "Stealing from the bank is a crime", filter={"source": "news"}
)
for res in results:
    print(f"* {res.page_content} [{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'}]
```

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`.

> **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.

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`.

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.WebClient).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>
```

- `-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`).

Режим и параметры аутентификации выбираются заданием одной из перечисленных ниже опций:

< path="options_cloud.md" keyword="undefined">

- `--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 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

< path="env_static.md" keyword="undefined">

- 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

Дополнительные параметры

При использовании протокола gRPCs (с шифрованием) может потребоваться выбор корневого сертификата:

- `--ca-file <filepath>` : Файл корневого PEM-сертификата для TLS-соединения

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
config info	Просмотр параметров соединения
config profile activate	Активация профиля
config profile create	Создание профиля
config profile delete	Удаление профиля
config profile get	Получение параметров профиля
config profile list	Список профилей
config profile set	Активация профиля
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 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

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)

table attribute drop	Удаление атрибута у строковой или колоночной таблицы
table drop	Удаление строковой или колоночной таблицы
table index add global-async	Добавление асинхронного индекса для строковых таблиц
table index add global-sync	Добавление синхронного индекса для строковых таблиц
table index drop	Удаление индекса у строковых таблиц
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 директории или таблиц в файловую систему
tools pg-convert	Конвертация дампа PostgreSQL, полученного утилитой pg_dump, в формат, понятный YDB
tools rename	Переименование строковых таблиц
tools restore	Восстановление 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 drop	Удаление читателя из топика
topic consumer offset commit	Сохранение позиции 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	Генерация нагрузки

yql	Выполнение YQL-скрипта с поддержкой стриминга (команда устарела, используйте `ydb sql`)

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`.

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 cluster 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
```

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.

```
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.

```
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.granular.remove_schema,ydb
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.granular.remove_schema,ydb
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.

```
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

Внимание

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.

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/seasons-
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

Внимание

Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

С помощью подкоманды `tools rename` вы можете переименовать одну или несколько строковых 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 строковых 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.

```
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. Настройки БД по умолчанию не позволяют задать интервал меньше 15 минут (900 секунд). Значение по умолчанию — `3600`.

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.

В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Удалите таблицу `series`:

```
ydb -p quickstart table drop series
```

Получение плана исполнения запроса и AST

Получить план запроса 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 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
'"episode_id")) '('"season_id" (Member $15 '"season_id")) '('"title" (Member $15 '"title"))))) $5 '($13 $13) (lambda
'($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 'String))))) '0 '0)) '('('"type"
'"data_query"))))
)
```

Потоковое чтение строковой таблицы

Внимание

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
```

Выполнение скан запросов

Важно

Данная команда устарела.
Рекомендуемый инструмент для выполнения запросов в YDB CLI — это команда ydb sql.

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"            |
└───────────┴────────────┴──────────────────────────────────┘
```

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 в файловую систему `tools dump`
- Загрузка из файловой системы `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`

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.

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 объекты 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

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

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:

```
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:

```
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 на файловой системы, ранее выгруженную туда командой `admin cluster dump`, или подготовленную 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:

```
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:

```
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.
- `--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.

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 данных, в описанном в статье Файловая структура формате:

```
ydb [connection options] export s3 [options]
```

Важно

Выгрузка доступна только для объектов следующих типов:

- директория;
- строковая таблица;
- вторичный индекс.
- векторный индекс.

, где `[connection options]` — опции соединения с БД

Параметры командной строки

`[options]` - параметры команды:

Параметры соединения с 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.

Перечень выгружаемых 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 объекты в ней, имена которых не начинаются с точки, а также рекурсивно все поддиректории, имена которых не начинаются с точки.
- `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.

Дополнительные параметры

Параметр	Описание
`--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`).
`--format STRING`	Формат вывода результата. Допустимые значения: • `pretty` — человекопонятный формат (по умолчанию); • `proto-json-base64` — Protocol Buffers в формате JSON, бинарные строки закодированы в Base64.

```
{"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

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 выполнения выгрузки воспользуйтесь командой `operation forget` для того, чтобы выгрузка была завершена: удалена из перечня операций, а также были удалены все созданные для неё файлы:

```
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

> **Примечание**
>
> В примерах используется профиль `quickstart`, подробнее смотрите в Создание профиля для соединения с тестовой БД.

Выгрузка базы данных

Выгрузка всех объектов базы данных, имена которых не начинаются с точки, и не размещенных внутри директорий, имена которых начинаются с точки, 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 \
  --item src=.,dst=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 VJGSOScgs-5kDGeo2hO9 --secret-key fZ_VB1Wi5-fdKSqH6074a7w0J4X0 \
  --item src=dir1,dst=export1/dir1 --item src=dir2,dst=export1/dir2
```

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 данных, в описанном в статье Файловая структура формате:

```
ydb [connection options] import s3 [options]
```

, 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`.

Параметры командной строки

`[options]` - параметры команды:

Параметры соединения с 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.

Перечень загружаемых объектов

`--item STRING`: Описание объекта загрузки. Параметр `--item` может быть указан несколько раз, если необходимо выполнить загрузку нескольких объектов. `STRING` задается в формате `<свойство>=<значение>,...`, со следующими обязательными свойствами:

- `source`, `src` или `s` — путь (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.

Дополнительные параметры

`--description STRING`: Текстовое описание операции, сохраняемое в истории операций.
`--retries NUM`: Количество повторных попыток загрузки, которые будет предпринимать сервер. По умолчанию 10.
`--skip-checksum-validation`: Пропустить этап валидации контрольных сумм загружаемых объектов.
`--format STRING`: Формат вывода результата.

- `pretty`: Человекочитаемый формат (по умолчанию)
- `proto-json-base64`: Protobuf в формате json, бинарные строки закодированы в base64

Выполнение загрузки

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://export/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 выгрузка 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

> **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 \
  --item src=export1,dst=.
```

Загрузка нескольких директорий

Загрузка объектов из директорий dir1 и dir2 бакета S3 `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 VJGSOScgs-5kDGeo2hO9 --secret-key fZ_VB1Wi5-fdKSqH6074a7w0J4X0 \
  --item src=export/dir1,dst=dir1 --item src=export/dir2,dst=dir2
```

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
```

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 их 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.

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`. Значение по умолчанию: `""`.
- `--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-hours`	Время хранения данных в топике, задается в часах. Значение по умолчанию — `18`.
`--partition-write-speed-kbps`	Максимальная скорость записи в партицию, задается в КБ/с. Значение по умолчанию — `1024`.
`--retention-storage-mb`	Максимальный объем хранения, задается в МБ. При достижении ограничения будут удаляться самые старые данные. Значение по умолчанию — `0` (ограничение не задано).
`--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.

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-hours`	Время хранения данных в топике, задается в часах.
`--partition-write-speed-kbps`	Максимальная скорость записи в партицию, задается в КБ/с. Значение по умолчанию — `1024`.
`--retention-storage-mb`	Максимальный объем хранения, задается в МБ. При достижении ограничения будут удаляться самые старые данные. Значение по умолчанию — `0` (ограничение не задано).
`--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.

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. 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.

Убедитесь, что читатель создан:

```
ydb -p quickstart scheme describe my-topic
```

Результат:

```
RetentionPeriod: 2 hours
PartitionsCount: 2
SupportedCodecs: RAW, GZIP

Consumers:
┌──────────────┬─────────────────┬──────────────────────────────┬─────────────┐
| ConsumerName | SupportedCodecs | ReadFrom                     | Important |
├──────────────┼─────────────────┼──────────────────────────────┼─────────────┤
| my-consumer  | RAW, GZIP       | Mon, 15 Aug 2022 16:00:00 MSK | 0         |
└──────────────┴─────────────────┴──────────────────────────────┴─────────────┘
```

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.

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 писатель 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.

те 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

`--wa`
- `--li`
- `--tr`

Поддерживает ли формат потоковый режим отбора	Значение лимита по умолчанию	Допустимые значения
Нет	10	1-500
Да	0 (без ограничений)	0-500

ителя - Да

делителя Да

`--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.

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`

в топик.

`none` — не выполнять преобразовании, 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.

- 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 выолнения 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:

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

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`.

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.

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

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 - 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):

```
┌──────────┬──────────────
```

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 в YDB SDK

Интерактивный режим выполнения запросов

Выполнив команду `ydb` без подкоманд, запустится интерактивный режим выполнения запросов. Консоль или терминал будут переведены в интерактивный режим. 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. Также между запусками сохраняется история запросов, доступны автодополнение команд и поиск по истории запросов.

Общий вид команды:

```
ydb [global options...]
```

- `global options` — глобальные параметры.

Горячие клавиши

Горячая клавиша	Описание
`CTRL + D`	Позволяет выйти из интерактивного режима.

ачены для выполнения различных

Команда	Описание
`SET param = value`	Команда `SET` устанавливает значение внутренней переменной `param` в `value`.

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.

алентна команде ydb sql --explain.

е с AST. Эквивалентна команде ydb sql --explain-ast.

```
| 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 скрипта (с поддержкой стриминга)

Важно

Данная команда устарела.
Рекомендуемый инструмент для выполнения запросов в YDB CLI — это команда ydb sql.

С помощью подкоманды `yql` вы можете выполнить YQL-скрипт. Скрипт может содержать запросы разных типов. В отличие от `scripting yql`, подкоманда `yql` устанавливает стрим и получает данные через него. Выполнение запроса в стриме позволяет снять ограничение на размер читаемых данных.

Общий вид команды:

```
ydb [global options...] yql [options...]
```

- `global options` — глобальные параметры.
- `options` — параметры подкоманды.

Посмотрите описание команды выполнения YQL-скрипта:

```
ydb yql --help
```

Параметры подкоманды

Имя	Описание

Имя	Описание
`-p, --param`	Значение одного параметра YQL-запроса в формате `$name=value`, где `$name` — имя параметра, а `value` — его значение (корректный JSON value).
`--param-file`	Имя файла в формате JSON в кодировке UTF-8, в котором заданы значения параметров, сопоставляемые с параметрами YQL-запроса по именам ключей.
`--input-format`	Формат представления значений параметров. Действует на все способы их передачи (через параметр команды, файл или `stdin`). Возможные значения: • `json-unicode` (по умолчанию) — JSON. • `json-base64` — JSON, в котором значения параметров с типом «бинарная строка» (`DECLARE $par AS String`) представлены в кодировке Base64.
`--stdin-format`	Формат представления параметров и фрейминг для `stdin`. Чтобы задать оба значения, укажите параметр дважды. **Формат представления параметров на `stdin`** Возможные значения: • `json-unicode` — JSON. • `json-base64` — JSON, в котором значения параметров с типом «бинарная строка» (`DECLARE $par AS String`) представлены в кодировке Base64. • `raw` — бинарные данные, имя параметра задается опцией `--stdin-par`. • `csv` — формат CSV. • `tsv` — формат TSV. Если формат представления параметров на `stdin` не задан, то применяется формат, заданный параметром `--input-format`. **Разделение наборов параметров (фрейминг) для `stdin`** Возможные значения: • `no-framing` (по умолчанию) — фрейминг не применяется • `newline-delimited` — символ перевода строки отмечает на `stdin` окончание одного набора параметров, отделяя его от следующего.
`--columns`	Строка с именами колонок, заменяющими header CSV/TSV документа, читаемого со stdin'а. Имена колонок должны быть в том же формате, что и сам документ.

`--skip-rows`	Число строк с начала данных, читаемых со stdin'а, которые нужно пропустить, не включая строку header'а.
`--stdin-par`	Имя параметра, значение которого будет передано через `stdin`, указывается без символа `$`.
`--batch`	Режим пакетирования значений наборов параметров, получаемых через `stdin`. Возможные значения: • `iterative` (по умолчанию) — пакетирование выключено • `full` - полный пакет • `adaptive` - адаптивное пакетирование
`--batch-limit`	Максимальное количество наборов параметров в пакете для адаптивного режима пакетирования. Установка в `0` снимает ограничение. Значение по умолчанию — `1000`.
`--batch-max-delay`	Максимальная задержка отправки на обработку полученного набора параметров для адаптивного режима пакетирования. Задается в виде числа с размерностью времени - `s`, `ms`, `m`. Значение по умолчанию — `1s` (1 секунда).

- `csv` — вывод в формате CSV;

Имя	Описание

Имя	Описание
`-p, --param`	Значение одного параметра YQL-запроса в формате `$name=value`, где `$name` — имя параметра, а `value` — его значение (корректный JSON value).
`--param-file`	Имя файла в формате JSON в кодировке UTF-8, в котором заданы значения параметров, сопоставляемые с параметрами YQL-запроса по именам ключей.
`--input-format`	Формат представления значений параметров. Действует на все способы их передачи (через параметр команды, файл или `stdin`). Возможные значения: • `json-unicode` (по умолчанию) — JSON. • `json-base64` — JSON, в котором значения параметров с типом «бинарная строка» (`DECLARE $par AS String`) представлены в кодировке Base64.
`--stdin-format`	Формат представления параметров и фрейминг для `stdin`. Чтобы задать оба значения, укажите параметр дважды. **Формат представления параметров на `stdin`** Возможные значения: • `json-unicode` — JSON. • `json-base64` — JSON, в котором значения параметров с типом «бинарная строка» (`DECLARE $par AS String`) представлены в кодировке Base64. • `raw` — бинарные данные, имя параметра задается опцией `--stdin-par`. • `csv` — формат CSV. • `tsv` — формат TSV. Если формат представления параметров на `stdin` не задан, то применяется формат, заданный параметром `--input-format`. **Разделение наборов параметров (фрейминг) для `stdin`** Возможные значения: • `no-framing` (по умолчанию) — фрейминг не применяется • `newline-delimited` — символ перевода строки отмечает на `stdin` окончание одного набора параметров, отделяя его от следующего.
`--columns`	Строка с именами колонок, заменяющими header CSV/TSV документа, читаемого со stdin'а. Имена колонок должны быть в том же формате, что и сам документ.

`--skip-rows`	Число строк с начала данных, читаемых со stdin'а, которые нужно пропустить, не включая строку header'а.
`--stdin-par`	Имя параметра, значение которого будет передано через `stdin`, указывается без символа `$`.
`--batch`	Режим пакетирования значений наборов параметров, получаемых через `stdin`. Возможные значения: • `iterative` (по умолчанию) — пакетирование выключено • `full` - полный пакет • `adaptive` - адаптивное пакетирование
`--batch-limit`	Максимальное количество наборов параметров в пакете для адаптивного режима пакетирования. Установка в `0` снимает ограничение. Значение по умолчанию — `1000`.
`--batch-max-delay`	Максимальная задержка отправки на обработку полученного набора параметров для адаптивного режима пакетирования. Задается в виде числа с размерностью времени - `s`, `ms`, `m`. Значение по умолчанию — `1s` (1 секунда).

результат выводится в виде массива строк JSON, каждая строка JSON выводится в отдельной строке;

- `parquet`: вывод в формате Apache Parquet.

Имя	Описание

Имя	Описание
`-p, --param`	Значение одного параметра YQL-запроса в формате `$name=value`, где `$name` — имя параметра, а `value` — его значение (корректный JSON value).
`--param-file`	Имя файла в формате JSON в кодировке UTF-8, в котором заданы значения параметров, сопоставляемые с параметрами YQL-запроса по именам ключей.
`--input-format`	Формат представления значений параметров. Действует на все способы их передачи (через параметр команды, файл или `stdin`). Возможные значения: • `json-unicode` (по умолчанию) — JSON. • `json-base64` — JSON, в котором значения параметров с типом «бинарная строка» (`DECLARE $par AS String`) представлены в кодировке Base64.
`--stdin-format`	Формат представления параметров и фрейминг для `stdin`. Чтобы задать оба значения, укажите параметр дважды. **Формат представления параметров на `stdin`** Возможные значения: • `json-unicode` — JSON. • `json-base64` — JSON, в котором значения параметров с типом «бинарная строка» (`DECLARE $par AS String`) представлены в кодировке Base64. • `raw` — бинарные данные, имя параметра задается опцией `--stdin-par`. • `csv` — формат CSV. • `tsv` — формат TSV. Если формат представления параметров на `stdin` не задан, то применяется формат, заданный параметром `--input-format`. **Разделение наборов параметров (фрейминг) для `stdin`** Возможные значения: • `no-framing` (по умолчанию) — фрейминг не применяется • `newline-delimited` — символ перевода строки отмечает на `stdin` окончание одного набора параметров, отделяя его от следующего.
`--columns`	Строка с именами колонок, заменяющими header CSV/TSV документа, читаемого со stdin'а. Имена колонок должны быть в том же формате, что и сам документ.

`--skip-rows`	Число строк с начала данных, читаемых со stdin'а, которые нужно пропустить, не включая строку header'а.
`--stdin-par`	Имя параметра, значение которого будет передано через `stdin`, указывается без символа `$`.
`--batch`	Режим пакетирования значений наборов параметров, получаемых через `stdin`. Возможные значения: • `iterative` (по умолчанию) — пакетирование выключено • `full` - полный пакет • `adaptive` - адаптивное пакетирование
`--batch-limit`	Максимальное количество наборов параметров в пакете для адаптивного режима пакетирования. Установка в `0` снимает ограничение. Значение по умолчанию — `1000`.
`--batch-max-delay`	Максимальная задержка отправки на обработку полученного набора параметров для адаптивного режима пакетирования. Задается в виде числа с размерностью времени - `s`, `ms`, `m`. Значение по умолчанию — `1s` (1 секунда).
`--tx-mode`	Режим транзакций (для запросов типа `data`). Возможные значения: • `serializable-rw` — результат успешно выполненных параллельных транзакций эквивалентен определенному последовательному порядку их выполнения; • `online-ro` — каждое из чтений в транзакции читает последние на момент своего выполнения данные; • `stale-ro` — чтения данных в транзакции возвращают результаты с возможным отставанием от актуальных (доли секунды).Значение по умолчанию — `serializable-rw`.
`-q`, `--query`	Текст YQL-запроса для выполнения.
`-f`, `--file`	Путь к файлу с текстом YQL-запроса для выполнения.

`--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.

Имя	Описание
`-p, --param`	Выражение в формате `$name=value`, где `$name` — имя параметра YQL-запроса, а `value` — его значение (корректный JSON value). Может быть указано несколько раз. Все указываемые параметры должны быть декларированы в YQL-запросе оператором DECLARE, иначе будет выдана ошибка «Query does not contain parameter». Если один и тот же параметр указан несколько раз, будет выдана ошибка «Parameter value found in more than one source». В зависимости от используемой операционной системы может понадобиться экранирование символа `$` или запись выражения в одинарных кавычках `'`.
`--param-file`	Имя файла в формате JSON в кодировке UTF-8, в котором заданы значения параметров, сопоставляемые с параметрами YQL-запроса по именам ключей. Может быть указано несколько раз. Если значения декларированного в YQL-запросе параметра будут обнаружены одновременно в нескольких файлах, или заданы в командной строке опцией `--param`, будет выдана ошибка «Parameter value found in more than one source». Имена ключей в JSON-файле указываются без начального символа `$`. Ключи, которые присутствуют в файле, но не декларированы в YQL-запросе, будут проигнорированы без выдачи сообщения об ошибке.
`--input-format`	Формат представления значений параметров. Действует на все способы их передачи (через параметр команды, файл или `stdin`). Возможные значения: • `json-unicode` (по умолчанию) — JSON. • `json-base64` — JSON, в котором значения параметров с типом «бинарная строка» (`DECLARE $par AS String`) представлены в кодировке Base64. Такая возможность позволяет передавать бинарные данные, декодирование которых из Base64 выполнит YDB CLI.

`--stdin-format`	Формат представления значений параметров на `stdin`. YDB CLI автоматически определяет что на стандартное устройство ввода `stdin` перенаправлен файл или выход другой команды командной оболочки, и в этом случае интерпретирует полученные данные в соответствии с возможными значениями: • `json-unicode` — JSON. • `json-base64` — JSON, в котором значения параметров с типом «бинарная строка» (`DECLARE $par AS String`) представлены в кодировке Base64. • `raw` — бинарные данные. • `csv` — формат CSV. По умолчанию имена параметров должны находиться в header'е CSV файла. При единичном исполнении запроса допустима только одна строка в файле, не считая header'а. • `tsv` — формат TSV. Если формат представления параметров на `stdin` не задан, то применяется формат, заданный параметром `--input-format`.
`--columns`	Строка с именами колонок, заменяющими header CSV/TSV документа, читаемого со stdin'а. Имена колонок должны быть в том же формате, что и сам документ. При указании опции считается, что header отсутствует. Опция допустима только с форматами stdin'а CSV и TSV.
`--skip-rows`	Число строк с начала данных, читаемых со stdin'а, которые нужно пропустить, не включая строку header'а, если она имеется. Опция допустима только с форматами stdin'а CSV и TSV.
`--stdin-par`	Имя параметра, значение которого передано через `stdin`. Указывается без символа `$`. Обязательно при использовании формата `raw` в `--stdin-format`. При использовании с JSON-форматами `stdin` интерпретируется не как JSON-документ, а как JSON value, с передачей значения в параметр с указанным именем.

Прямая передача бинарного контента

```
curl -Ls http://ydb.tech/docs | ydb -p quickstart table query execute \
  -q 'declare $a as String;
      select LEN($a)' \
  --stdin-format raw \
  --stdin-par a
```

Вывод команды (точное количество байт может отличаться):

```
┌──────────┐
| column0 |
├──────────┤
| 66426   |
└──────────┘
```

Передача файла в формате CSV

```
echo '10,Some text' | ydb -p quickstart table query execute \
  -q 'declare $a as Int32;
      declare $b as String
      select $a, $b' \
  --stdin-format csv \
  --columns 'a,b'
```

Вывод команды:

```
┌──────────┬──────────────┐
| column0 | column1     |
├──────────┼──────────────┤
| 10      | "Some text" |
└──────────┴──────────────┘
```

Итеративная потоковая обработка

YDB CLI поддерживает возможность многократного исполнения YQL-запроса с разными наборами значений параметров, при их передаче через `stdin`. 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 `stdin` 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.

YQL-запрос выполняется столько раз, сколько наборов значений параметров получено через `stdin`. Каждый полученный на `stdin` набор объединяется со значениями параметров, определенными через другие источники (`--param`, `--param-file`). Исполнение команды будет завершено после завершения потока на `stdin`. Каждый запрос исполняется в своей транзакции.

Правило отделения наборов параметров друг от друга (фрейминг) дополняет описание формата представления параметров на `stdin`, задаваемое параметром `--stdin-format`:

Имя	Описание
`--stdin-format`	Задает фрейминг для `stdin`. Возможные значения: • `no-framing` (по умолчанию) — на `stdin` ожидается один набор параметров, YQL-запрос исполняется однократно после завершения чтения `stdin`. • `newline-delimited` — символ перевода строки отмечает на `stdin` окончание одного набора параметров, отделяя его от следующего. YQL-запрос исполняется каждый раз при получении на `stdin` символа перевода строки.

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}' > par1.txt
cat 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 table query execute \
  -q 'declare $a as Int64;
      declare $b as Int64;
      select $a+$b' \
  --stdin-format 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 YQL-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' > par1.txt
cat 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 table query execute \
  -q 'declare $a as Int64;
      declare $b as Int64;
      select $a+$b, $a-$b' \
  --stdin-format newline-delimited \
  --stdin-format csv \
  --format csv
```

Вывод команды:

```
30,-10
40,-10
83,-13
```

Полученный в таком формате результат можно использовать для передачи на вход команде исполнения следующего YQL-запроса, задав header данным в CSV формате опцией `--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\trow1\n15\t"row\t2"\n35\t"""row""\n3"' > par1.txt
cat par1.txt
```

Вывод команды:

```
a  b
10  row1
15  "row  2"
35  """row""
3"
```

Исполним запрос, передав на `stdin` содержимое данного файла, с форматированием вывода в TSV:

```
cat par1.txt | \
ydb -p quickstart table query execute \
  -q 'declare $a as Int64;
      declare $b as Utf8;
      select $a, $b' \
  --stdin-format newline-delimited \
  --stdin-format tsv \
  --format tsv
```

Вывод команды:

```
10  "row1"
15  "row\t2"
35  "\"row\"\n3"
```

Полученный в таком формате результат можно использовать для передачи на вход команде исполнения следующего YQL-запроса, задав header данным в TSV формате опцией `--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 table query execute \
  -q 'declare $a as Int64;
      declare $b as Int64;
      select $a+$b as sum1' \
  --param '$b=100' \
  --stdin-format newline-delimited \
  --stdin-par 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
- Адаптивный

Полный режим пакетирования

Полный (`full`) режим является упрощенным вариантом пакетирования, когда запрос выполняется один раз, с заворачиванием в `List<>` всех полученных на `stdin` 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 основных 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 YQL-запроса, и выбрать режим следующим параметром:

Имя	Описание

Имя	Описание
`--batch-limit`	Максимальное количество наборов параметров в пакете для адаптивного режима пакетирования. Следующий пакет будет отправлен на исполнение YQL-запросом, если количество наборов данных в нем достигло указанного значения. Установка в `0` снимает ограничение. Значение по умолчанию — `1000` . Параметры передаются в YQL без стриминга и общий объем одного GRPC-запроса, в который включаются значения параметров, имеет верхнюю границу около 5 МБ.

`--batch-max-delay`	Максимальная задержка отправки на обработку полученного набора параметров для адаптивного режима пакетирования. Задается в виде числа с размерностью времени - `s`, `ms`, `m`. 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`.

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 YQL-запроса. Команда исполнения YQL-запроса, в свою очередь, отображает пакеты параметров в каждом следующем вызове YQL-запроса.

```
for i in $(seq 1 1000); do echo "Line$i"; sleep 0.2; done | \
ydb -p quickstart table query execute \
  -q 'declare $x as List<Utf8>;
      select ListLength($x), $x' \
  --stdin-format newline-delimited \
  --stdin-format raw \
  --stdin-par x \
  --batch adaptive
```

Вывод команды (точные значения могут отличаться):

```
┌──────────┬──────────────
```

```
{"count":20,"p2":10,"items":
["Line1","Line2","Line3","Line4","Line5","Line6","Line7","Line8","Line9","Line10","Line11","Line12","Line13","Line14","Line
{"count":20,"p2":10,"items":
["Line21","Line22","Line23","Line24","Line25","Line26","Line27","Line28","Line29","Line30","Line31","Line32","Line33","Line
...
{"count":20,"p2":10,"items":
["Line161","Line162","Line163","Line164","Line165","Line166","Line167","Line168","Line169","Line170","Line171","Line172","L
{"count":20,"p2":10,"items":
["Line181","Line182","Line183","Line184","Line185","Line186","Line187","Line188","Line189","Line190","Line191","Line192","L
```

Удаление множества записей из строковой таблицы YDB по первичным ключам

> **Внимание**
>
> Поддерживается только для строковых таблиц. Поддержка функциональности для колоночных таблиц находится в разработке.

Данный пример показывает как можно удалять неограниченное количество записей из строковых таблиц YDB, не рискуя превысить ограничения на количество записей в транзакции.

Создадим тестовую строковую таблицу:

```
ydb -p quickstart yql -s 'create table test_delete_1( id UInt64 not null, primary key (id))'
```

Занесем в неё 100000 записей:

```
for i in $(seq 1 100000); do echo "$i"; done | \
ydb -p quickstart import file csv -p test_delete_1
```

Удалим все записи с ID > 10:

```
ydb -p quickstart table query execute -t scan \
  -q 'select t.id from test_delete_1 as t where t.id > 10' \
  --format json-unicode | \
ydb -p quickstart table query execute \
  -q 'declare $lines as List<Struct<id:UInt64>>;
      delete from test_delete_1 where id in (select tl.id from AS_TABLE($lines) as tl)' \
  --stdin-format newline-delimited \
  --stdin-par lines \
  --batch adaptive \
  --batch-limit 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 YQL-запросы в YDB SDK

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:
  - `buildindex` — операции построения индекса;
  - `export/s3` — операции экспорта;
  - `import/s3` — операции импорта.

Посмотрите описание команды получения списка фоновых операций:

```
ydb operation list --help
```

Параметры подкоманды

Имя	Описание
`-s`, `--page-size`	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`	Токен страницы.
`--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.

```
┌──────────────────────────────────────────┬───────┬─────────┬───────┬──────────┬──────────────────────┬──────────────┐
| id                                       | ready | status  | state | progress | table                | index        |
├──────────────────────────────────────────┼───────┼─────────┼───────┼──────────┼──────────────────────┼──────────────┤
| ydb://buildindex/7?id=281489389055514 | true  | SUCCESS | Done  | 100.00%  | /my-database/series | idx_release |
└──────────────────────────────────────────┴───────┴─────────┴───────┴──────────┴──────────────────────┴──────────────┘

Next page token: 0
```

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.

ой 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_release |
└──────────────────────────────────────────┴───────┴─────────┴───────┴──────────┴──────────────────────┴──────────────┘
```

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.
   cloud.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-for-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 profiles
```

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 #rate_limiter
#locking #kesus
grpcs://vm-etn01q5-arum.etn01ftr.ydb.mdb.yandexcloud.net:2135 [vla] #table_service #scripting #discovery #rate_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 воспринимает запросы сервер:

```
ydb [connection options] discovery whoami [-g]
```

, где `[connection options]` — опции соединения с БД

В ответ выводится имя учетной записи (User SID) и, если указана опция `-g`, то информация о принадлежности учетной записи группам.

Если на сервере YDB не включена аутентификация (что может применяться, например, при самостоятельном локальном развертывании), то выполнение команды завершится с ошибкой.

Поддержка опции `-g` зависит от конфигурации сервера. Если она не включена, то вы будете получать в ответ `User has no groups` вне зависимости от фактического включения вашей учетной записи в какие-либо группы.

Пример

```
$ ydb -p quickstart discovery whoami -g
User SID: aje5kkjdgs0puc18976co@as

User has no groups
```

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.

ности новой версии. Если хост, на котором выполняется
и выводу предупреждения при выполнении команды.

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`.

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-H - TPC-H бенчмарк.
- TPC-DS - TPC-DS бенчмарк.
- Topic - Topic нагрузка.
- Transfer - Transfer нагрузка.

Stock нагрузка

Внимание

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 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.

```
    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
```

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.

Параметры для user-hist

Имя параметра	Короткое имя	Описание параметра
`--limit <значение>`	`-l <значение>`	Необходимое количество заказов. Значение по умолчанию: 10.

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.

1 штуке. Количество

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.

ных товаров по
аза 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.

в уменьшении

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.

```
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`

```
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	`row`
`--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	

Имя	Описание	Значение по умолчанию
`--upload-threads <значение>` или `-t <значение>`	Количество потоков исполнения для подготовки данных.	Соответствует количеству доступных ядер на клиентской машине.
`--bulk-size <значение>`	Размер порции для отправки данных в строках.	10000
`--max-in-flight <значение>`	Максимально количество порций данных, одновременно находящихся в обработке.	128

Общие параметры для всех видов нагрузки

Имя	Описание	Значение по умолчанию

`--ext-queries <запросы>` или `-q <запросы>`	Внешние запросы для выполнения нагрузки, разделенные точкой с запятой.	
`--ext-queries-file <имя>`	Имя файла, в котором можно указать внешние запросы для выполнения нагрузки, разделенные точкой с запятой.	
`--ext-query-dir <имя>`	Директория с внешними запросами для выполнения нагрузки. Запросы должны лежать в файлах с именами `q[0-42].sql`.	
`--ext-results-dir <имя>`	Директория с внешними результатами запросов для сравнения. Результаты должны лежать в файлах с именами `q[0-42].sql`.	
`--check-canonical` или `-c`	Использовать специальные детерминированные внутренние запросы и сверять результаты с каноническими.	
`--retries`	Количество повторных попыток выполнения каждого запроса	`0`
`--include`	Номера или отрезки номеров запросов, которые нужно выполнить в рамках нагрузки. Указываются через запятую, например `1,2,4-6`.	Все запросы
`--exclude`	Номера или отрезки номеров запросов, которые нужно исключить в рамках нагрузки. Указываются через запятую, например `1,2,4-6`.	
`--executer`	Механизм выполнения запросов, доступные значения: `scan`, `generic`.	`generic`
`--verbose` или `-v`	Выводить больше информации на экран в процессе выполнения запросов.	
`--threads <значение>` или `-t <значение>`	Количество потоков, генерирующих нагрузку	

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.

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: .
Имя параметра	Описание параметра	
`--len`	Размер строк в байтах, которые вставляются в таблицу, как values. Значение по умолчанию: 8.	
`--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.

- `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.

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`.

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.

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`.

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`.

Имя параметра	Описание параметра
`--topic`	Имя топика. Значение по умолчанию: `workload-topic`.

	`consumer`. `--consumers` равно `2` и префикс `--consumer-prefix` равен `ьзованы 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`.

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`

елей создаются по правилу
араметра `--consumers` минус 1.

```
ic-partitions 143 --table-partitions
```

с 237 партициями. Имена читателей

ает 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`

- `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`

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.	`/`

```
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`.	`row`
`--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`,	`Date32`,

Имя	Описание	Значение по умолчанию

Имя	Описание	Значение по умолчанию
`--upload-threads <значение>` или `-t <значение>`	Количество потоков исполнения для подготовки данных.	Соответствует количеству доступных ядер на клиентской машине.
`--bulk-size <значение>`	Размер порции для отправки данных в строках.	10000
`--max-in-flight <значение>`	Максимально количество порций данных, одновременно находящихся в обработке.	128

`-i <значение>`	процессов, этот параметр задает номер процесса.	
`--state <путь>`	Путь к файлу состояния загрузки. Если загрузка была прервана по какой-то причине, при новом запуске загрузка будет	

Имя	Описание	Значение по умолчанию

Имя	Описание	Значение по умолчанию
`--ext-query-dir <имя>`	Директория с внешними запросами для выполнения нагрузки. Запросы должны лежать в файлах с именами `q[1-23].sql`.	

	формате `json`.	сохраняется

`--ministat <имя>`	Имя файла, в котором будет сохранена статистика выполнения запросов в формате `ministat`.	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 сохраняется.
`--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`	Номера или отрезки номеров запросов, которые нужно выполнить в апример `1,2,4-6`.	Все запросы

Имя	Описание	Значение по умолчанию
`--path` или `-p`	Путь к каталогу с таблицами.	`/`

	е нужно исключить в апример `1,2,4-6`.	
`--executer`	Механизм выполнения запросов, доступные значения: `scan`, `generic`.	`generic`
`--verbose` или `-v`	Выводить больше информации на экран в процессе выполнения запросов.	
`--threads <значение>` или `-t <значение>`	Количество потоков, генерирующих нагрузку	

Доступные параметры

Имя	Описание	Значение по умолчанию
`--store <значение>`	Тип хранилища таблиц. Возможные значения: `row`, `column`, `external-s3`.	`row`
`--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`,	`Date32`,

Имя	Описание	Значение по умолчанию

Имя	Описание	Значение по умолчанию
`--upload-threads <значение>` или `-t <значение>`	Количество потоков исполнения для подготовки данных.	Соответствует количеству доступных ядер на клиентской машине.
`--bulk-size <значение>`	Размер порции для отправки данных в строках.	10000
`--max-in-flight <значение>`	Максимально количество порций данных, одновременно находящихся в обработке.	128

`-i <значение>`	процессов, этот параметр задает номер процесса.	
`--state <путь>`	Путь к файлу состояния загрузки. Если загрузка была прервана по какой-то причине, при новом запуске загрузка будет	

Имя	Описание	Значение по умолчанию

Имя	Описание	Значение по умолчанию
`--ext-query-dir <имя>`	Директория с внешними запросами для выполнения нагрузки. Запросы должны лежать в файлах с именами `q[1-99].sql`.	

	формате `json`.	сохраняется

`--ministat <имя>`	Имя файла, в котором будет сохранена статистика выполнения запросов в формате `ministat`.	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 сохраняется.
`--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`	Номера или отрезки номеров запросов, которые нужно выполнить в рамках нагрузки. Указываются через запятую, например `1,2,4-6`.	Все запросы
`--exclude`	Номера или отрезки номеров запросов, которые нужно исключить в рамках нагрузки. Указываются через запятую, например `1,2,4-6`.	
`--executer`	Механизм выполнения запросов, доступные значения: `scan`, `generic`.	`generic`
`--verbose` или `-v`	Выводить больше информации на экран в процессе выполнения запросов.	
`--threads <значение>` или `-t <значение>`	Количество потоков, генерирующих нагрузку	

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)

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 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.

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.

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.

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.

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.

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.

C# (.NET)

```
dotnet add package Ydb.Sdk
```

Java

Добавьте зависимости в Maven-проект, как описано в пункте "Install the SDK" файла `readme.md` в репозитории исходного кода.

PHP

```
composer require ydb-platform/ydb-php-sdk
```

Node.JS

```
npm install ydb-sdk
```

Rust

```
cargo add ydb
```

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.

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()`

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	

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.

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 кэшировать план запроса для параметризованных запросов. Это радикально снижает потребление CPU и повышает пропускную способность системы;
- Использование параметризованных запросов спасает от уязвимостей вида SQL Injection.

YDB SDK автоматически кэшируют планы параметризованных запросов по умолчанию, для этого обычно используется настройка `KeepInCache = true`.

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

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);
```

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 нужно создать экземпляр драйвера YDB.

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.

Фрагмент кода приложения для инициализации драйвера YDB:

```
var config = new DriverConfig(
    endpoint: "grpc://localhost:2136",
    database: "/local"
);

await using var driver = await Driver.CreateInitialized(
    config: config,
    loggerFactory: loggerFactory
);
```

В этом примере используется анонимная аутентификация. Подробнее про соединение с базой данных и аутентификацию.

Фрагмент кода приложения для создания различных клиентов к топикам:

```
var topicClient = new TopicClient(driver);

await using var writer = new WriterBuilder<string>(driver, topicName)
{
    ProducerId = "ProducerId_Example"
}.Build();

await using var reader = new ReaderBuilder<string>(driver)
{
    ConsumerName = "Consumer_Example",
    SubscribeSettings = { new SubscribeSettings(topicName) }
}.Build();
```

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

```
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
    }
});
```

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

```
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());
```

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

```
info = 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();
```

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

```
driver.topic_client.drop_topic(topic_path)
```

Java

```
topicClient.dropTopic(topicPath);
```

C#

```
await topicClient.DropTopic(topicName);
```

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++

Подключение к топику на запись представлено объектом сессии записи с интерфейсом `IWriteSession` или `ISimpleBlockingWriteSession` (вариант для простой записи по одному сообщению без подтверждения, блокирующейся при превышении числа inflight записей или размера буфера SDK). Настройки сессии записи представлены структурой `TWriteSessionSettings`, для варианта `ISimpleBlockingWriteSession` часть настроек не поддерживается.

Полный список настроек смотри в заголовочном файле.

Пример создания сессии записи с интерфейсом `IWriteSession`.

```
std::string producerAndGroupID = "group-id";
auto settings = NYdb::NTopic::TWriteSessionSettings()
    .Path("my-topic")
    .ProducerId(producerAndGroupID)
    .MessageGroupId(producerAndGroupID);

auto session = topicClient.CreateWriteSession(settings);
```

Go

```
producerAndGroupID := "group-id"
writer, err := db.Topic().StartWriter(producerAndGroupID, "topicName",
  topicoptions.WithMessageGroupID(producerAndGroupID),
)
if err != nil {
```

```
    return err
}
```

Python

```
writer = driver.topic_client.writer(topic_path)
```

Java (sync)

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 succsessfully");
  } catch (Exception exception) {
      logger.error("Exception while initializing writer: ", exception);
      return;
  }
  ```

Java (async)

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>(driver, topicName)
{
    ProducerId = "ProducerId_Example"
}.Build();
```

Запись сообщений

C++

Асинхронная запись возможна через интерфейс `IWriteSession`.

Работа пользователя с объектом `IWriteSession` в общем 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, которую невозможно повторить, сессия чтения 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;
    }
}
```

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, которую невозможно повторить 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
}
```

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.

```
# 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(),
  ])
```

Java (sync)

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", timeoutSeconds);
} catch (InterruptedException | ExecutionException exception) {
    logger.error("Couldn't put the message into sending queue due to exception: ", exception);
}
```

Java (async)

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>
```

Запись сообщений с подтверждением о сохранении на сервере

C++

Получение подтверждений от сервера возможно через интерфейс `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.

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.

```
# 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")
```

Java (async)

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);
```

Выбор кодека для сжатия сообщений

Подробнее о сжатии данных в топиках.

C++

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.

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();
```

Запись сообщений без дедупликации

Подробнее о записи без дедупликации — в соответствующем разделе концепций.

C++

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".

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++

Воспользоваться функцией записи метаданных можно с помощью метода `Write()`, принимающего `TWriteMessage` объект:

```
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));
}
```

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:

```
message = ydb.TopicWriterMessage(data=f"message-data", metadata_items={"meta-key": "meta-value"})
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()) } }
);
```

Запись в транзакции

C++

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::TStatus {
    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();
}));
```

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

```
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)
```

Python (asyncio)

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

```
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 (sync)

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);
```

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`.

Java (async)

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`.

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:

```
reader = driver.topic_client.reader(topic="my-topic", consumer="my-consumer")
```

Java (sync)

Инициализация настроек читателя

```
ReaderSettings settings = ReaderSettings.newBuilder()
        .setConsumerName(consumerName)
        .addTopic(TopicReadSettings.newBuilder()
                .setPath(topicPath)
                .setReadFrom(Instant.now().minus(Duration.ofHours(24))) // Optional
                .setMaxLag(Duration.ofMinutes(30)) // Optional
                .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 succsessfully");
  } catch (Exception exception) {
      logger.error("Exception while initializing reader: ", exception);
      return;
  }
  ```

Java (async)

Инициализация настроек читателя

```
ReaderSettings settings = ReaderSettings.newBuilder()
        .setConsumerName(consumerName)
        .addTopic(TopicReadSettings.newBuilder()
                .setPath(topicPath)
                .setReadFrom(Instant.now().minus(Duration.ofHours(24))) // Optional
                .setMaxLag(Duration.ofMinutes(30)) // Optional
                .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.
Для реализации объекта-наследника ReadEventHandler можно воспользоваться дефолтным абстрактным классом `AbstractReadEventHandler`.
Достаточно переопределить метод onMessages, отвечающий за обработку самих сообщений. 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>(driver)
{
    ConsumerName = "Consumer_Example",
    SubscribeSettings = { new SubscribeSettings(topicName) }
}.Build();
```

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>(driver)
{
    ConsumerName = "Consumer_Example",
    SubscribeSettings =
    {
        new SubscribeSettings(topicName),
        new SubscribeSettings(topicName + "_another") { ReadFrom = DateTime.Now }
    }
}.Build();
```

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.

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

```
while True:
    message = reader.receive_message()
    process(message)
```

Java (sync)

Чтобы читать сообщения без подтверждения обработки, по одному, используйте следующий код:

```
while(true) {
    Message message = reader.receive();
    process(message);
}
```

Java (async)

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)
{
}
```

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

```
while True:
  batch = reader.receive_batch()
  process(batch)
```

Java (sync)

В синхронном клиенте нет возможности прочитать сразу пакет сообщений.

Java (async)

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)
{
}
```

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.

```
while True:
    message = reader.receive_message()
    process(message)
    reader.commit(message)
```

`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

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)
{
}
```

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

```
while True:
  batch = 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 (sync)

Неактуально, т.к. в синхронном читателе нет возможности читать сообщения пакетами.

Java (async)

В обработчике `onMessage` 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)
{
}
```

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

```
func ReadWithExplicitPartitionStartStopHandlerAndOwnReadProgressStorage(ctx context.Context, db ydb.Connection) 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.

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.

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(),
)
```

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.

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)
```

Python (asyncio)

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.

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 (sync)

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.

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`.

Java (async)

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);
    }
}
```

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.

```
while True:
  batch = reader.receive_batch()
  process(batch)
  reader.commit(batch)
```

Java (sync)

Неактуально, т.к. в синхронном читателе нет возможности настраивать обработку подобных событий.
Клиент сразу ответит серверу подтверждением остановки.

Java (async)

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();
}
```

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::TPartitionSessionClosedEvent>
(&*event)) {
    if (partitionSessionClosedEvent->GetReason() == NYdb::NTopic::TPartitionSessionClosedEvent::EReason::ConnectionLost)
{
        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.

```
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)
```

Java (sync)

Неактуально, т.к. в синхронном читателе нет возможности настраивать обработку подобных событий.

Java (async)

```
@Override
public void onPartitionSessionClosed(PartitionSessionClosedEvent event) {
    logger.info("Partition session {} is closed.", event.getPartitionSession().getPartitionId());
}
```

Поддержка автомасштабирования топиков

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`:

```
    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.

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.

Python

Подтверждения обработки вне читателя производится с помощью метода `topic_client.commit_offset`:

```
driver.topic_client.commit_offset(
    topic_path,
    consumer_name,
    partition_id,
    offset,
)
```

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 семафоры.

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.

Go

```
err := db.Coordination().CreateNode(ctx,
    "/path/to/mynode",
)
```

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`.

Работа с сессиями

Создание сессии

Для начала работы клиент должен установить сессию, в рамках которой он будет осуществлять все операции с узлом координации.

Go

```
session, err := db.Coordination().CreateSession(ctx,
    "/path/to/mynode", // имя Coordination Node в базе
)
```

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.

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.

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.

C++

В C++ SDK установленная сессия в фоне поддерживает и автоматически восстанавливает связь с кластером YDB.

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.

Go

```
err := session.CreateSemaphore(ctx,
    "my-semaphore", // semaphore name
    10              // semaphore limit
)
```

C++

```
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:

```
session
    .CreateSemaphore(
        "my-semaphore",  // semaphore name
        10,              // semaphore limit
        "my-data"        // semaphore data
    )
    .ExtractValueSync()
    .ExtractResult();
```

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.

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`.

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>()`).

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.

Go

```
err := session.UpdateSemaphore(
    "my-semaphore",                                                  // semaphore name
    options.WithUpdateData([]byte("updated-data")),   // new semaphore data
)
```

C++

```
session
    .UpdateSemaphore(
        "my-semaphore",  // semaphore name
        "updated-data"   // new semaphore data
    )
    .ExtractValueSync()
    .ExtractResult();
```

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

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
)
```

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`.

Освобождение семафора

Go

Чтобы отпустить захваченный в сессии семафор, необходимо вызвать метод `Release` у объекта `Lease`.

```
err := lease.Release()
```

C++

```
session
    .ReleaseSemaphore(
        "my-semaphore"  // semaphore name
    )
    .ExtractValueSync()
    .ExtractResult();
```

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

- Статусы завершения gRPC
- Статусы завершения сервера YDB
- Ошибки

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'	–	+	+	+	+	–	–	–

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
Инициализация балансировщика через Discovery/ListEndpoints	+	+	+	+	+	+	+	+

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
Anonymous (по умолчанию)	+	+	+	+	+	+	+	+

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
Int/Uint(8,16,32,64)	+	+	+	+	+	+	+	+
Int128, UInt128	–	–	–	–	–	–	–	–

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
ListEndpoints	+	–	+	–	+	–	–	–

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
Пул сессий	+	+	+	+	–	+	–	–

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
MakeDirectory	+	+	+	+	+	–	–	+

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
Пул сессий	+	+	+	+	+	+	+	+

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
Consumed Units из метаданных	+	+	+	+	+	–	–	–

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
ExecuteYql	+	+	+	–	–	–	–	+

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
CreateNode	+	–	+	+	–	–	–	–

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
CreateTopic	+	+	+	+	+	+	–	–

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP
CreateResource	+	–	+	–	–	–	–	–

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP

Feature	C++	Python	Go	Java	NodeJS	C#	Rust	PHP	
Логирование событий SDK	–	–	+	+	+	+	+	+	–
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	–	+	+	+	–	–	–	–	
* 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)	–	–	+	–	+	–	–	–

Код	Статус	Возможность повтора	Стратегия задержек	Пересоздать сессию
400000	SUCCESS	–	–	–
400010	BAD_REQUEST	неповторяемый	–	нет
400020	UNAUTHORIZED	неповторяемый	–	нет
400030	INTERNAL_ERROR	неповторяемый	–	нет
400040	ABORTED	повторяемый	короткая	нет
400050	UNAVAILABLE	повторяемый	короткая	нет
400060	OVERLOADED	повторяемый	большая	нет
400070	SCHEME_ERROR	неповторяемый	–	нет
400080	GENERIC_ERROR	неповторяемый	–	нет
400090	TIMEOUT	неповторяемый	–	нет
400100	BAD_SESSION	повторяемый	моментально	да
400120	PRECONDITION_FAILED	неповторяемый	–	нет
400130	ALREADY_EXISTS	неповторяемый	–	нет
400140	NOT_FOUND	неповторяемый	–	нет
400150	SESSION_EXPIRED	повторяемый	моментально	да
400160	CANCELLED	неповторяемый	–	нет
400170	UNDETERMINED	условно повторяемый	короткая	нет

400180	UNSUPPORTED	неповторяемый	–	нет
400190	SESSION_BUSY	повторяемый	короткая	да
400200	EXTERNAL_ERROR	неповторяемый	–	нет

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 встроенном UI, который выполнялся слишком долго, или запрос был сделан с опцией таймаута cancel_after.

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

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	короткая	да
2	UNKNOWN	неповторяемый	–	да
3	INVALID_ARGUMENT	неповторяемый	–	да
4	DEADLINE_EXCEEDED	условно повторяемый	короткая	да
5	NOT_FOUND	неповторяемый	–	да
6	ALREADY_EXISTS	неповторяемый	–	да
7	PERMISSION_DENIED	неповторяемый	–	да
8	RESOURCE_EXHAUSTED	повторяемый	большая	нет
9	FAILED_PRECONDITION	неповторяемый	–	да
10	ABORTED	повторяемый	моментально	да
11	OUT_OF_RANGE	неповторяемый	–	нет
12	UNIMPLEMENTED	неповторяемый	–	да
13	INTERNAL	условно повторяемый	короткая	да
14	UNAVAILABLE	условно повторяемый	короткая	да
15	DATA_LOSS	неповторяемый	–	да

пространства, которые не

которые

ек. Также

об ошибке).

16	UNAUTHENTICATED	неповторяемый	–	да

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.

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 = TSelfCheckSettings();
settings.ReturnVerboseStatus(true);
auto result = client.SelfCheck(settings).GetValueSync();
```

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);
};
```

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.

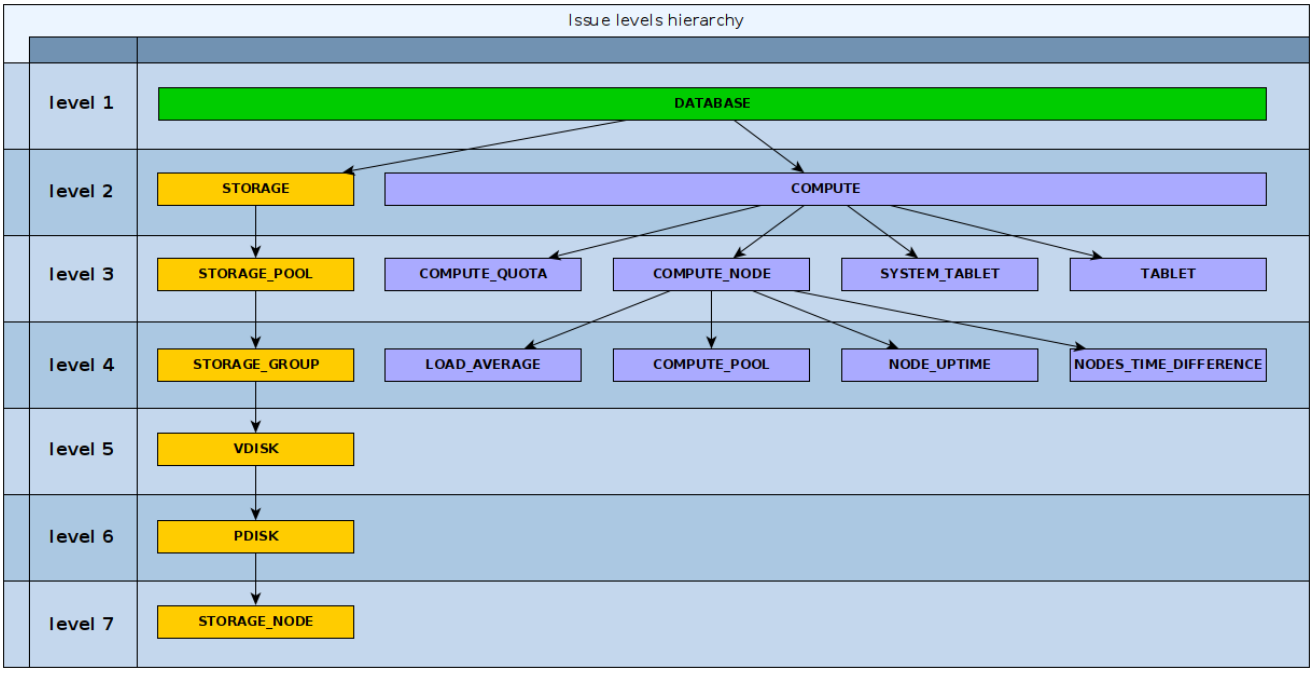
Если обнаружены проблемы, поле `issue_log` будет содержать описания этих проблем со следующей структурой:

```
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`.



Результат проверки базы данных

Наиболее общий статус базы данных может принимать следующие значения:

Поле	Описание
`GOOD`	Проблем не обнаружено.

Поле	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.

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 Embedded 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 Embedded 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 Embedded 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 Embedded 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 Embedded 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 Embedded 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 Embedded 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 Embedded 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 Embedded 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 Embedded 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: Стореджовая 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 Embedded 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 Embedded 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 Embedded 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
}
}
```

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

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

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>"
      ]
    }
  }
}
```

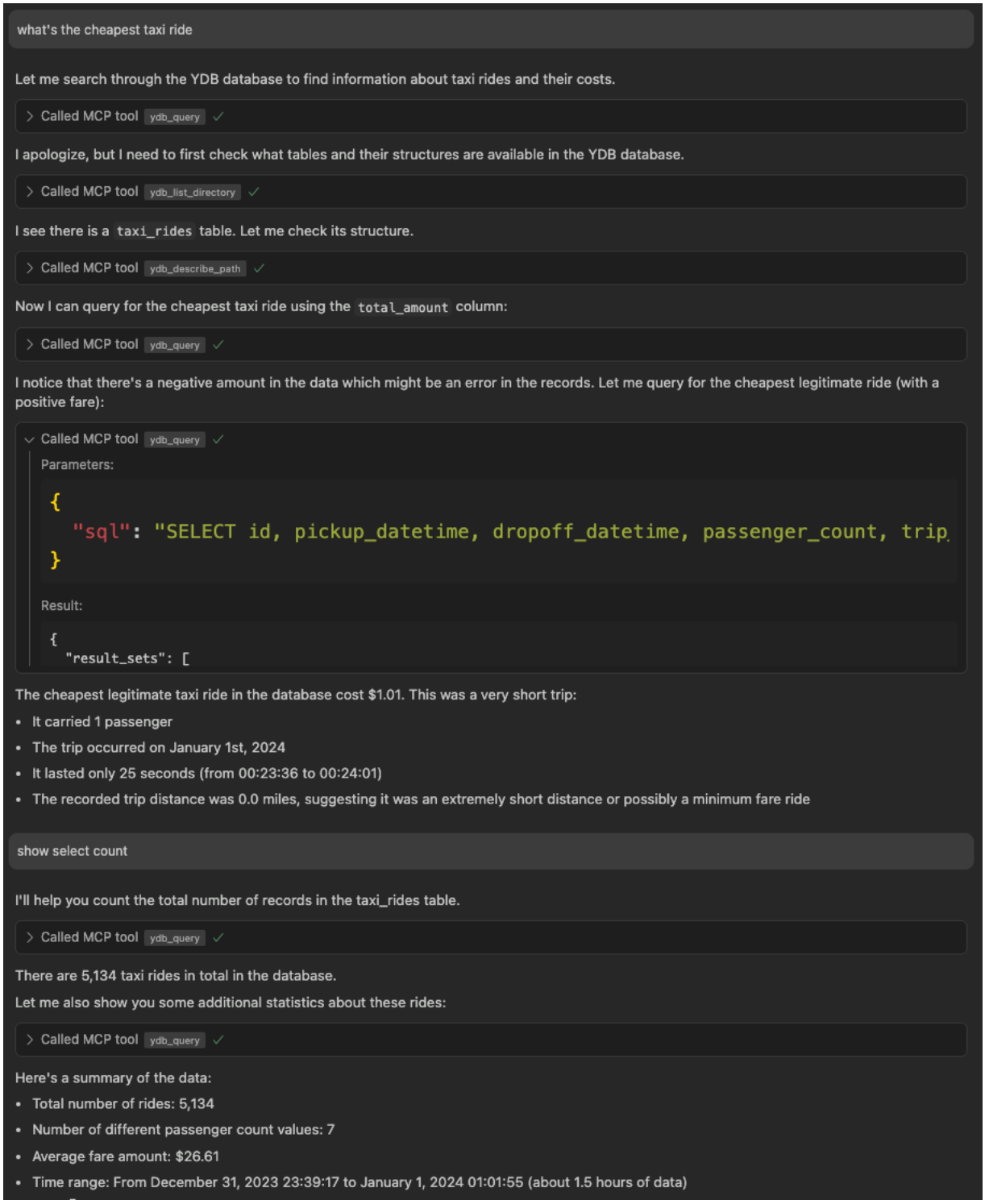
pip

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`

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 connection = new YdbConnection("Host=localhost;Port=2136;Database=/local;MaxSessionPool=50");
await connection.OpenAsync();

var ydbCommand = connection.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}, title:
{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.

Подключения

Подключение к YDB устанавливается через `YdbConnection`.

1. **Использование пустой строки подключения**:

   Следующий код создаёт подключение с настройками по умолчанию:

   ```
   await using var ydbConnection = new YdbConnection("");
   await ydbConnection.OpenAsync();
   ```

   Этот вариант создаёт подключение к базе данных по URL: `grpc://localhost:2136/local`, с анонимной аутентификацией.

2. **Использование конструктора со строкой подключения**:

   В следующем примере происходит создание подключения при помощи строки подключения в ADO.NET:

   ```
   await using var ydbConnection = new YdbConnection(
       "Host=database-sample-grpc;Port=2135;Database=/root/database-sample");
   await ydbConnection.OpenAsync();
   ```

   В данном случае подключение будет установлено по URL: `grpc://database-sample-grpc:2135/root/database-sample`. Поддерживаемый набор настроек описан на странице параметров подключения.

3. **Использование конструктора с аргументом `YdbConnectionStringBuilder`**:

   Вариант с использованием `YdbConnectionStringBuilder` демонстрируется в коде ниже:

   ```
   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();
   ```

   `YdbConnectionStringBuilder` поддерживает дополнительные настройки, помимо строки подключения, такие как логирование, расширенные параметры аутентификации.

Pooling

Открытие и закрытие логического соединения с YDB является дорогостоящим и трудоемким процессом. Поэтому соединения с YDB объединяются в пул. Закрытие или удаление соединения не проводит к закрытию основного логического соединения; скорее, она возвращается в пул, управляемый `Ydb.Sdk.Ado`. Когда соединение снова понадобится, будет возвращено соединение из пула. Это делает операции открытия и закрытия быстрыми. Не стесняйтесь часто открывать и закрывать соединения, если это необходимо, вместо того, чтобы без необходимости поддерживать соединение открытым в течении длительного периода времени.

ClearPool

Немедленно закрывает незанятые соединения. Активные соединения закрываются при возврате.

```
YdbConnection.ClearPool(ydbConnection)
```

ClearAllPools

Закрывает все незанятые соединения во всех пулах. Активные соединения закрываются при возврате.

```
YdbConnection.ClearAllPools()
```

Data Source

Начиная с .NET 7.0, отправной точкой для любой операции с базой данных является DbDataSource.

Самый простой способ создать источник данных заключается в следующем:

```
await using var dataSource = new YdbDataSource("Host=localhost;Port=2136;Database=/local");
```

Или

```
var ydbConnectionBuilder = new YdbConnectionStringBuilder
{
    Host = "localhost",
    Port = 2136,
    Database = "/local",
    UseTls = false
};

await using var dataSource = new YdbDataSource(ydbConnectionBuilder);
```

Базовое выполнение SQL

Как только у вас есть `YdbConnection`, для выполнения SQL-запроса можно использовать `YdbCommand`:

```
await using var command = dataSource.CreateCommand("SELECT some_field FROM some_table")
await using var reader = await command.ExecuteReaderAsync();

while (await reader.ReadAsync())
{
    Console.WriteLine(reader.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 command = dataSource.CreateCommand("INSERT INTO some_table (some_field) VALUES ('Hello YDB!'u)");
await command.ExecuteNonQueryAsync();
```

Параметры

При отправке пользовательских данных в базу данных всегда рекомендуется использовать параметры, а не включать значения в SQL следующим образом:

```
await using var connection = new YdbConnection(_cmdOptions.SimpleConnectionString);
await connection.OpenAsync();

var ydbCommand = connection.CreateCommand();
ydbCommand.CommandText = """
                         DECLARE $series_id AS Uint64;
                         DECLARE $season_id AS Uint64;
                         DECLARE $limit_size AS Uint64;

                         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. А тип будет определен согласно DbType или .NET тип самого 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

Чтобы создать клиентскую транзакцию, используйте стандартный метод ADO.NET `ydbConnection.BeginTransaction()`.

Есть две сигнатуры этого метода с единственным параметром уровня изоляции:

- `BeginTransaction(TxMode txMode)`

  Параметр `Ydb.Sdk.Services.Query.TxMode` - это YDB специфичный уровень изоляции, ознакомиться поподробнее 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`.

Рассмотрим пример использования транзакции:

```
await using var connection = await dataSource.OpenConnectionAsync();
await using var transaction = await connection.BeginTransactionAsync();

await using var command1 = new YdbCommand(connection) { CommandText = "...", Transaction = transaction };
await command1.ExecuteNonQueryAsync();

await using var command2 = new YdbCommand(connection) { CommandText = "...", Transaction = transaction };
await command2.ExecuteNonQueryAsync();

await transaction.CommitAsync();
```

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.

Обработка ошибок

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
Параметр	Описание	Значение по умолчанию
Параметр	Описание	Значение по умолчанию
		`false`
Параметр	Описание	Значение по умолчанию
`KeepAlivePingDelay`	Клиент отправляет на сервер запрос о поддержании связи, если в течение этого периода времени не получает никаких сообщений по конкретному соединению. Это	`10` секунд
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.

YDB тип	.NET тип
`Yson`	`byte[]`

ись

YDB тип	DbType	.NET тип
`Bool`	`Boolean`	`bool`
`Text` (синоним `Utf8`)	`String`, `AnsiString`, `AnsiStringFixedLength`, `StringFixedLength`	`string`
`Bytes` (синоним `String`)	`Binary`	`byte[]`
`Uint8`	`Byte`	`byte`
`Uint16`	`UInt16`	`ushort`
`Uint32`	`UInt32`	`uint`
`Uint64`	`UInt64`	`ulong`
`Int8`	`SByte`	`sbyte`
`Int16`	`Int16`	`short`
`Int32`	`Int32`	`int`
`Int64`	`Int64`	`long`
`Float`	`Single`	`float`
`Double`	`Double`	`double`
`Date`	`Date`	`DateTime`
`Datetime`	`DateTime`	`DateTime`
`Timestamp`	`DateTime2` (для .NET типа `DateTime`), `DateTimeOffset` (для .NET типа `DateTimeOffset`)	`DateTime`, `DateTimeOffset`
`Decimal(22,9)`	`Decimal`, `Currency`	`decimal`

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

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=file:~/<myca>.cer`
   - Экземпляр облачной базы данных с токеном:
     `jdbc:ydb:grpcs://<host>:2135/<path/to/database>?token=file:~/my_token`
   - Экземпляр облачной базы данных с файлом сервисного аккаунта:
     `jdbc:ydb:grpcs://<host>:2135/<path/to/database>?saFile=file:~/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`. 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 `saFile`. 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:

- `saFile` — ключ сервисной учётной записи (Service Account Key). Значением свойства может быть содержимое ключа или путь к файлу с ключом.
- `iamEndpoint` — IAM-эндпойнт для аутентификации с помощью ключа сервисной учётной записи (Service Account Key).
- `token` — токен для аутентификации. Значением свойства может быть содержимое токена или путь к файлу с токеном.
- `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.
- `secureConnection` — использование TLS. Возможные значения:
  - `true` — принудительно использовать TLS;
  - `false` — не использовать TLS.

  Обычно безопасное соединение указывается с помощью протокола `grpcs://` в строке JDBC URL. Небезопасное соединение указывается с помощью протокола `grpc://`.
- `secureConnectionCertificate` — сертификат CA для TLS-соединения. Значением свойства может быть содержимое сертификата или путь к файлу с сертификатом.

> **Примечание**
>
> Ссылки на файлы в свойствах `saFile`, `token` или `secureConnectionCertificate` должны содержать префикс `file:`. Примеры:
>
> - `saFile=file:~/mysakey1.json`
> - `token=file:/opt/secret/token-file`
> - `secureConnectionCertificate=file:/etc/ssl/cacert.cer`

Использование режима QueryService

По умолчанию JDBC-драйвер для YDB использует старый API для обработки запросов, обеспечивая совместимость с предыдущими версиями YDB. Однако у этого API есть дополнительные ограничения. Чтобы отключить это поведение и использовать новый API QueryService, укажите свойство `useQueryService=true` в строке JDBC URL.

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=file:~/<myca>.cer`
- Экземпляр облачной базы данных с токеном:
  `jdbc:ydb:grpcs://<host>:2135/<path/to/database>?token=file:~/my_token`
- Экземпляр облачной базы данных с файлом сервисного аккаунта:
  `jdbc:ydb:grpcs://<host>:2135/<path/to/database>?saFile=file:~/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`.

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

Механизм аутентификации

В Kafka API аутентификация выполняется через механизмы `SASL_PLAINTEXT/PLAIN` или `SASL_SSL/PLAIN`.

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 базы данных.

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.

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 являются:

- отсутствие поддержки опции check.crcs;
- только одна стратегия назначения партиция - roundrobin;
- отсутствие возможности читать без предварительно созданной группы читателей.

Поэтому в конфигурации читателя всегда нужно указывать **имя группы читателей** и параметры:

- `check.crcs=false`
- `partition.assignment.strategy=org.apache.kafka.clients.consumer.RoundRobinAssignor`

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

Apache Spark при работе с Kafka не использует ничего из существующих ограничений Kafka API в YDB Topics.
Благодаря этому использование Spark с Ydb topics возможно в полном объеме.

```
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

> **Примечание**
>
> Сейчас поддержана не вся функциональность Flink при чтении и записи. Существуют следующие ограничения:
>
> ```
> - Работа exactly-once по Kafka API сейчас не поддержана, так как поддержка транзакций в Kafka API сейчас в разра
> - Подписка на топики через паттерн сейчас недоступна;
> - Использование CreateTime сообщения в качестве watermark сейчас недоступно, так как вместо CreateTime используе
> ```

```
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.

Частые проблемы и их решение

Ошибка Unexpected error in join group response

Полный текст ошибки:

```
Unexpected error in join group response: This most likely occurs because of a request being malformed by the client
library or the message was sent to an incompatible broker. See the broker logs for more details.
```

Скорее всего проблема в том, что не указано имя читателя или указанное имя читателя не существует в кластере YDB.

Решение: создайте читателя с помощью CLI или SDK

Запись

> **Примечание**
>
> Сейчас не поддержана запись по Kafka API с использованием Kafka транзакций. Транзакции доступны только при использовании
> YDB Topic API.
>
> В остальном запись в Apache Kafka и в YDB Topics через Kafka API ничем не отличается.

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

Apache Spark при работе с Kafka не использует ничего из существующих ограничений Kafka API в YDB Topics.
Благодаря этому использование Spark с Ydb topics возможно в полном объеме.

```
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

Примечание

Сейчас поддержана не вся функциональность Flink при чтении и записи. Существуют следующие ограничения:

```
- Работа exactly-once по Kafka API сейчас не поддержана, так как поддержка транзакций в Kafka API сейчас в разра
- Подписка на топики через паттерн сейчас недоступна;
- Использование CreateTime сообщения в качестве watermark сейчас недоступно, так как вместо CreateTime используе
```

```
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 9092 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 username=\"
<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 required
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());
  }
}
```

Ограничения Kafka API

Поддержка протокола Kafka версии 3.4.0 осуществляется в ограниченном объеме:

1. Поддержана только SASL/PLAIN-аутентификация.
2. Не поддержаны топики с настройкой `cleanup.policy=compact`.
   В связи с этим поверх Kafka API в YDB Topics не работают Kafka Connect, Schema Registry и Kafka Streams.
3. Не поддержано сжатие сообщений.
4. Не поддержаны транзакции.
5. Не поддержаны DDL операции. Для осуществления DDL-операций пользуйтесь YDB SDK или YDB CLI.
6. Не поддержана проверка crc.
7. Работа Kafka Connect поддерживается только в режиме одного процесса-исполнителя (standalone mode).
8. Если на топике включено автопартиционирование, то в такой топик нельзя писать или читать по протоколу Kafka API.

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®.

Важно

Экземпляры Kafka Connect для работы с YDB нужно разворачивать только в режиме одного процесса-исполнителя (standalone mode). YDB не поддерживает работу Kafka Connect в распределенном режиме (distributed mode).

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.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 --strip 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=PLAIN
   security.protocol=SASL_SSL
   sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="<sasl.username>"
   password="<sasl.password>";

   # Producer properties
   producer.sasl.mechanism=PLAIN
   producer.security.protocol=SASL_SSL
   producer.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="
   <sasl.username>" password="<sasl.password>";

   # Consumer properties
   consumer.sasl.mechanism=PLAIN
   consumer.security.protocol=SASL_SSL
   consumer.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="
   <sasl.username>" password="<sasl.password>";

   consumer.partition.assignment.strategy=org.apache.kafka.clients.consumer.RoundRobinAssignor
   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
   offset.storage.file.filename=/etc/kafka-connect-worker/worker.offset
   ```

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
```

Секция конфигурации `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 внутренних пользователей по аутентификационным

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`

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`.

Параметр	Описание
`hosts`	Список имен хостов, на котором работает LDAP-сервер

Параметр	Описание
use_access_service	Флаг разрешает аутентификацию пользователей в Yandex Cloud через IAM с использованием AccessService. Значение по умолчанию: `false`

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	Определяет время, прошедшее с последнего обновления, когда узел YDB попытается обновить токен пользователя. Конкретное время обновления будет лежать в интервале от `refresh_time/2` до `refresh_time`. 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 истечения срока действия токена пользователя, после которого токен удаляется из кэша узла YDB. Удаление происходит независимо от периода, указанного в параметре `life_time`. **Важно** Если сторонняя система успешно аутентифицировалась на узле YDB и с регулярностью чаще интервала `life_time` посылает запросы на тот же узел, YDB гарантированно определит возможное удаление или изменение привилегий аккаунта пользователя только по истечении срока `expire_time`. Чем короче указанный период, тем чаще узел YDB заново аутентифицирует пользователей и обновляет их привилегии. Однако, чрезмерно частое повторение аутентификации пользователей замедляет работу YDB, особенно если речь идёт о внешних пользователях. Установка значения этого параметра в секундах нивелирует работу кэша для токенов пользователей. Значение по умолчанию: `24h`
min_error_refresh_time	Минимальный период, после которого повторяется попытка обновить токен пользователя, при получении которого уже происходила ошибка (временный сбой). Совместно с параметром `max_error_refresh_time`, определяет границы для выбора задержки перед повторной попыткой обновить токен пользователя, при получении которого происходила ошибка. Каждая последующая задержка увеличивается, пока не достигнет значения `max_error_refresh_time`. Попытки обновить токен пользователя продолжаются до успешного обновления или до окончания периода `expire_time`.

Ключ	Описание
`request_client_certificate`	Требовать корректный клиентский сертификат для подключения узлов. Допустимые значения: • `false` — сертификат не требуется (применяется по умолчанию, если параметр не задан); • `true` — для подключения узлов требуется доверенный клиентский сертификат.
`default_group`	SID, присваиваемый всем подключениям с доверенным клиентским сертификатом, если нет явно заданных настроек в секции `client_certificate_definitions`.
`client_certificate_definitions`	Блок настроек требований к сертификатам узлов баз данных.
`member_groups`	Массив SID, присваиваемых подключениям, сертификаты которых соответствуют требованиям этого блока.
`require_same_issuer`	Требовать совпадение поля "Issuer" (наименование центра регистрации) для клиентского (узел базы данных) и серверного (узел хранения) 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.

`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.

Название параметра в конфигурации	Название параметра в Hive web-viewer	Формат	Описание	Значение п
`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_`

Название параметра в конфигурации	Название параметра в Hive web-viewer	Формат	Описание
`min_scatter_to_balance`	MinScatterToBalance	Вещественное число	Порог метрики Scatter для ресурсов CPU Network. Имеет приоритет ниже, чем пар
`min_cpuscatter_to_balance`	MinCPUScatterToBalance	Вещественное число	Порог метрики Scatter для ресурса CPU.
`min_memory_scatter_to_balance`	MinMemoryScatterToBalance	Вещественное число	Порог метрики Scatter для ресурса Memo
`min_network_scatter_to_balance`	MinNetworkScatterToBalance	Вещественное число	Порог метрики Scatter для ресурса Netwo

`min_counter_scatter_to_balance`	MinCounterScatterToBalance	Вещественное число	Порог метрики Scatter для фиктивного ре
`min_node_usage_to_balance`	MinNodeUsageToBalance	Вещественное	Потребление ресурсов на узле ниже дан

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	10000000

Название параметра в конфигурации	Название параметра в Hive web-viewer	Формат	Описание
`default_unit_iops`	DefaultUnitIOPS	Целое число	Значение по умолчанию для IOPS одного канала.

Название параметра в конфигурации	Название параметра в Hive web-	Формат	Описание	Значение по умолчанию

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

Название параметра в конфигурации	Название параметра в Hive web-viewer	Формат	Описание	Значение
`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	true

Параметр	Тип	Значение по умолчанию	Описание

Параметр	Тип	Описание
`component`	string	Имя компонента. С полным списком доступных компонентов можно ознакомиться на GitHub.

Параметр	Тип	Значение по умолчанию	Описание

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.

			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).

			30000
			600
			false
			1000

`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.

`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

- Справка по наблюдаемости YDB
- Справка по метрикам
- Трассировка в YDB
- Аудитный лог

Секция конфигурации `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 привилегий
  viewer_allowed_sids: <список SID'ов с правами просмотра состояния кластера>
  monitoring_allowed_sids: <список SID'ов с правами просмотра и изменения состояния кластера>
  administration_allowed_sids: <список SID'ов с доступом администратора 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 режима аутентификации

Параметр	Описание

Параметр	Описание

Параметр	Описание
`viewer_allowed_sids`	Список SID'ов с доступом уровня наблюдателя. Даёт возможность просмотра состояния системы, закрытого от публичного доступа (это большая часть страниц Embedded UI (YDB Monitoring)), без возможности делать изменения.

`monitoring_allowed_sids`	Список SID'ов с доступом уровня оператора. Даёт дополнительные возможности просмотра и выполнения действий, меняющих состояние системы. Например, выполнение бекапа, восстановления базы или выполнение YQL-запросов через Embedded UI.
`administration_allowed_sids`	Список SID'ов с доступом уровня администратора. Даёт право на выполнение административных действий с базами или кластером.
`register_dynamic_node_allowed_sids`	Список SID'ов , для которых разрешена регистрация узлов баз данных. По техническим причинам в этом списке также должен присутствовать субъект доступа `root@builtin`.

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`

`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
```

Пример включения TLS на HTTP-порту, что делает его использования HTTPS:

```
monitoring_config:
    monitoring_certificate_file: "/opt/ydb/certs/node.crt"
```

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 {#T}.

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 которых выступает в роли клиента для 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.

Справка по метрикам

Примечание

См. также Дашборды Grafana для YDB.

Метрики использования ресурсов

Имя метрики Тип, единицы измерения	Описание Метки
`table.session.active_count` `GAUGE`, штуки	Количество сессий, открытых клиентами в данный момент времени.
`topic.producers_count` `GAUGE`, штуки	Количество уникальных источников топика. 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.

Имя метрики Тип, единицы измерения	Описание Метки
`kqp.workload_manager.CpuQuotaManager.AverageLoadPercentage` `RATE`, штуки	Средняя загрузка базы данных, по этой метрики 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`.

`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 последнего закомиченного сообщения. 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.

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)

Name	Description
CPU by execution pool	Потребление CPU в различных пулах исполнения на всех нодах, микросекунды в секунду (один миллион соответствует потреблению одного ядра)

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)

Имя	Описание
Requests	Количество входящих запросов в секунду (по типу запроса)
Request bytes	Размер входящих запросов, байты в секунду (query, parameters, total)

Имя	Описание
Transactions	Метрики транзакций даташардов
Latencies	Гистограммы времен исполнения различных этапов транзакций даташардов

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

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.

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 Включение трассировки в Jaeger.

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 Embedded 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 Embedded 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.Net.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 Embedded 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.
`--mon-protocol`	Если протокол для подключения к кластеру не задан явно в эндпоинте, то используется указанное здесь значение.
`--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.

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)

оманду `make`.

ydbops_darwin_arm64	MacOS(arm64)

новлен исполняемый файл. По умолчанию: `~/ydb/bin`.

рованный 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`.

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 run \
    -d \ # запуск в фоне
    --rm \ # автоматическое удаление после установки
    --name ydb-local \ # имя контейнера
    -h localhost \ # хостейм
    -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 5432:5432 \ # открытие внешнего доступа к порту, обеспечивающему PostgreSQL-совместимость
    -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 # имя и тег образа
```

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.

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, Embedded UI YDB будет доступен по адресу http://localhost:8765.

Подробнее про остановку и удаление Docker-контейнера с YDB можно узнать в разделе Остановка Docker.

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
`POSTGRES_USER`	`string`	`postgres`	Имя пользователя для доступа через режим совместимости с PostgreSQL.
`POSTGRES_PASSWORD`	`string`		Пароль пользователя для доступа через режим совместимости с PostgreSQL.
`YDB_GRPC_ENABLE_TLS`	`0` или `1`	`1`	Включает использование TLS для gRPC соединений.
`YDB_GRPC_TLS_DATA_PATH`	`string`	`/ydb_data`	Путь до директории с TLS сертификатами для gRPC соединений.
`MON_PORT`	`int`	`8765`	HTTP-порт встроенного веб-интерфейса YDB.
`GRPC_PORT`	`int`	`2135`	gRPC порт.
`IC_PORT`	`int`	`19001`	Порт интерконнекта.
`GRPC_TLS_PORT`	`int`	`2137`	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_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 экспериментальных возможностей YDB, через запятую.
`YDB_ENABLE_COLUMN_TABLES`	`0` или `1`	`0`	Включает поддержку колоночных таблиц.

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.

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

Для аналитиков

В этом разделе собраны примеры и рекомендации по работе с аналитическими (OLAP) сценариями в YDB.

Данный раздел содержит следующие материалы

- Типичные OLAP сценарии
- Оптимизация аналитических запросов
- Импорт датасетов

Связанные разделы

- Вопросы и ответы об аналитике в YDB
- Колоночные таблицы
- Агрегатные функции

Рецепты кода на YDB SDK

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
- Координация
  - Распределённая блокировка
  - Обнаружение сервисов
  - Публикация конфигурации
  - Выбор лидера
- Диагностика проблем
  - Включить логирование
  - Подключить метрики в Prometheus
  - Подключить трассировку в Jaeger

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 подлкючения к YDB (создания драйвера) в разных YDB SDK.

Go (native)

```
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()

  // ...
}
```

Go (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 подчеркивавния:

```go
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

```java
public void work() {
    GrpcTransport transport = GrpcTransport.forConnectionString("grpc://localhost:2136/local")
            .build());
    // Работа с transport
    doWork(transport);
    transport.close();
}
```

JDBC Driver

```java
public void work() {
    // JDBC Driver должен быть доступен в classpath для автоматической загрузки
    Connection connection = DriverManager.getConnection("jdbc:ydb:grpc://localhost:2136/local");
    // Работа с connection
    doWork(connection);
    connection.close();
}
```

Python

```python
import ydb

with ydb.Driver(connection_string="grpc://localhost:2136?database=/local") as driver:
  driver.wait(timeout=5)
  ...
```

Python (asyncio)

```python
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# (.NET)

```csharp
using Ydb.Sdk;

var config = new DriverConfig(
    endpoint: "grpc://localhost:2136",
    database: "/local"
);

await using var driver = await Driver.CreateInitialized(config);
```

PHP

```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('AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA') // use from
reference/ydb-sdk/auth
];

$ydb = new Ydb($config);
```

Rust

```rust
let client = ClientBuilder::new_from_connection_string("grpc://localhost:2136?database=local")?
      .with_credentials(AccessTokenCredentials::from("..."))
      .client()?
```

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:

Go (native)

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: поторять 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.

Go (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 episode_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 episode_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

В YDB Java SDK механизм повторных запросов реализован в виде класс хелпера `SessionRetryContext`. Данный класс конструируется с помощью метода `SessionRetryContext.create` в который требуется передать реализацию интерфейса `SessionSupplier` - как правило это экземпляр класса `TableClient` или `QueryClient`.

Дополнительно пользователь может задавать некоторые другие опции:

- `maxRetries(int maxRetries)` - максимальное количество повторов операции, не включает в себя первое выполение. Значение по умолчанию `10`
- `retryNotFound(boolean retryNotFound)` - опция повтора операций, вернувших статус `NOT_FOUND`. По умолчанию включено.
- `idempotent(boolean idempotent)` - признак идемпотентности операций. Идемпотентные операции будут повторяться для более широкого списка ошибок. По умолчанию отключено.

Для запуска операций с ретраями класс `SessionRetryContext` предоставляет два метода:

- `CompletableFuture<Status> supplyStatus` - выполнение операции, возвращающей статус. В качестве аргумента принимает лямбду `Function<Session, CompletableFuture<Status>> fn`
- `CompletableFuture<Result<T>> supplyResult` - выполнение операции, возвращающей данные. В качестве аргумента принимает лямбду `Function<Session, CompletableFuture<Result<T>>> fn`

При использовании класса `SessionRetryContext` нужно учитывать, что повторное исполнение операции будет выполняться в следующих случаях:

- Лямбда вернула retryable код ошибки
- В рамках исполнения лямбды была вызвано `UnexpectedResultException` с retryable кодом ошибки

  Пример кода, использующего SessionRetryContext.supplyStatus:

```
private void createTable(TableClient tableClient, String database, String tableName) {
    SessionRetryContext retryCtx = SessionRetryContext.create(tableClient).build();
    TableDescription pets = TableDescription.newBuilder()
            .addNullableColumn("species", PrimitiveType.Text)
            .addNullableColumn("name", PrimitiveType.Text)
            .addNullableColumn("color", PrimitiveType.Text)
            .addNullableColumn("price", PrimitiveType.Float)
            .setPrimaryKeys("species", "name")
            .build();

    String tablePath = database + "/" + tableName;
    retryCtx.supplyStatus(session -> session.createTable(tablePath, pets))
```

```
            .join().expectSuccess();
}
```

Пример кода, использующего SessionRetryContext.supplyResult:

```
private void selectData(TableClient tableClient, String tableName) {
    SessionRetryContext retryCtx = SessionRetryContext.create(tableClient).build();
    String selectQuery
            = "DECLARE $species AS Text;"
            + "DECLARE $name AS Text;"
            + "SELECT * FROM " + tableName + " "
            + "WHERE species = $species AND name = $name;";

    Params params = Params.of(
            "$species", PrimitiveValue.newText("cat"),
            "$name", PrimitiveValue.newText("Tom")
    );

    DataQueryResult data = retryCtx
            .supplyResult(session -> session.executeDataQuery(selectQuery, TxControl.onlineRo(), params))
            .join().getValue();

    ResultSetReader rsReader = data.getResultSet(0);
    logger.info("Result of select query:");
    while (rsReader.next()) {
        logger.info("  species: {}, name: {}, color: {}, price: {}",
                rsReader.getColumn("species").getText(),
                rsReader.getColumn("name").getText(),
                rsReader.getColumn("color").getText(),
                rsReader.getColumn("price").getFloat()
        );
    }
}
```

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.

Go (native)

```
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)
  ...
}
```

Go (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

```
this.queryClient = QueryClient.newClient(transport)
        // 10 - minimum number of active sessions to keep in the pool during the cleanup
        // 500 - maximum number of sessions in the pool
        .sessionPoolMinSize(10)
        .sessionPoolMaxSize(500)
        .build();
```

JDBC Driver

При работе с JDBC, как правило, используются внешние пулы соединений, такие как HikariCP или C3p0. В режиме работы по умолчанию YDB JDBC драйвер определяет количество соединений, открытых внешним пулом, и самостоятельно подстраивает размер пула сессий. Поэтому для настройки пула сессий достаточно корректно настроить `HikariCP` или `C3p0`.

Пример настройки пула HikariCP в конфигурационном файле Spring:

```
  spring.datasource.url=jdbc:ydb:grpc://localhost:2136/local
  spring.datasource.driver-class-name=tech.ydb.jdbc.YdbDriver
  spring.datasource.hikari.maximum-pool-size=100 # maximum size of JDBC connections
```

Вставка данных

Ниже приведены примеры кода использования встроенных в YDB SDK средств выполнения вставки:

Go (native)

```
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)
  }
}
```

Go (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 and "+
            "Dave Krinsky. The series focuses on five young men who founded a startup company in Silicon Valley.",
          )),
          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)
  }
}
```

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.

Ниже приведены примеры кода использования встроенных в YDB SDK средств выполнения пакетной вставки:

Go (native)

```
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)
```

```
  }
}
```

Go (database/sql)

Реализация `database/sql` драйвера для YDB не поддерживает нетранзакционную пакетную вставку данных.
Для пакетной вставки следует пользоваться транзакционной вставкой.

Java

```
private static final String TABLE_NAME = "bulk_upsert";
private static final int BATCH_SIZE = 1000;

public static void main(String[] args) {
    String connectionString = args[0];

    try (GrpcTransport transport = GrpcTransport.forConnectionString(connectionString)
            .withAuthProvider(NopAuthProvider.INSTANCE) // анонимная аутентификация
            .build()) {

        // Для bulk upsert необходимо использовать полный путь к таблице
        String tablePath = transport.getDatabase() + "/" + TABLE_NAME;
        try (TableClient tableClient = TableClient.newClient(transport).build()) {
            SessionRetryContext retryCtx = SessionRetryContext.create(tableClient).build();
            execute(retryCtx, tablePath);
        }
    }
}

public static void execute(SessionRetryContext retryCtx, String tablePath) {
    // описание таблицы
    StructType structType = StructType.of(
        "app", PrimitiveType.Text,
        "timestamp", PrimitiveType.Timestamp,
        "host", PrimitiveType.Text,
        "http_code", PrimitiveType.Uint32,
        "message", PrimitiveType.Text
    );

    // генерация пакета записей
    List<Value<?>> list = new ArrayList<>(50);
    for (int i = 0; i < BATCH_SIZE; i += 1) {
        // добавление новой строки в виде значения-структуры
        list.add(structType.newValue(
            "app", PrimitiveValue.newText("App_" + String.valueOf(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")
        ));
    }

    // Create list of structs
    ListValue rows = ListType.of(structType).newValue(list);
    // Do retry operation on errors with best effort
    retryCtx.supplyStatus(
        session -> session.executeBulkUpsert(tablePath, rows, new BulkUpsertSettings())
    ).join().expectSuccess("bulk upsert problem");
}
```

JDBC

```
private static final int BATCH_SIZE = 1000;

public static void main(String[] args) {
    String connectionUrl = args[0];

    try (Connection conn = DriverManager.getConnection(connectionUrl)) {
        try (PreparedStatement ps = conn.prepareStatement(
                "BULK UPSERT INTO bulk_upsert (app, timestamp, host, http_code, message) VALUES (?, ?, ?, ?, ?);"
        )) {
            for (int i = 0; i < BATCH_SIZE; i += 1) {
                ps.setString(1, "App_" + String.valueOf(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();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

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*.

Serializable

Go (native)

```
package main

import (
  "context"
  "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)
  txControl := table.TxControl(
    table.BeginTx(table.WithSerializableReadWrite()),
    table.CommitTx(),
  )
  err := driver.Table().Do(scope.Ctx, func(ctx context.Context, s table.Session) error {
    _, _, err := s.Execute(ctx, txControl, "SELECT 1", nil)
    return err
  })
  if err != nil {
    fmt.Printf("unexpected error: %v", err)
  }
}
```

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 only)
    ],

    'credentials' => new AccessTokenAuthentication('AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA') // use from reference/ydb-
sdk/auth
];

$ydb = new Ydb($config);
$ydb->table()->retryTransaction(function(Session $session){
  $session->query('SELECT 1;');
})
```

Online Read-Only

Go (native)

```go
package main

import (
  "context"
  "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)
  txControl := table.TxControl(
    table.BeginTx(table.WithOnlineReadOnly(table.WithInconsistentReads())),
    table.CommitTx(),
  )
  err := driver.Table().Do(scope.Ctx, func(ctx context.Context, s table.Session) error {
    _, _, err := s.Execute(ctx, txControl, "SELECT 1", nil)
    return err
  })
  if err != nil {
    fmt.Printf("unexpected error: %v", err)
  }
}
```

Stale Read-Only

Go (native)

```go
package main

import (
  "context"
  "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)
  txControl := table.TxControl(
    table.BeginTx(table.WithStaleReadOnly()),
    table.CommitTx(),
  )
  err := driver.Table().Do(scope.Ctx, func(ctx context.Context, s table.Session) error {
    _, _, err := s.Execute(ctx, txControl, "SELECT 1", nil)
    return err
  })
  if err != nil {
    fmt.Printf("unexpected error: %v", err)
  }
}
```

Snapshot Read-Only

Go (native)

```go
package main

import (
  "context"
  "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)
  txControl := table.TxControl(
    table.BeginTx(table.WithSnapshotReadOnly()),
    table.CommitTx(),
  )
  err := driver.Table().Do(scope.Ctx, func(ctx context.Context, s table.Session) error {
    _, _, err := s.Execute(ctx, txControl, "SELECT 1", nil)
    return err
  })
  if err != nil {
    fmt.Printf("unexpected error: %v", err)
  }
}
```

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))
```

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))
```

Включение вытеснения во внешнее S3-совместимое хранилище

> **Внимание**
>
> 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_data")),
                    TTtlTierSettings(TDuration::Hours(24), TTtlDeleteAction("/Root/s3_cold_data"))
                })
        .EndAlterTtlSettings()
);
```

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'))
)
```

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)
```

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
```

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

Ниже приведены примеры кода аутентификации при помощи токена в разных YDB SDK.

Go (native)

```
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)
  ...
}
```

Go (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://localohost:2135/local?token="+os.Getenv("YDB_TOKEN"))
  if err != nil {
    panic(err)
  }
  defer db.Close()
  ...
}
```

Java

```
public void work(String accessToken) {
    AuthProvider authProvider = new TokenAuthProvider(accessToken);

    GrpcTransport transport = GrpcTransport.forConnectionString("grpcs://localohost:2135/local")
            .withAuthProvider(authProvider)
            .build());

    QueryClient queryClient = QueryClient.newClient(transport).build();

    doWork(queryClient);

    queryClient.close();
    transport.close();
}
```

JDBC

```
public void work() {
    // Подключение с указанием значения токена аутентификации
    Properties props1 = new Properties();
    props1.setProperty("token", "AQAD-XXXXXXXXXXXXXXXXXXX");
    try (Connection connection = DriverManager.getConnection("jdbc:ydb:grpc://localhost:2136/local", props1)) {
      doWork(connection);
    }

    // Подключение с чтением токена аутентификации из указанного файла
    Properties props2 = new Properties();
    props2.setProperty("tokenFile", "~/.ydb_token");
    try (Connection connection = DriverManager.getConnection("jdbc:ydb:grpc://localhost:2136/local", props2)) {
      doWork(connection);
    }
}
```

Node.js

```
import { Driver, TokenAuthService } from 'ydb-sdk';

export async function connect(endpoint: string, database: string, accessToken: string) {
    const authService = new TokenAuthService(accessToken);
    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

```
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)
    ...
```

Python (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())
```

C# (.NET)

```
using Ydb.Sdk;
using Ydb.Sdk.Auth;

const string endpoint = "grpc://localhost:2136";
const string database = "/local";
const string token = "MY_VERY_SECURE_TOKEN";

var config = new DriverConfig(
    endpoint: endpoint,
    database: database,
    credentials: new TokenProvider(token)
);

await using var driver = await Driver.CreateInitialized(config);
```

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

Ниже приведены примеры кода анонимной аутентификации в разных YDB SDK.

Go (native)

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)
  ...
}
```

Go (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

```
public void work(String connectionString) {
    AuthProvider authProvider = NopAuthProvider.INSTANCE;

    GrpcTransport transport = GrpcTransport.forConnectionString(connectionString)
            .withAuthProvider(authProvider)
            .build());
```

```
    QueryClient queryClient = QueryClient.newClient(transport).build();

    doWork(queryClient);

    queryClient.close();
    transport.close();
}
```

JDBC

```
public void work() {
    // Подключение без дополнительных опций будет осуществляться с анонимной аутентификацией
    try (Connection connection = DriverManager.getConnection("jdbc:ydb:grpc://localhost:2136/local")) {
      doWork(connection);
    }
}
```

Node.js

```
import { Driver, AnonymousAuthService } from 'ydb-sdk';

export async function connect(endpoint: string, database: string) {
    const authService = new AnonymousAuthService();
    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

```
import os
import ydb

with ydb.Driver(
    connection_string=os.environ["YDB_CONNECTION_STRING"],
    credentials=ydb.credentials.AnonymousCredentials(),
) as driver:
    driver.wait(timeout=5)
    ...
```

Python (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())
```

C# (.NET)

```
using Ydb.Sdk;
using Ydb.Sdk.Auth;

const string endpoint = "grpc://localhost:2136";
const string database = "/local";

var config = new DriverConfig(
    endpoint: endpoint,
    database: database,
```

```
    credentials: new AnonymousProvider()
);

await using var driver = await Driver.CreateInitialized(config);
```

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

Ниже приведены примеры кода аутентификации при помощи файла сервисного аккаунта в разных YDB SDK.

Go (native)

```
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)
  ...
}
```

Go (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

```
public void work(String connectionString, String saKeyPath) {
    AuthProvider authProvider = CloudAuthHelper.getServiceAccountFileAuthProvider(saKeyPath);
```

```
    GrpcTransport transport = GrpcTransport.forConnectionString(connectionString)
            .withAuthProvider(authProvider)
            .build());

    QueryClient queryClient = QueryClient.newClient(transport).build();

    doWork(queryClient);

    queryClient.close();
    transport.close();
}
```

JDBC

```
public void work() {
    Properties props = new Properties();
    props.setProperty("saKeyFile", "~/keys/sa_key.json");
    try (Connection connection = DriverManager.getConnection("jdbc:ydb:grpc://localhost:2136/local", props)) {
      doWork(connection);
    }

    // Опцию saKeyFile также можно указать прямо в JDBC URL
    try (Connection connection = DriverManager.getConnection("jdbc:ydb:grpc://localhost:2136/local?
saKeyFile=~/keys/sa_key.json")) {
      doWork(connection);
    }
}
```

Node.js

Загрузка данных сервисного аккаунта из файла:

```
import { Driver, getSACredentialsFromJson, IamAuthService } from 'ydb-sdk';

export async function connect(endpoint: string, database: string, serviceAccountFilename: string) {
    const saCredentials = getSACredentialsFromJson(serviceAccountFilename);
    const authService = new IamAuthService(saCredentials);
    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
}
```

Загрузка данных сервисного аккаунта из стороннего источника (например, из хранилища секретов):

```
import { Driver, IamAuthService } from 'ydb-sdk';
import { IIamCredentials } from 'ydb-sdk/build/cjs/src/credentials';

export async function connect(endpoint: string, database: string) {
    const saCredentials: IIamCredentials = {
        serviceAccountId: 'serviceAccountId',
        accessKeyId: 'accessKeyId',
        privateKey: Buffer.from('-----BEGIN PRIVATE KEY-----\nyJ1yFwJq...'),
        iamEndpoint: 'iam.api.cloud.yandex.net:443',
    };
    const authService = new IamAuthService(saCredentials);
    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

```
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)
    ...
```

Python (asyncio)

```
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())
```

C# (.NET)

```
using Ydb.Sdk;
using Ydb.Sdk.Yc;

const string endpoint = "grpc://localhost:2136";
const string database = "/local";

var saProvider = new ServiceAccountProvider(
    saFilePath: "path/to/sa_file.json" // Path to file with service account JSON info);
);
await saProvider.Initialize();

var config = new DriverConfig(
    endpoint: endpoint,
    database: database,
    credentials: saProvider
);

await using var driver = await Driver.CreateInitialized(config);
```

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
<?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

Ниже приведены примеры кода аутентификации при помощи переменных окружения в разных YDB SDK.

Go (native)

```
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)
  ...
}
```

Go (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

```
public void work(String connectionString) {
    AuthProvider authProvider = CloudAuthHelper.getMetadataAuthProvider();

    GrpcTransport transport = GrpcTransport.forConnectionString(connectionString)
            .withAuthProvider(authProvider)
            .build());
```

```
    QueryClient queryClient = QueryClient.newClient(transport).build();

    doWork(queryClient);

    queryClient.close();
    transport.close();
}
```

JDBC

```
public void work() {
    Properties props = new Properties();
    props.setProperty("useMetadata", "true");
    try (Connection connection = DriverManager.getConnection("jdbc:ydb:grpc://localhost:2136/local", props)) {
      doWork(connection);
    }

    // Опцию useMetadata также можно указать прямо в JDBC URL
    try (Connection connection = DriverManager.getConnection("jdbc:ydb:grpc://localhost:2136/local?useMetadata=true")) {
      doWork(connection);
    }
}
```

Node.js

```
import { Driver, MetadataAuthService } from 'ydb-sdk';

export async function connect(endpoint: string, database: string) {
    const authService = new MetadataAuthService();
    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

```
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)
    ...
```

Python (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())
```

C# (.NET)

```
using Ydb.Sdk;
using Ydb.Sdk.Yc;

var metadataProvider = new MetadataProvider();

// Await initial IAM token.
await metadataProvider.Initialize();

var config = new DriverConfig(
    endpoint: endpoint, // Database endpoint, "grpcs://host:port"
    database: database, // Full database path
    credentials: metadataProvider
);

await using var driver = await Driver.CreateInitialized(config);
```

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

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.

Go (native)

```
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)
  ...
}
```

Go (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

```
public void work(String connectionString) {
    AuthProvider authProvider = new EnvironAuthProvider();

    GrpcTransport transport = GrpcTransport.forConnectionString(connectionString)
            .withAuthProvider(authProvider)
            .build());

    QueryClient queryClient = QueryClient.newClient(transport).build();

    doWork(queryClient);

    queryClient.close();
    transport.close();
}
```

Node.js

```
  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

```
  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)
      ...
```

Python (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())
```

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

Ниже приведены примеры кода аутентификации при помощи логина и пароля в разных YDB SDK.

C++

```
auto driverConfig = NYdb::TDriverConfig()
  .SetEndpoint(endpoint)
  .SetDatabase(database)
  .SetCredentialsProviderFactory(NYdb::CreateLoginCredentialsProviderFactory({
      .User = "user",
      .Password = "password",
  }));

NYdb::TDriver driver(driverConfig);
```

Go (native)

Передать логин и пароль можно в составе строки подключения. Например, так:

```
"grpcs://login:password@localohost: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)
  ...
}
```

Go (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`:

```go
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

```java
public void work(String connectionString, String username, String password) {
    AuthProvider authProvider = new StaticCredentials(username, password);

    GrpcTransport transport = GrpcTransport.forConnectionString(connectionString)
            .withAuthProvider(authProvider)
            .build());

    QueryClient queryClient = QueryClient.newClient(transport).build();

    doWork(queryClient);

    queryClient.close();
    transport.close();
}
```

JDBC

```java
public void work(String username, String password) {
    Properties props = new Properties();
    props.setProperty("username", username);
    props.setProperty("password", password);
    try (Connection connection = DriverManager.getConnection("jdbc:ydb:grpc://localhost:2136/local", props)) {
      doWork(connection);
    }

    // Логин и пароль могут быть указаны напрямую
    try (Connection connection = DriverManager.getConnection("jdbc:ydb:grpc://localhost:2136/local", username,
password)) {
      doWork(connection);
    }
}
```

Node.js

```javascript
import { Driver, StaticCredentialsAuthService } from 'ydb-sdk';

export async function connect(endpoint: string, database: string, user: string, password: string) {
    const authService = new StaticCredentialsAuthService(user, password, endpoint, {
        tokenExpirationTimeout: 20000,
    })
    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

```
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)
    ...
```

Python (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())
```

C# (.NET)

```
using Ydb.Sdk;
using Ydb.Sdk.Auth;

const string endpoint = "grpc://localhost:2136";
const string database = "/local";

var config = new DriverConfig(
    endpoint: endpoint, // Database endpoint, "grpcs://host:port"
    database: database, // Full database path
    credentials: new StaticCredentialsProvider(user, password)
);

await using var driver = await Driver.CreateInitialized(config);
```

PHP

```
<?php

use YdbPlatform\Ydb\Ydb;
use YdbPlatform\Ydb\Auth\Implement\StaticAuthentication;
```

```php
$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

Равномерный случайный выбор

YDB SDK использует алгоритм `random_choice` (равномерную случайную балансировку) по умолчанию.

Ниже приведены примеры кода принудительной установки алгоритма балансировки "равномерный случайный выбор" в разных YDB SDK.

Go (native)

```
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)
  ...
}
```

Go (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()
```

```
  ...
}
```

```
  ...
```

Предпочитать ближайший дата-центр

Ниже приведены примеры кода установки опции алгоритма балансировки "предпочитать ближайший дата-центр" в разных YDB SDK.

Go (native)

```
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)
  ...
}
```

Go (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()
  ...
}
```

Предпочитать конкретную зону доступности

Ниже приведены примеры кода установки опции алгоритма балансировки "предпочитать зону доступности" в разных YDB SDK.

Go (native)

```
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)
  ...
}
```

Go (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()
  ...
}
```

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

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
}
```

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.

Диагностика проблем

При диагностике проблем, связанных с YDB, помогают средства диагностики, такие как логирование, метрики, трассы OpenTracing/Jaeger. Мы настоятельно рекомендуем подключать средства диагностики заранее, пока не случилось проблем. Как минимум при расследовании можно увидеть изменения картины до проблем, во время и после. Это сильно ускоряет расследование происшествия и нашу помощь.

В данном разделе содержатся рецепты кода с подключением средств диагностики в разных YDB SDK.

Содержание:

- Включить логирование
- Подключить метрики в Prometheus
- Подключить трассировку в Jaeger

Включение логирования

Ниже приведены примеры кода включения логирования в разных YDB SDK.

Go (native)

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.

Go (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

В YDB Java SDK для логирования используется библиотека slf4j, которая позволяет использовать различные уровни логирования (`error`, `warn`, `info`, `debug`, `trace`) для одного или нескольких логгеров. В текущей реализации доступны следующие логгеры:

- Логгер `tech.ydb.core.grpc` предоставляет информацию о внутренней реализации grpc протокола
- уровень `debug` логирует все операции по протоколу grpc, рекомедуется использовать только для отладки
- уровень `info` рекомендуется использовать по умолчанию
- Логгер `tech.ydb.table.impl` на уровне `debug` позволяет отслеживать внутреннее состояние драйвера ydb, в частности работу пула сессий.
- Логгер `tech.ydb.table.SessionRetryContext` на уровне `debug` будет информировать о количестве ретраев, результатах выполненных запросов, времени выполнения отдельных ретраев и общем времени выполнения всей операции
- Логгер `tech.ydb.table.Session` на уровне `debug` предоставляет информацию о тексте запроса, статусе ответа и времени выполнения для различных операций сессии

Включение и настройка логгеров Java SDK зависит от используемой реализации `slf4j-api`.
Здесь приведен пример конфигурации `log4j2` для библиотеки `log4j-slf4j-impl`

```
<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>
```

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);
```

Включение метрик в Prometheus

Ниже приведены примеры кода включения метрик в Prometheus в разных YDB SDK.

Go (native)

```
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)
    ...
}
```

Go (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.OpnDB(connector)
    defer db.Close()
    ...
}
```

Включение трассировки в Jaeger

Ниже приведены примеры кода включения трассировки в Jaeger в разных YDB SDK.

Go (native)

```
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)
    ...
}
```

Go (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.OpnDB(connector)
    defer db.Close()
    ...
}
```

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 встроен инструментарий для проведения нагрузочного тестирования несколькими стандартными бенчмарками:

Бенчмаркам	Справка
TPC-H	tpch
TPC-DS	tpcds
ClickBench	clickbench

ьное описание для каждого см. в соответствующих разделах, ссылки выше.
арками собраны в соответствующие группы, при этом для всех команд единым образом задается путь в

```
th path/in/database ...
n/in/database ...
th/in/database ...
```

Нагрузочное тестирование можно разбить на 3 этапа:

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 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
```

На данном этапе можно 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. Параметра `--clear`

Подробнее см. описание команд для каждого бенчмарка:

- clickbench init
- tpch init
- tpcds init

Наполнение данными

Наполнение данными выполняется при помощи команды `import`. Данная команда специфична для каждого бенчмарка и её поведение зависит от подкоманд. При этом есть и общие для всех параметры.

Подробное описание см. в соответствующих разделах:

- clickbench import
- tpch import
- tpcds import

Примеры:

```
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
```

Тестирование

Непосредственно тестирование выполняется командой `run`. Её поведение практически одинаково для разных бенчмарков, хотя некоторые различия всё-таки присутствуют.

Примеры:

```
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
```

Команда позволяет выбрать запросы для исполнения, сгенерировать несколько видов отчетов, собрать статистику исполнения и тд.

Подробное описание см. в соответствующих разделах:

- clickbench run
- tpch run
- tpcds run

Очистка

После выполнения всего необходимого тестирования данные могут быть удалены из БД.
Сделано это может быть при помощи команды `clean`:

```
ydb workload clickbench --path clickbench/hits clean
ydb workload tpch --path tpch/s1 clean
ydb workload tpcds --path tpcds/s1 clean
```

Подробное описание см. в соответствующих разделах:

- clickbench clean
- tpch clean
- tpcds 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 mytable
```

Включение вытеснения данных во внешнее S3-совместимое хранилище

Внимание

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

Внимание

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 OBJECT access_key (TYPE SECRET) WITH (value="...");
CREATE OBJECT secret_key (TYPE SECRET) 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_NAME="access_key",
    AWS_SECRET_ACCESS_KEY_SECRET_NAME="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("PT0S") ON expire_at
);
```

Выключение TTL

```
ALTER TABLE `mytable` RESET (TTL);
```

Типичные OLAP сценарии

1. **Пакетная аналитика**
   - Агрегация больших объемов исторических данных
   - Построение отчетов за длительные периоды
   - Вычисление статистик по всему набору данных
2. **Интерактивная аналитика**
   - Ad-hoc запросы для исследования данных
   - Построение динамических дашбордов
   - Drill-down анализ с разными уровнями детализации
3. **Предварительная обработка данных**
   - Очистка и нормализация данных
   - Обогащение данных из разных источников
   - Подготовка данных для машинного обучения

Связанные разделы

- Вопросы и ответы об аналитике в YDB
- Колоночные таблицы
- Агрегатные функции

Оптимизация аналитических запросов

1. **Правильно выбирать первичный ключ**
   - Использовать колонки, по которым часто выполняется фильтрация
   - Учитывать порядок колонок для оптимизации range-запросов
2. **Оптимизировать схему данных**
   - Использовать колоночное хранение для аналитических таблиц
   - Правильно выбирать типы данных для экономии места
   - Применять партиционирование для больших (более 2Гб) аналитических таблиц
3. **Оптимизировать запросы**
   - Запрашивать только необходимые колонки
   - Использовать подходящие индексы
   - Правильно формулировать условия фильтрации

Связанные разделы

- Вопросы и ответы об аналитике в YDB
- Колоночные таблицы
- Агрегатные функции

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:

Embedded UI

Подробнее про Embedded 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:

Embedded 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:

Embedded UI

Подробнее про Embedded 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:

Embedded 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.csv
```

3. Создайте таблицу в YDB одним из следующих способов:

Embedded UI

Подробнее про Embedded 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:

Embedded 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.csv
```

3. Создайте таблицу в YDB одним из следующих способов:

Embedded UI

Подробнее про Embedded 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:

Embedded 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:

Embedded UI

Подробнее про Embedded 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:

Embedded 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:

Embedded UI

Подробнее про Embedded 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:

Embedded 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          |
└──────────┴─────────────┘
```

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;
- Встроенный UI;
- Планы запросов;
- 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.

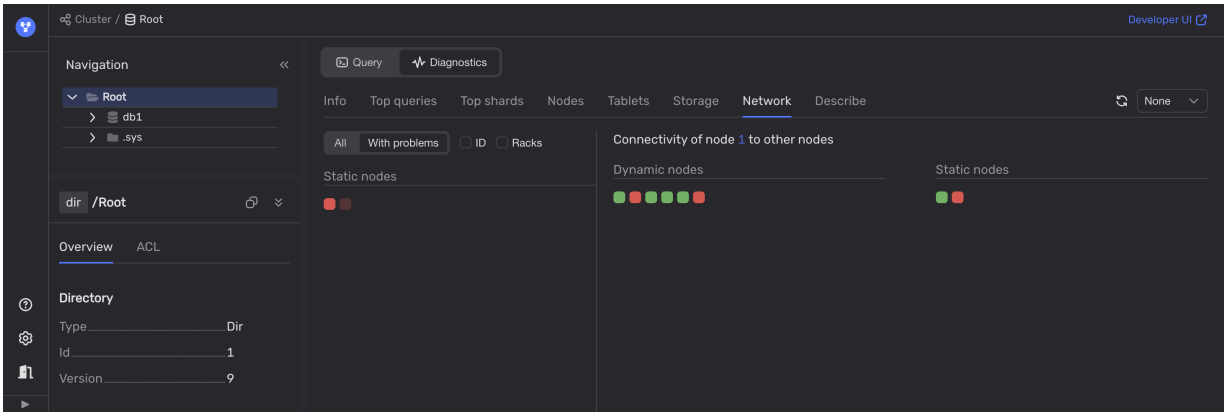
Сетевые проблемы

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 Встроенном UI:

1. Откройте Встроенный 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.

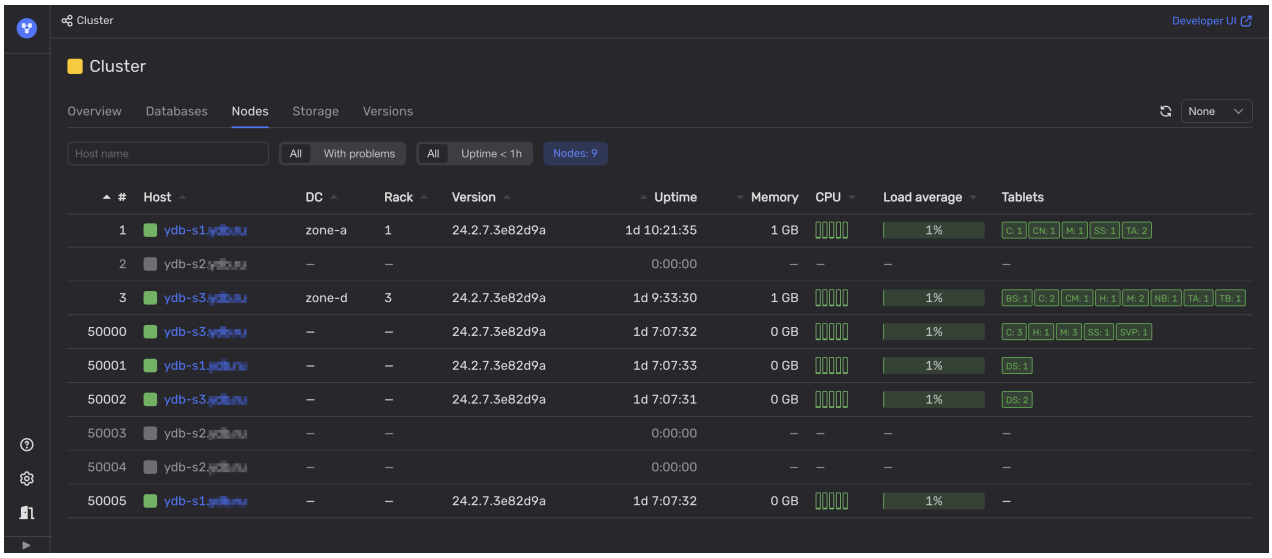
Сбои в работе датацентров

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 Встроенный 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 Встроенном 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.

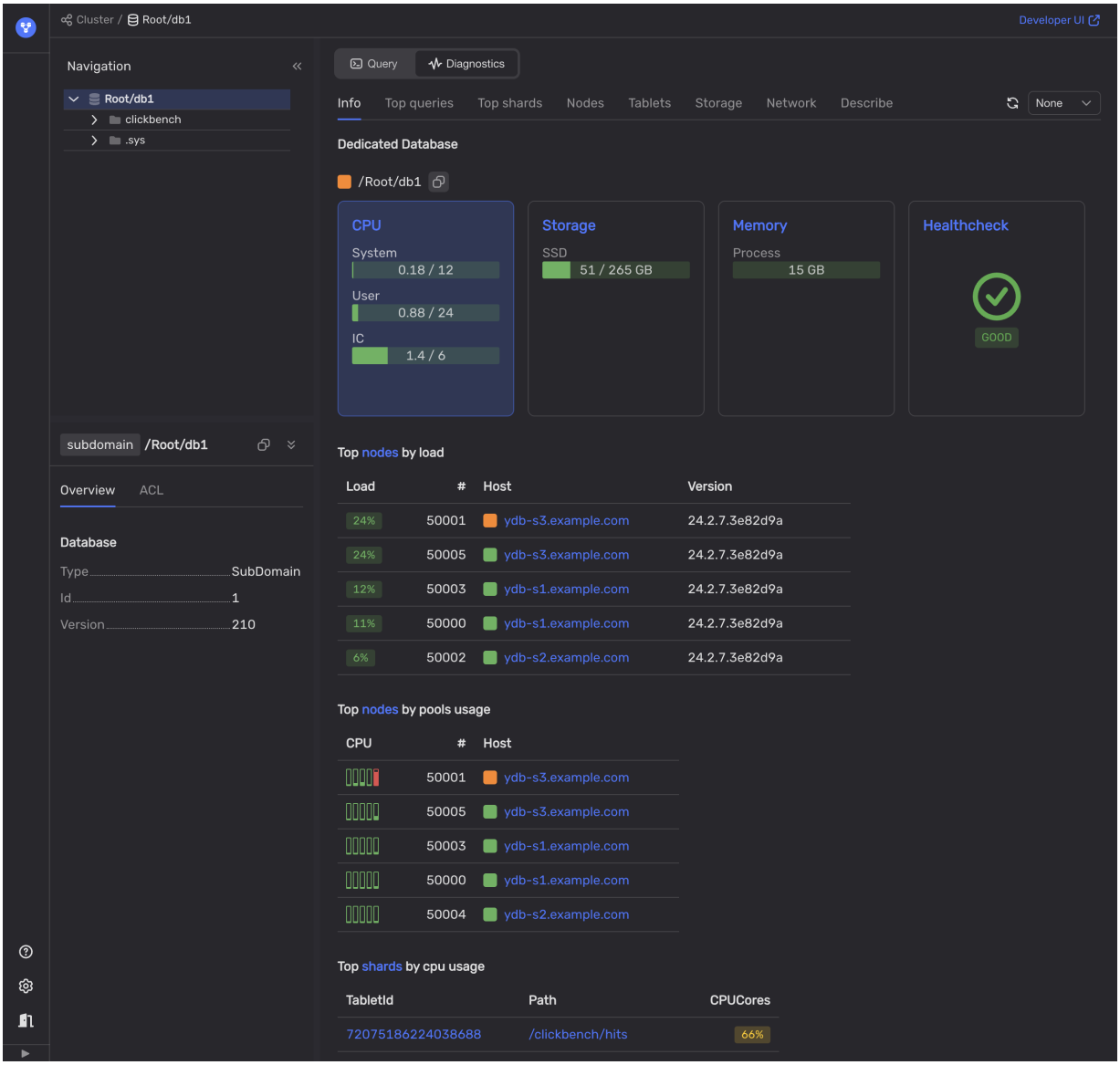
Недостаточное быстродействие процессора

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.

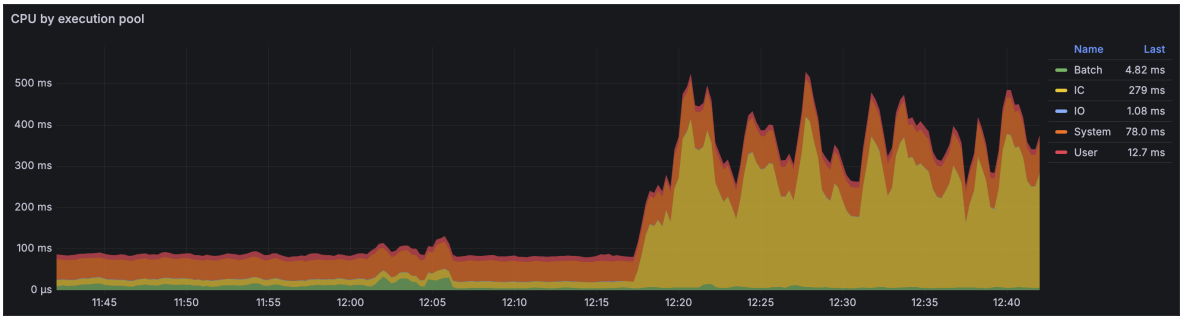
Узлы YDB в основном используют ресурсы процессора на выполнение акторов. На каждом узле акторы выполняются с использованием ресурсов одного из пулов акторной системы. Потребление ресурсов каждого пула измеряется отдельно, что позволяет точнее отслеживать изменения в потреблении ресурсов.

Диагностика

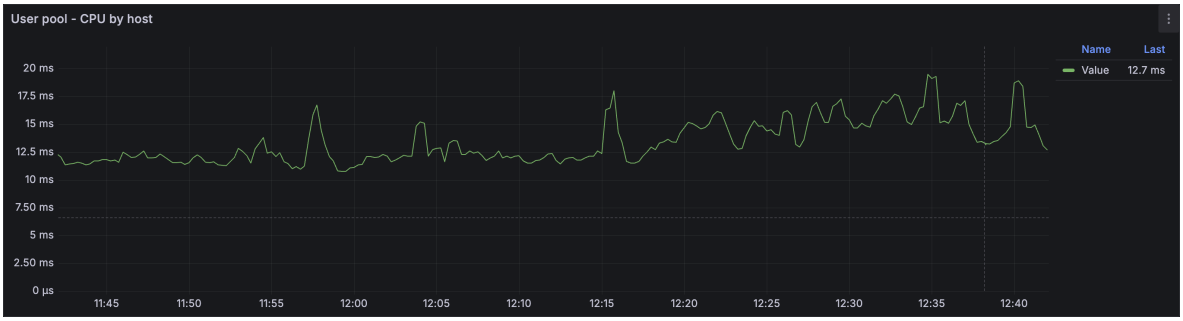
1. Используйте вкладку **Diagnostics** во встроенном UI для анализа загрузки процессора во всех пулах ресурсов:
   1.1. Откройте встроенный UI, перейдите на вкладку **Databases** и нажмите на требуемую базу данных.
   1.2. На вкладке **Navigation** убедитесь, что требуемая база данных выбрана.
   1.3. 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**

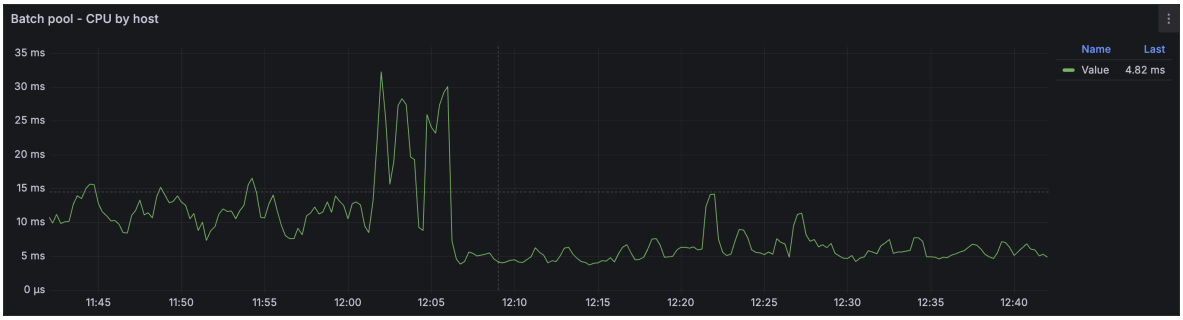


- User pool - CPU by host



- System pool - CPU by host

- Batch pool - CPU by host

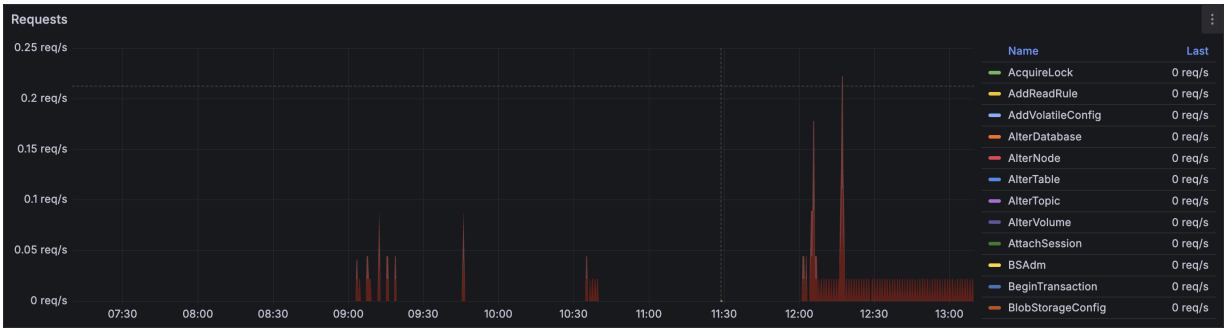


- IC pool - CPU by host

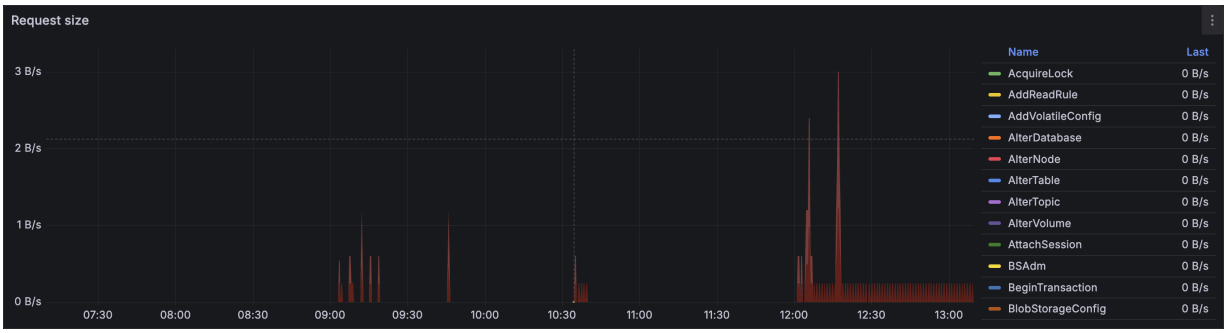
- IO pool - CPU by host

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 встроенный 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, file-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:

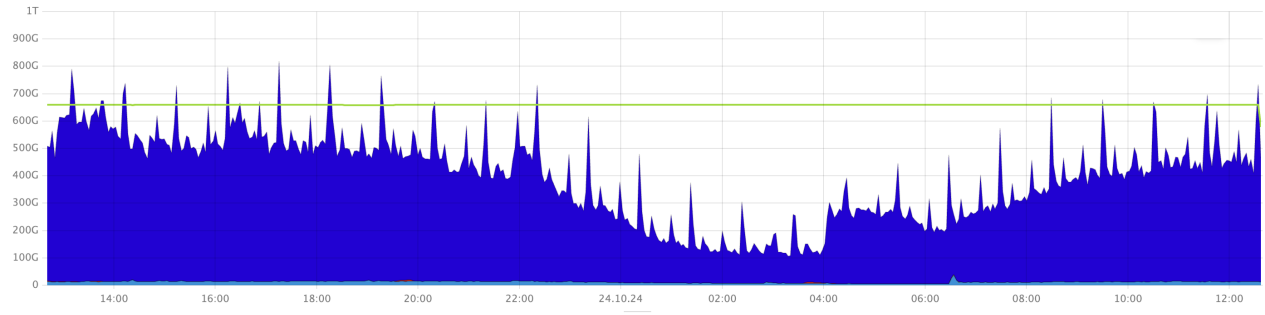
- Если нагрузка на YDB увеличилась из-за новых методов использования или возросшей частоты запросов, попробуйте оптимизировать приложение, чтобы снизить нагрузку на YDB, или добавьте больше узлов YDB.
- Если нагрузка на YDB не изменилась, но узлы всё равно перезапускаются, рассмотрите возможность добавления большего количества узлов YDB или увеличения жёсткого лимита памяти для узлов. Дополнительную информацию об управлении памятью в YDB см. в статье Контроллер памяти.

Недостаточная пропускная способность ввода-вывода

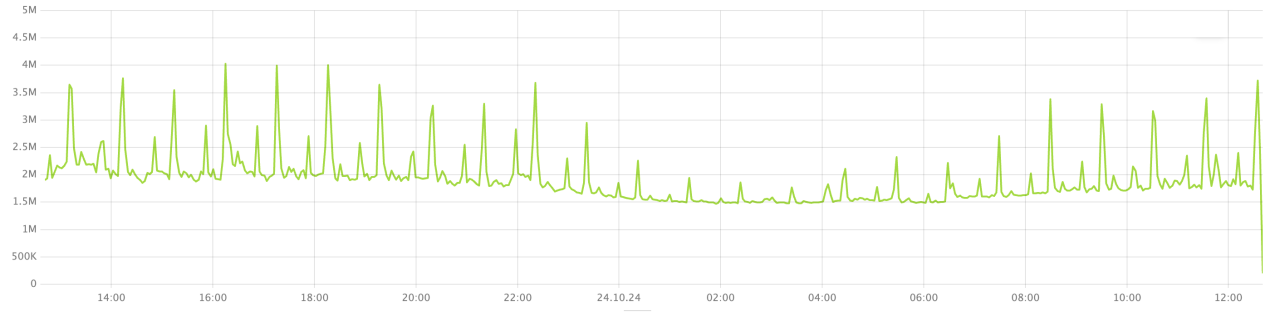
Высокая скорость операций чтения/записи может перегрузить дисковую систему и приводить к увеличению задержек доступа к данным. Когда распределённое хранилище не может читать или записывать данные с достаточной скоростью, запросы к базе данных, требующие доступа к диску, могут замедляться.

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.

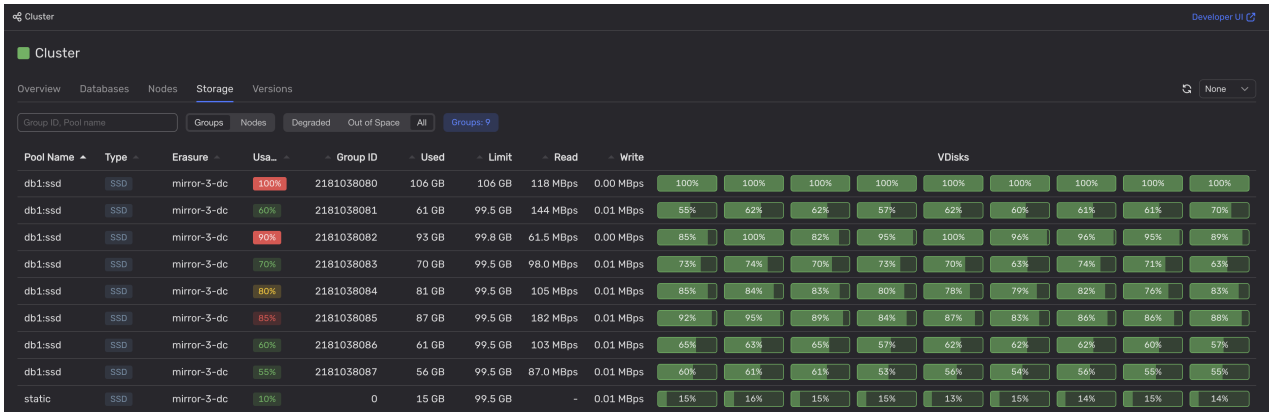
Недостаточное дисковое пространство

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 встроенном 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.

Если системное время узлов, на которых запущены таблетки-координаторы, 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 координатор с самыми отстающими часами достигнет того же времени.

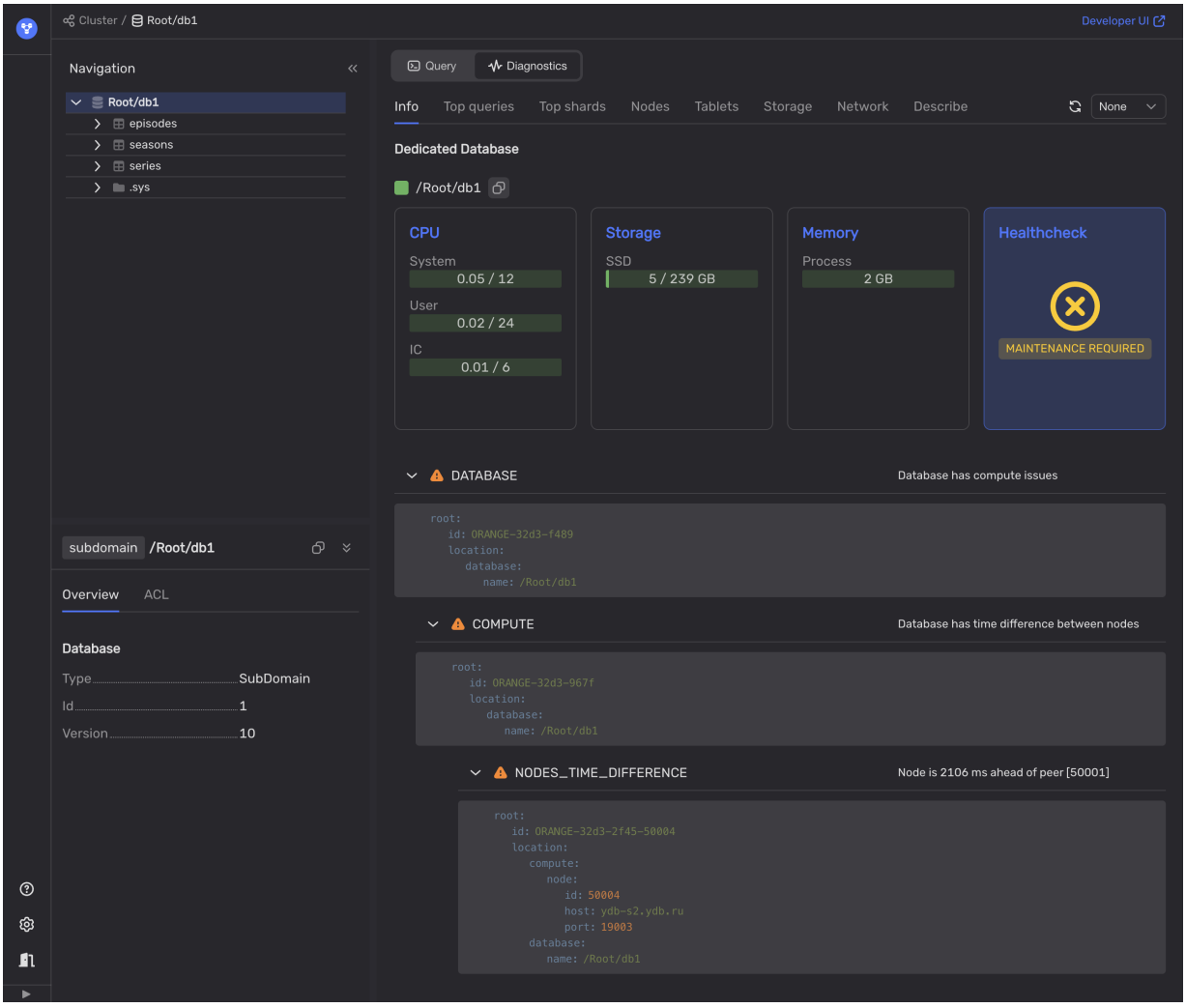
Более того, если отклонение во времени превысит 30 секунд, система YDB откажется обрабатывать распределённые транзакции. Перед тем как координаторы приступят к планированию транзакции, задействованные data shards определяют допустимый диапазон временных меток (timestamps) для транзакции. Начало этого 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 Встроенном UI:
   1.1. Во Встроенном 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 Встроенном UI.
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, `timedatectl show-timesync --all`.

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 `timesyncd`, `ntpd`, `chrony` или аналогичного инструмента. Рекомендуется использовать один и тот же источник времени для всех серверов в кластере YDB.

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

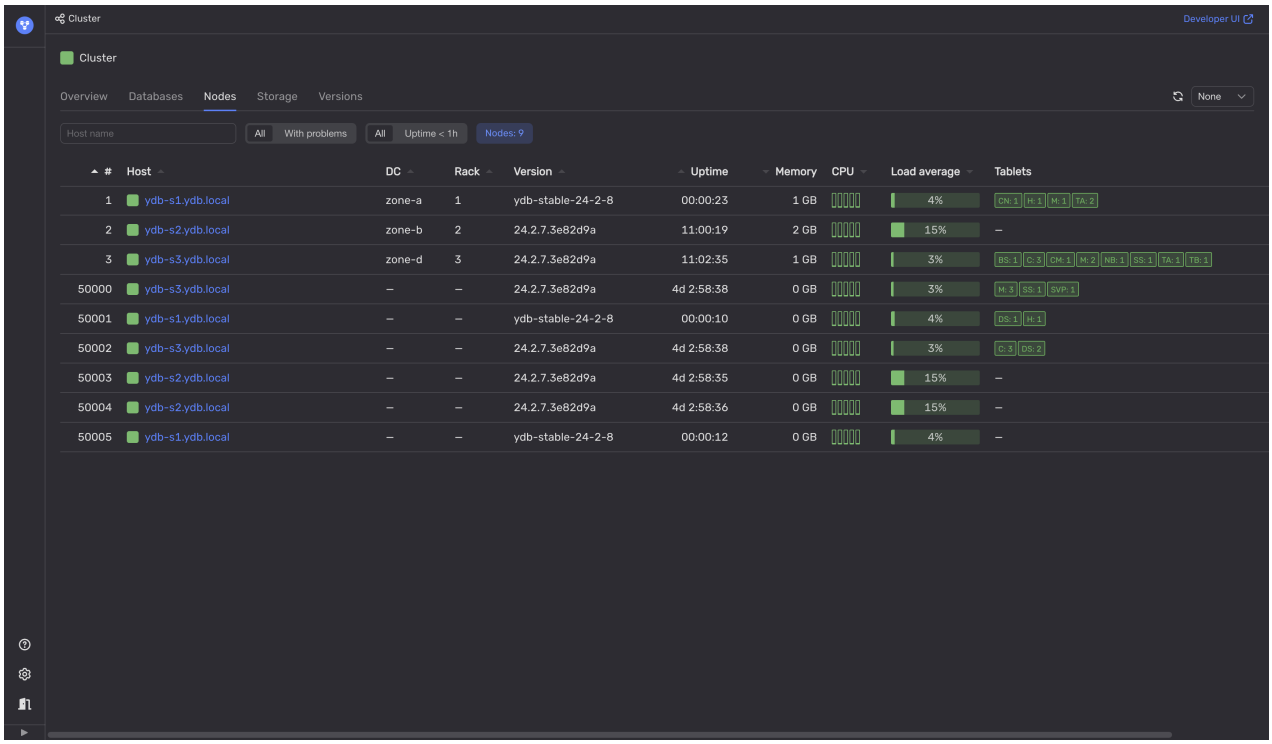
Важно

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 Встроенный 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;
- count.

Балансировка выполняется автоматически в следующих случаях:

- **Дисбаланс потребления аппаратных ресурсов**

  Для оценки сбалансированности потребления аппаратных ресурсов YDB использует метрику **scatter**, вычисляемую отдельно для каждого ресурса по формуле

  где:

  - — максимум по потреблению данного ресурса среди всех узлов;
  - — минимум по потреблению данного ресурса среди всех узлов.

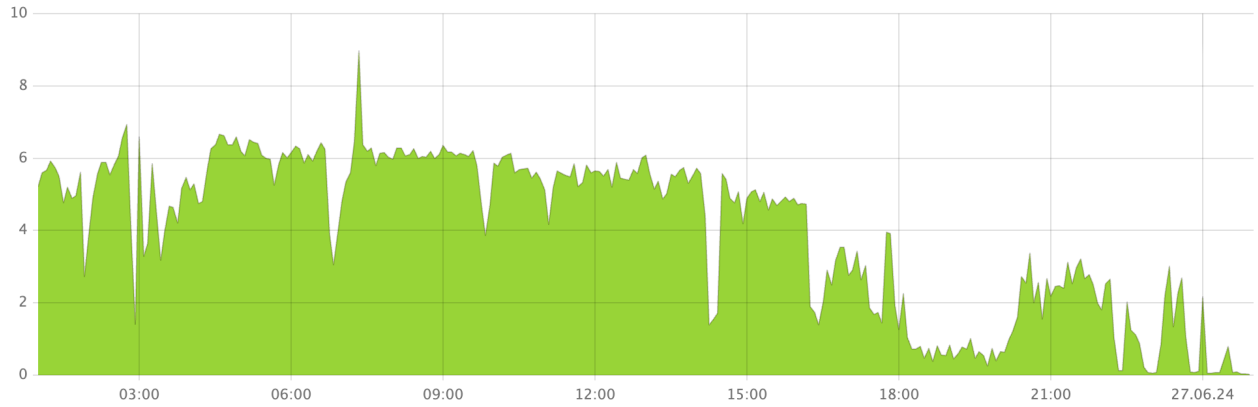
  Для нормирования потребления на каждом узле используется объём доступных ресурсов на узле, который может различаться между узлами. При низких нагрузках эта величина может сильно колебаться. Чтобы этого избежать, при вычислении **scatter** считается, что потребление ресурса не может быть ниже 30%. Если **scatter** превышает порог, запускается балансировка.
- **Перегруженность узла (по использованию процессора и памяти)**

  Hive запускает балансировку, если самый загруженный узел имеет загрузку больше 90%, а наименее загруженный — меньше 70%.
- **Неравномерное распределение конкретного объекта**

  Для таблеток, которые используют ресурс **count**, также отслеживается равномерность распределения таблеток каждого объекта (каждой таблицы), с помощью метрики ObjectImbalance, аналогичной описанной выше **scatter**. При рестартах узлов равномерность может нарушаться, и тогда запускается балансировка.

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 Встроенный UI.
   2.2. Нажмите на ссылку **Developer UI** в правом верхнем углу Встроенного 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			

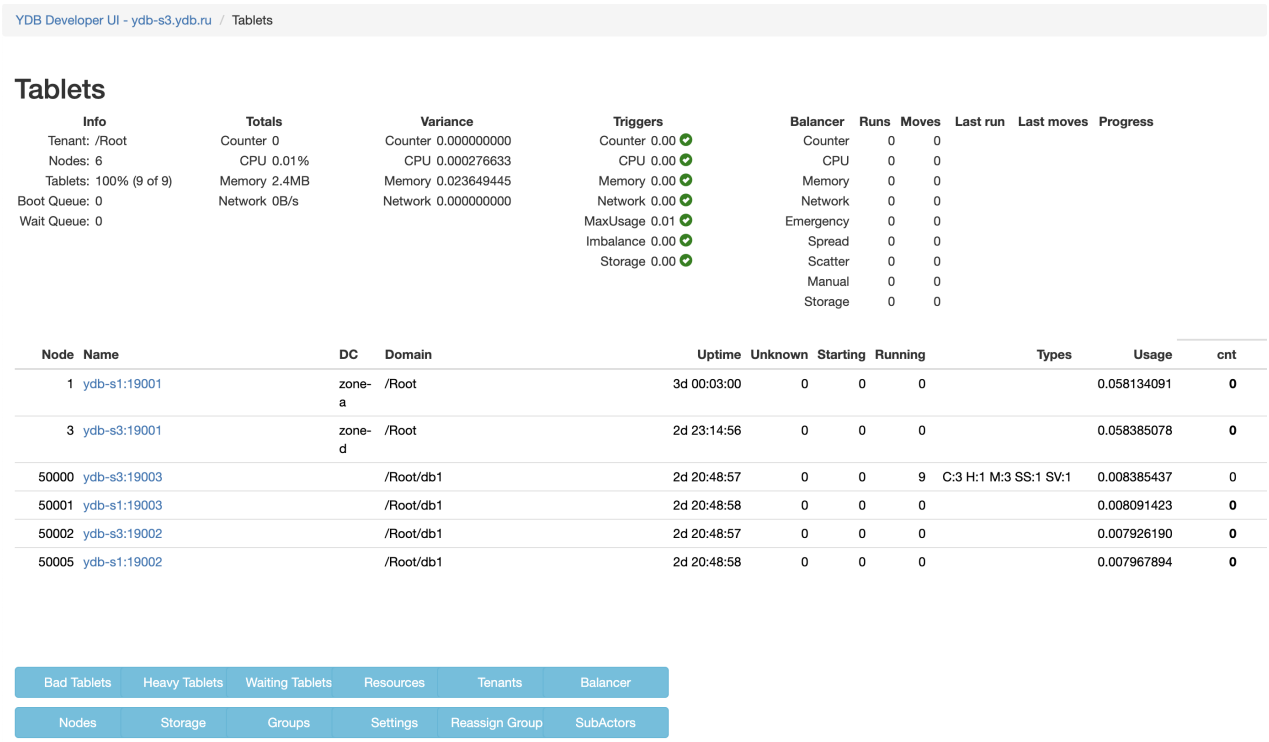
2.4. Кроме того, чтобы просмотреть недавно перемещённые таблетки, нажмите на кнопку **Balancer**.

Откроется окно **Balancer**, в котором список недавно перемещённых таблеток отображается в секции **Latest tablet moves**.

Рекомендации

Измените настройки балансировщика Hive:

1. Откройте Встроенный UI.
2. Нажмите на ссылку **Developer UI** в правом верхнем углу Встроенного 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

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 Встроенный UI или Grafana, чтобы проверить, не перегружены ли узлы YDB:
   - На панели мониторинга Grafana **DB overview** проанализируйте диаграмму **Overloaded shard count**.

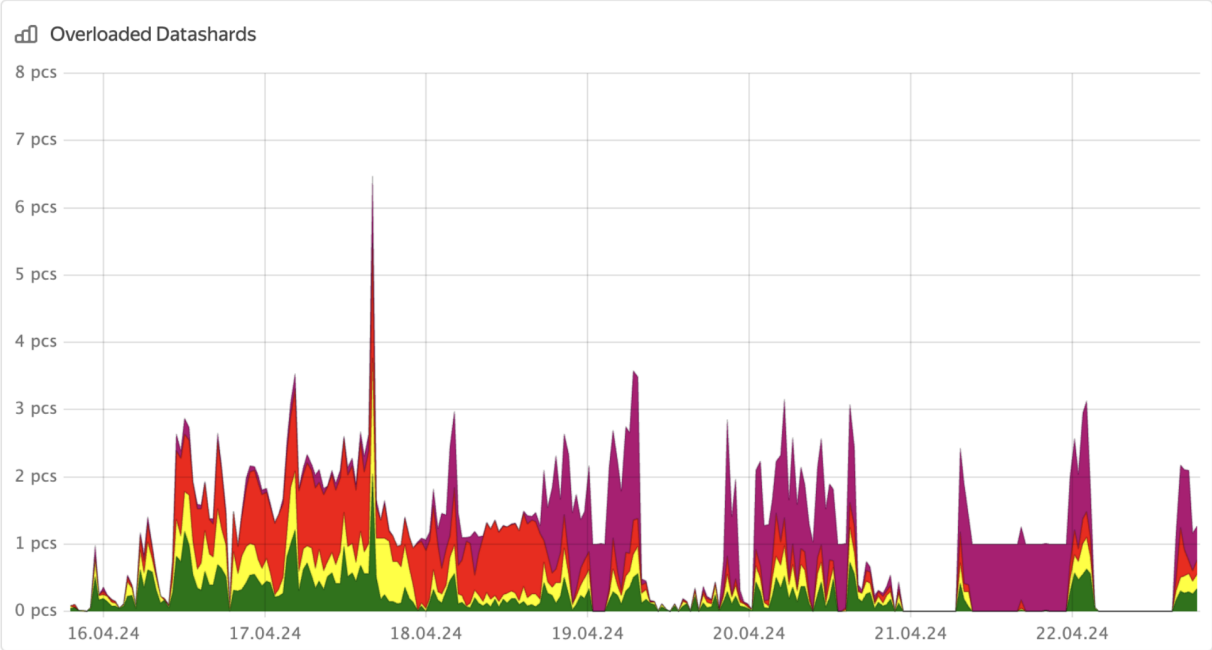
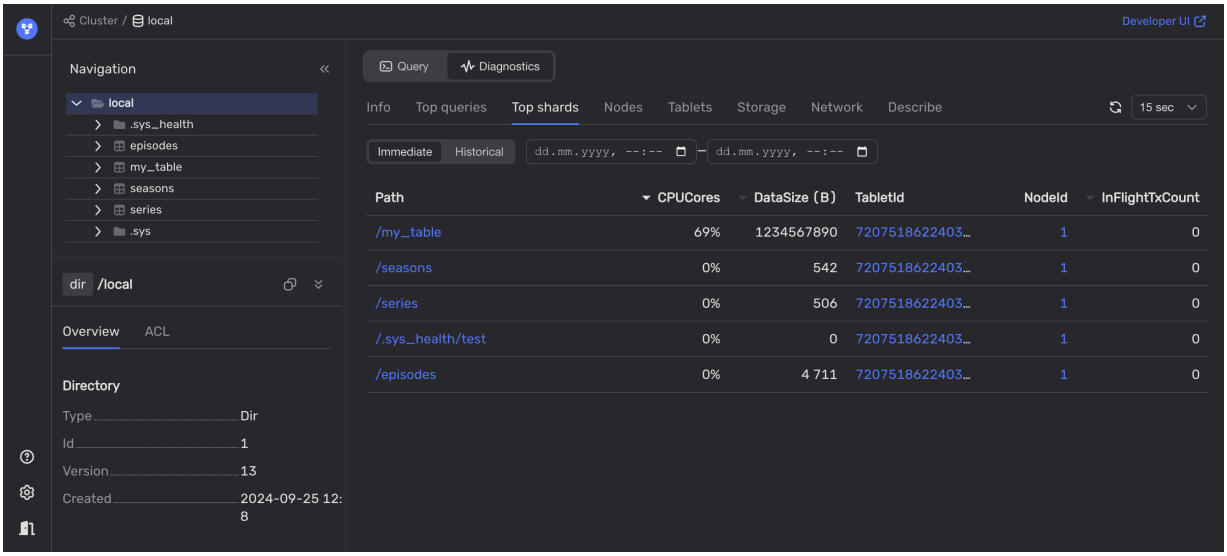


     Диаграмма отображает перегруженные таблетки data shard в кластере YDB, но она не показывает таблицы, в которых есть перегруженные таблетки.

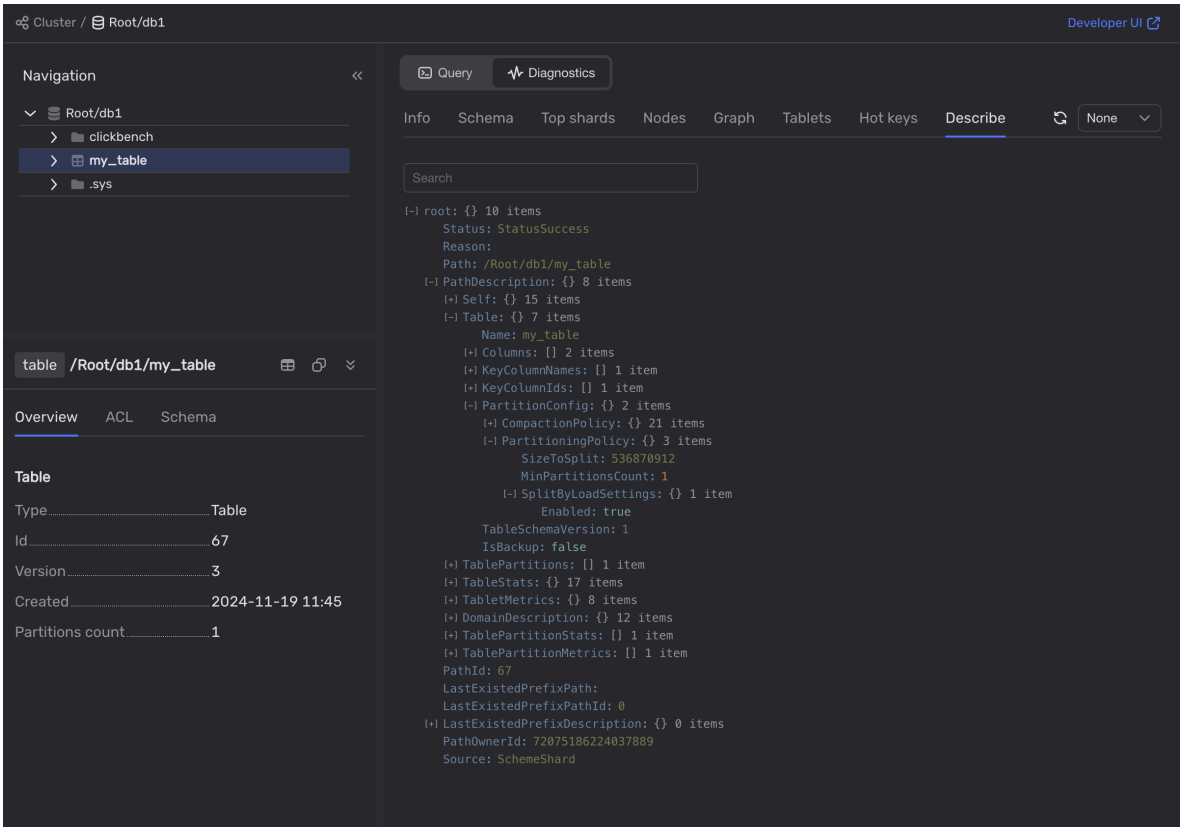
     > **Совет**
     >
     > Настройте уведомления в Grafana о перегрузках таблеток data shard.

   - Во Встроенном 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 Встроенный UI или YDB CLI:
   - Во Встроенном 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 **Встроенном 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 **Встроенном 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

> **Внимание**
>
> 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%.

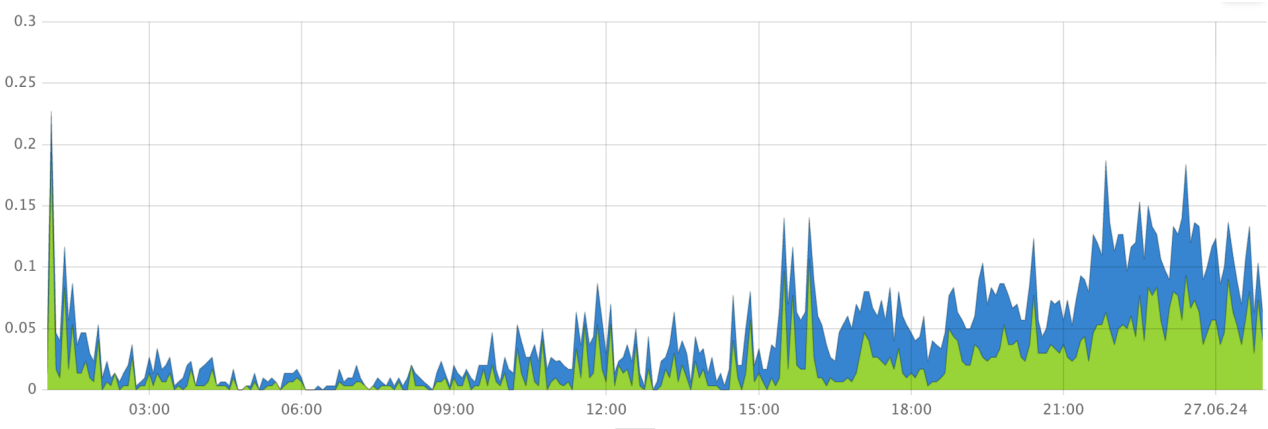
Когда YDB объединяет соседние партиции в строковой таблице, они заменяются одной партицией, которая охватывает их диапазон первичных ключей. Соответствующие таблетки data shard также объединяются в одну таблетку для управления новой партицией.

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

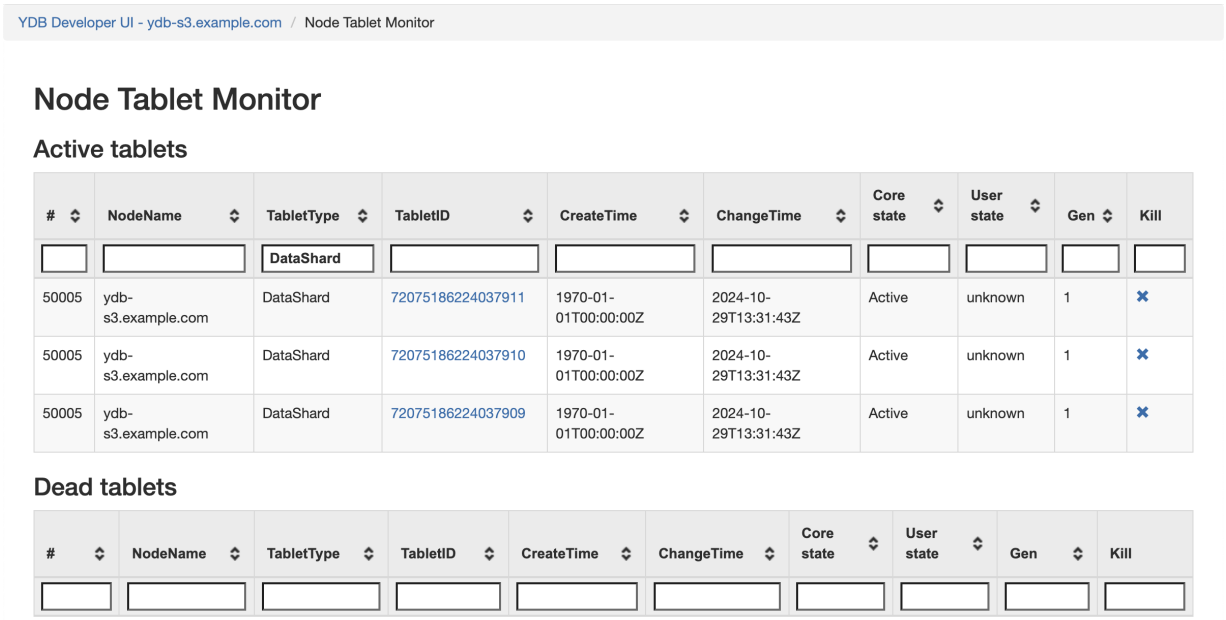
1. Посмотрите, есть ли всплески на графике **Split / Merge partitions** на панели мониторинга Grafana **DB status**.



   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 Встроенном 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.

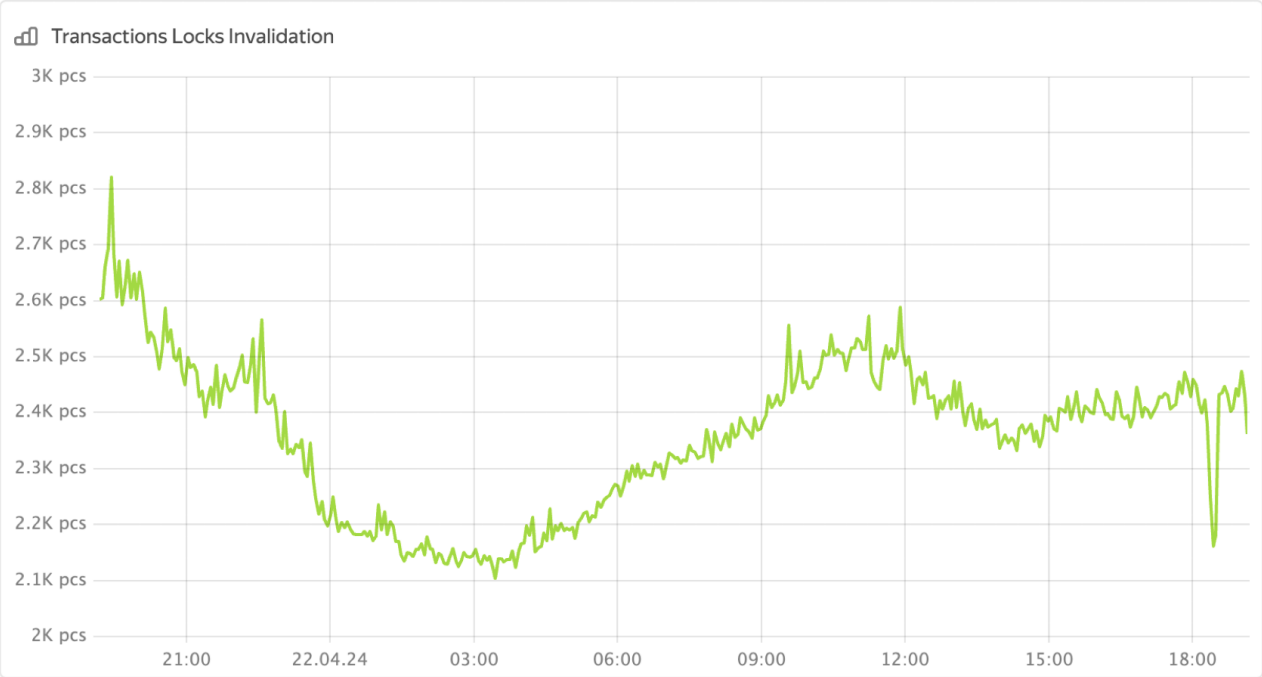
Инвалидация блокировок транзакций

YDB использует optimistic concurrency control для обнаружения конфликтов с другими выполняющимися транзакциями. Если проверка блокировок в стадии коммита обнаруживает конфликтующие изменения, транзакция откатывается и должна быть выполнена заново. В этом случае YDB возвращает ошибку `transaction locks invalidated`. Повторное выполнение значительного количества транзакций может замедлить ваше приложение.

> **Примечание**
>
> YDB SDK предоставляет встроенный механизм обработки временных ошибок. Подробнее см. Обработка ошибок.

Диагностика

1. Откройте панель мониторинга **DB overview** в Grafana.
2. Проверьте, есть ли всплески количества ошибок на диаграмме **Transaction Locks Invalidation**.



Эта диаграмма отображает количество запросов в секунду, возвращаемых с ошибкой «transaction locks invalidation».

Рекомендации

Примите во внимание следующие рекомендации:

- Чем дольше длится транзакция, тем выше вероятность возникновения ошибки `transaction locks invalidated`.

  По возможности избегайте интерактивных транзакций. Лучшим подходом является использование одного YQL-запроса с командами `BEGIN;` и `COMMIT;` для выбора данных, обновления данных и выполнения коммита транзакции.

  Если без интерактивных транзакций не обойтись, выполняйте коммит транзакции в последнем запросе.
- Проанализируйте диапазон первичных ключей, в которых происходят конфликтующие изменения, и попытайтесь изменить логику приложения, чтобы уменьшить количество конфликтов.

  Например, если одна строка с общим балансовым значением часто обновляется, разделите эту строку на сто строк и рассчитайте общий баланс как сумму этих строк. Это значительно сократит количество ошибок `transaction locks invalidated`.

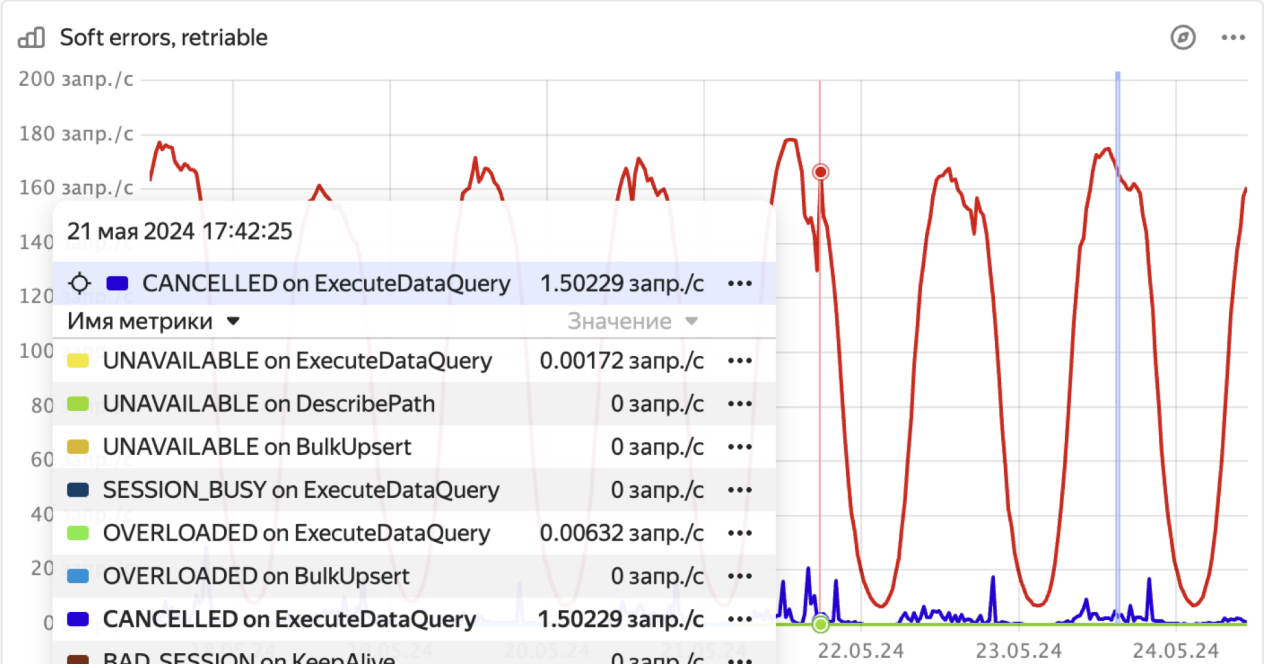
Ошибки «overloaded»

YDB возвращает ошибки `OVERLOADED` в следующих случаях:

- Перегруженные партиции таблиц, у которых в очереди на выполнение более 15000 запросов.
- Превышен лимит размера выходной очереди CDC в 10000 элементов или 125 МБ.
- Партиции таблиц не находятся в нормальном состоянии, например, разделяются/объединяются.
- Количество открытых сессий с узлом YDB достигло лимита в 1000.

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:
   3.1. Во Встроенном UI перейдите на вкладку **Databases** и нажмите на базу данных.
   3.2. На вкладке **Navigation** убедитесь, что требуемая база данных выбрана.
   3.3. Откройте вкладку **Diagnostics**.
   3.4. Откройте вкладку **Top shards**.
   3.5. На вкладках **Immediate** и **Historical** отсортируйте таблетки по столбцу **InFlightTxCount** и проверьте, не превышают ли максимальные значения лимит в 15000 запросов.
4. Чтобы проверить, не связаны ли всплески ошибок `OVERLOADED` со слишком частыми слияниями и разделениями таблеток, см. Избыточные разделения и слияния партиций таблиц.
5. 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**.
6. См. статью Перегруженные таблетки data shard.

Рекомендации

Если YQL-запрос возвращает ошибку `OVERLOADED`, выполните запрос повторно с экспоненциальной задержкой. YDB SDK предлагает встроенный механизм для обработки временных ошибок. Для получения дополнительной информации см. Обработка ошибок.

Превышение лимита открытых сессий на узле может указывать на проблему в логике приложения.

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 Embedded UI, чтобы найти решение, и проверим его эффективность.

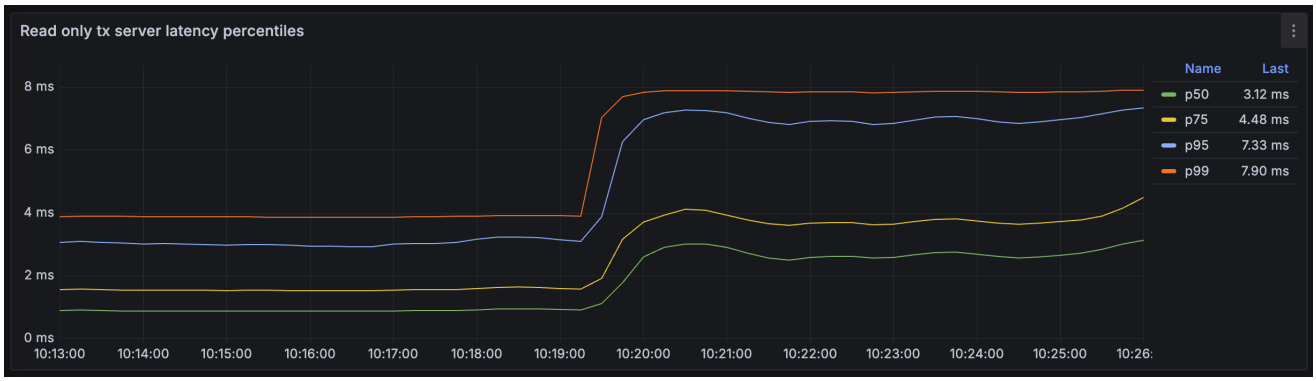
В конце статьи приводятся шаги по воспроизведению проблемы.

Описание проблемы

Вас уведомили о задержках при обработке пользовательских запросов в вашей системе.

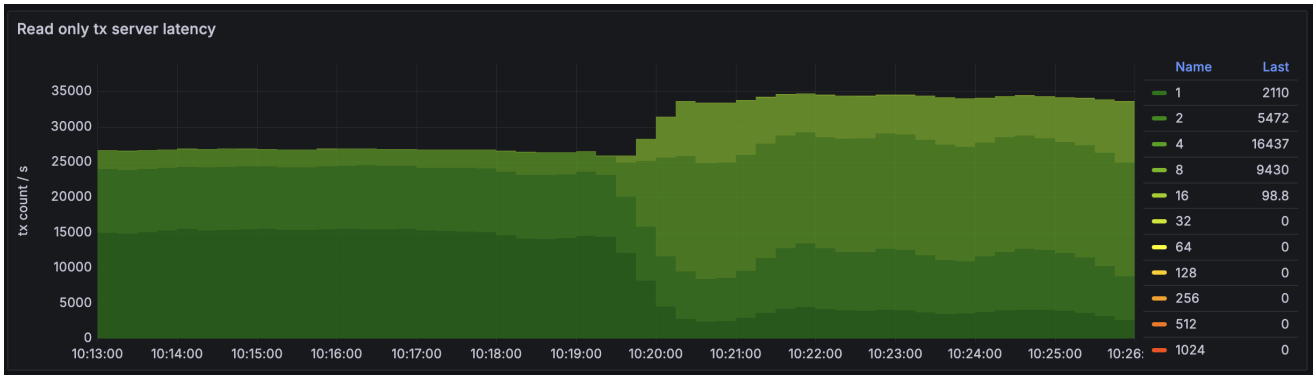
> **Примечание**
>
> Речь идёт о запросах к строковой таблице, управляемой data shard'ом.

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.



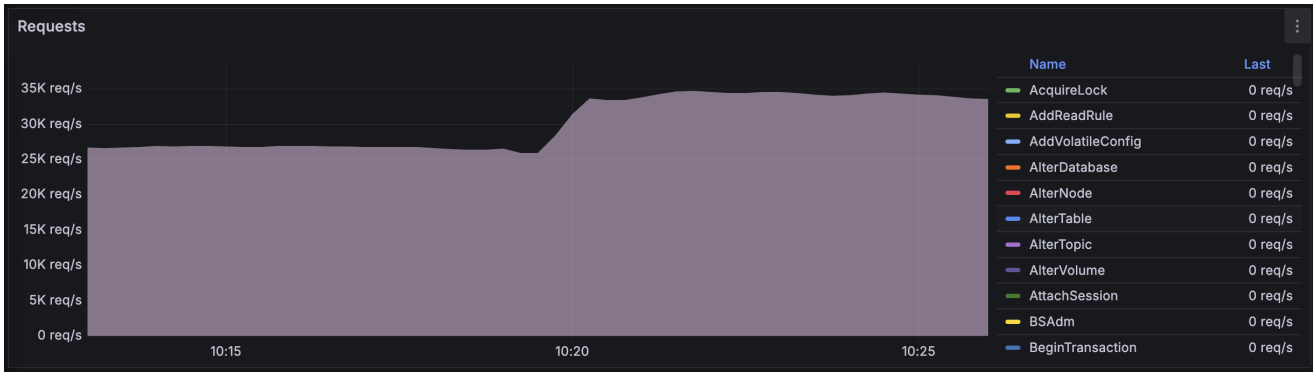
См. описание графика

> 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.

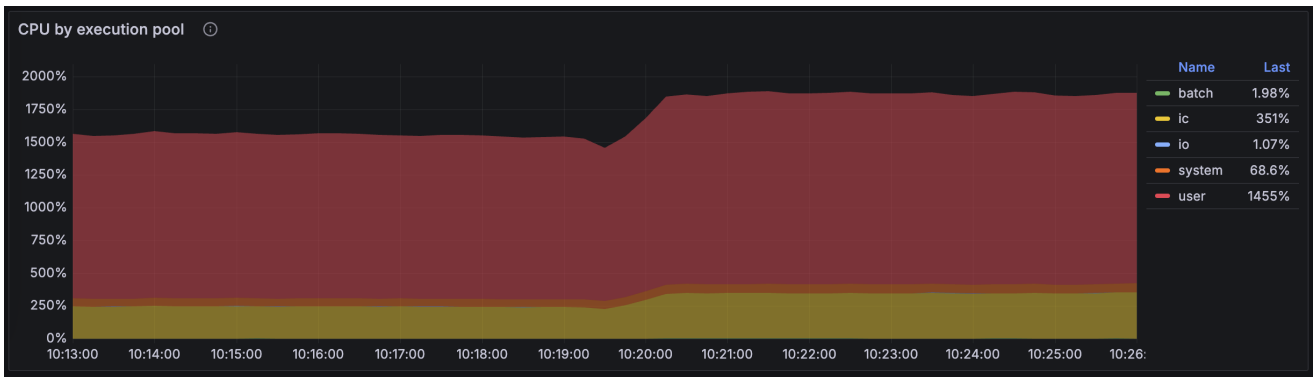
Диагностика

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:



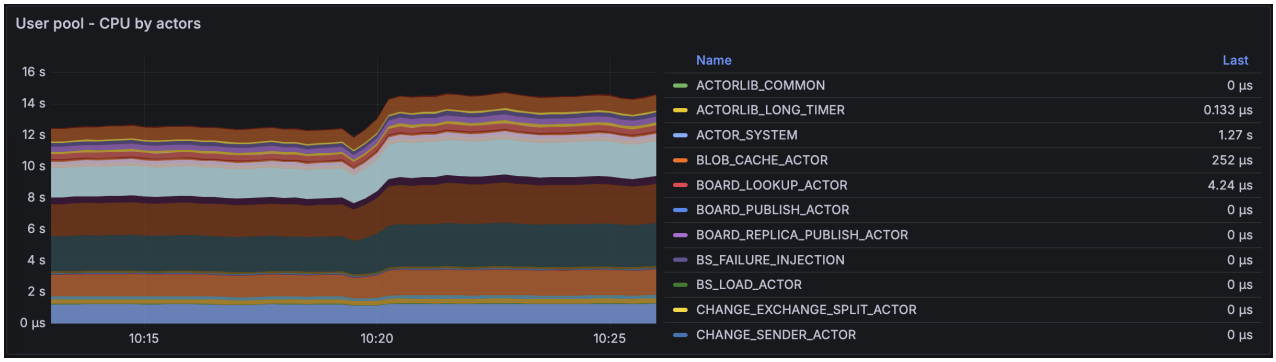
Количество пользовательских запросов выросло приблизительно с 27 000 до 35 000 в `10:20:00`. Но может ли YDB справиться с увеличившейся нагрузкой без дополнительных аппаратных ресурсов?

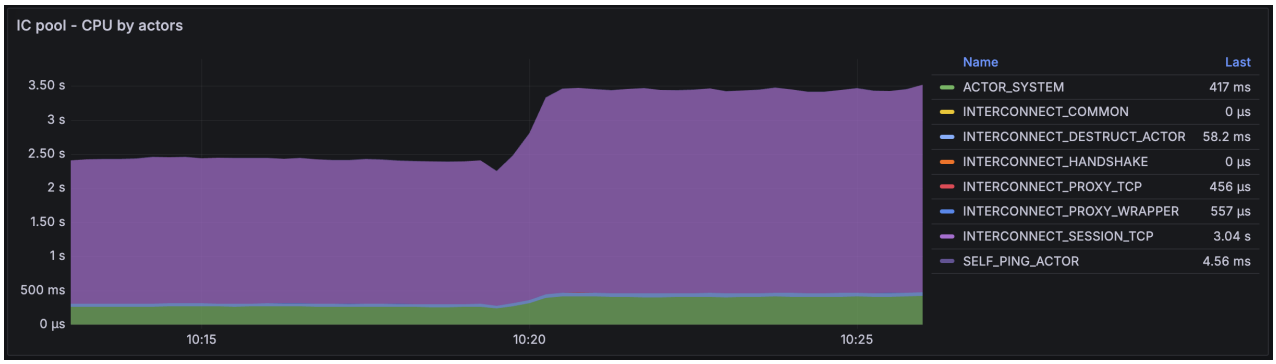
Загрузка CPU увеличилась, что видно на графике **CPU by execution pool**.

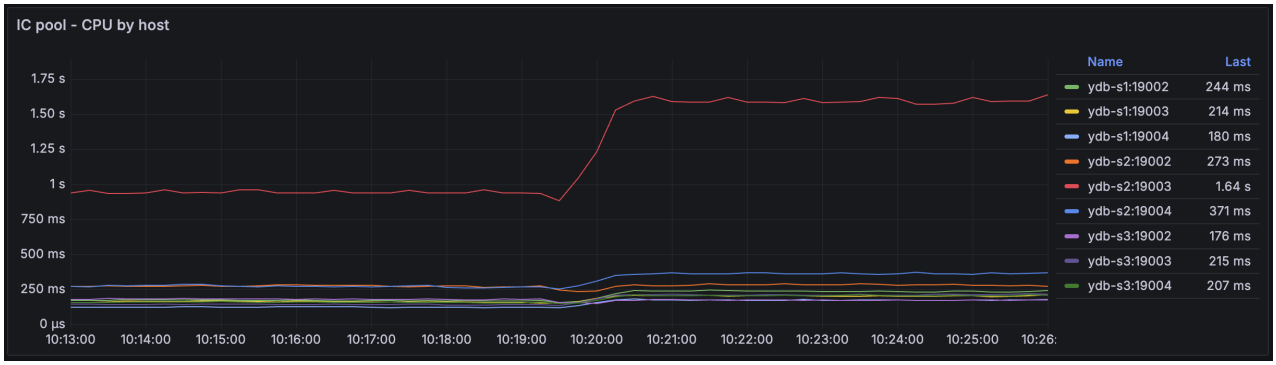


См. графики на панели мониторинга Grafana **CPU**

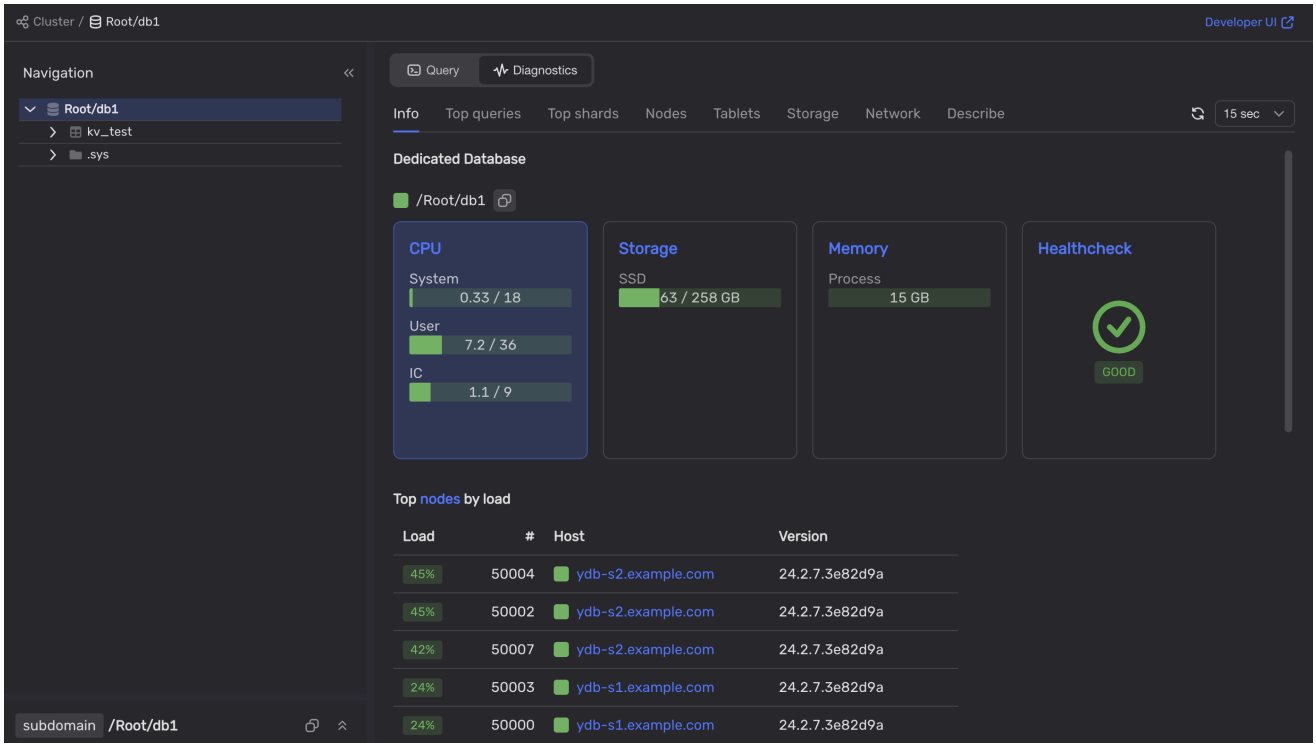
Графики на панели мониторинга Grafana **CPU** показывают рост нагрузки на CPU в пуле ресурсов пользователей и интерконнекта:





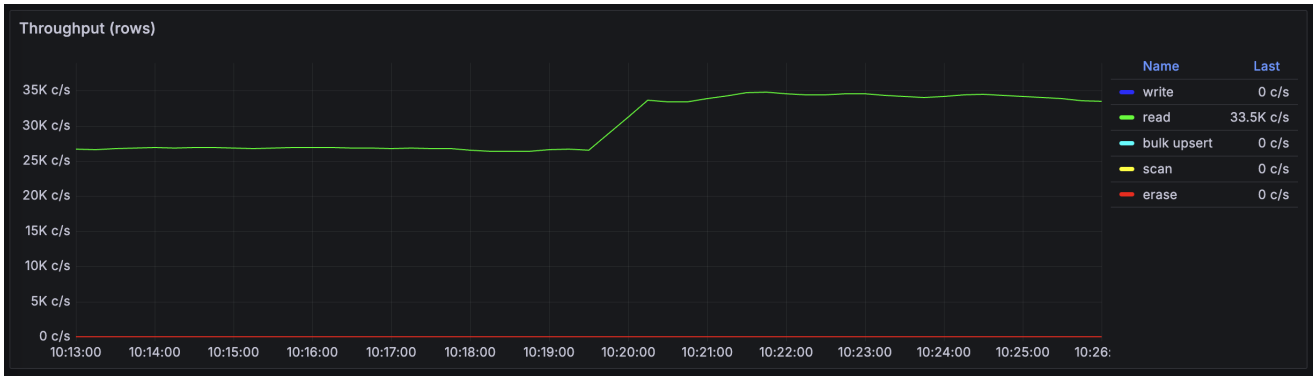


Мы также можем взглянуть на общее использование CPU на вкладке **Diagnostics** в Embedded UI:



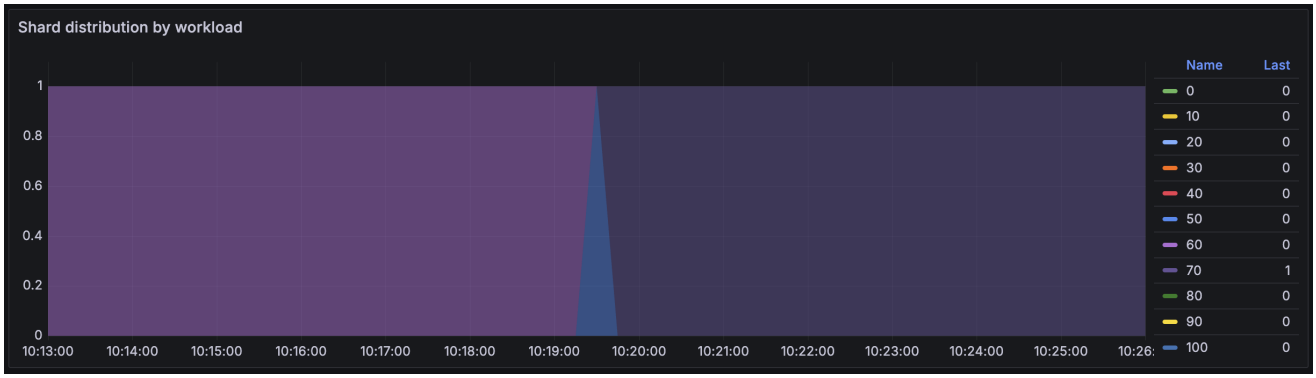
Кластер YDB не использует все ресурсы CPU.

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

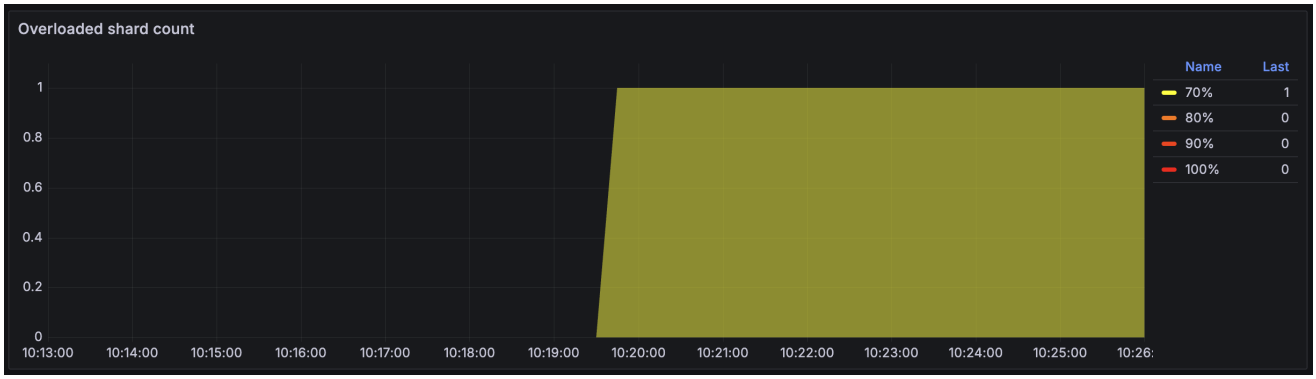
Этот график показывает, что количество читаемых строк в базе данных YDB увеличилось с ~26 000 до ~33 500 строк в секунду.



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%.

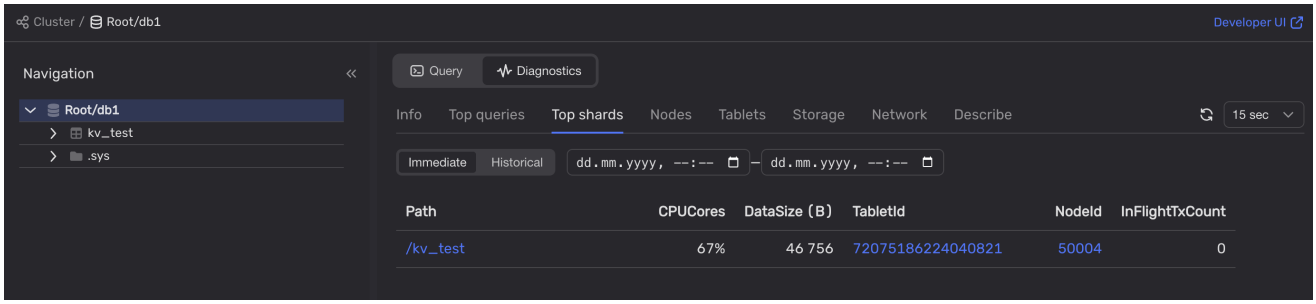


См. описание графика

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%.

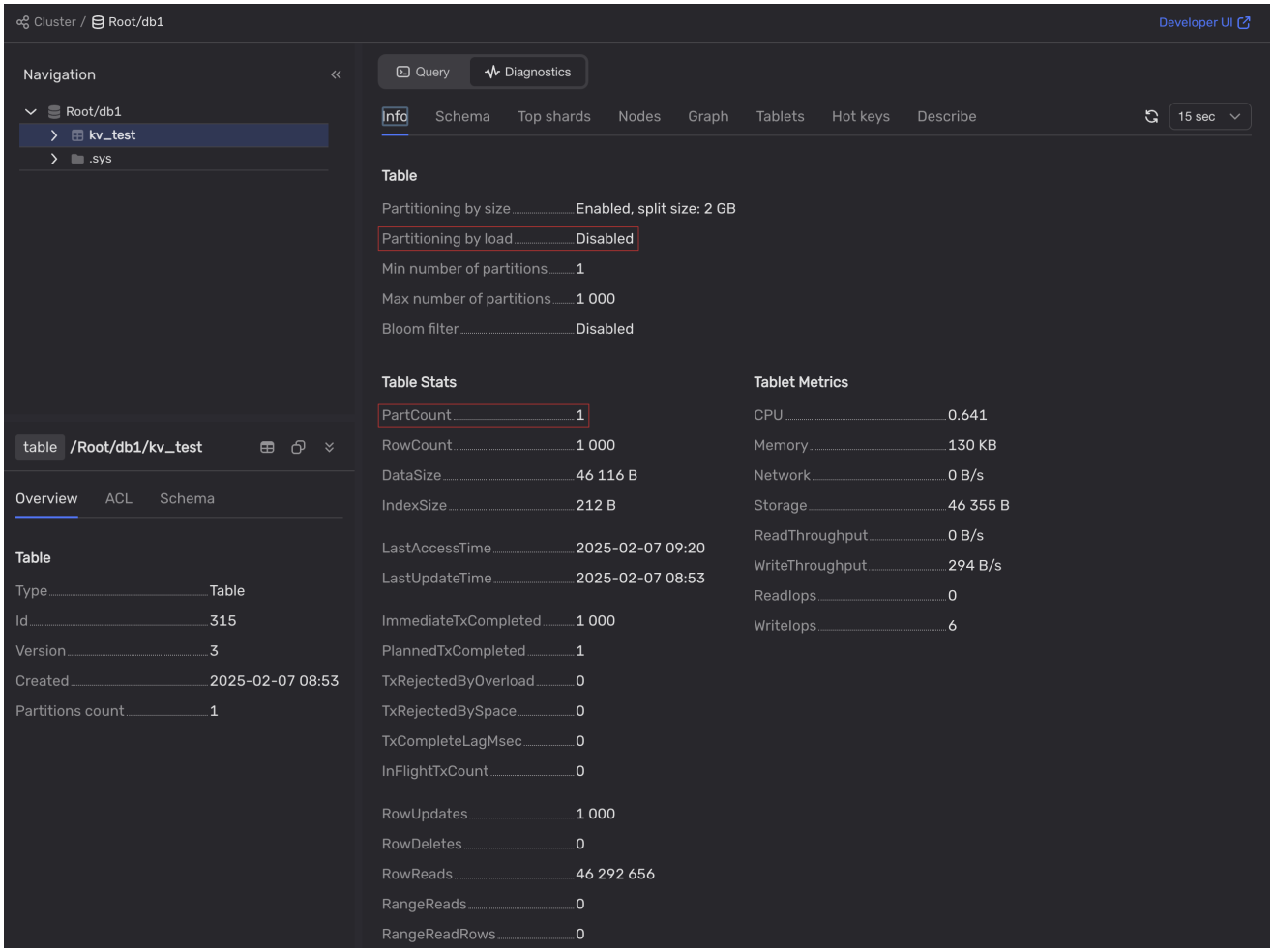
График показывает, что нагрузка на один data shard увеличилась до 70% примерно в `10:19:30`.

Чтобы определить, какую таблицу обслуживает перегруженный data shard, откроем вкладку **Diagnostics > Top shards** во встроенном 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

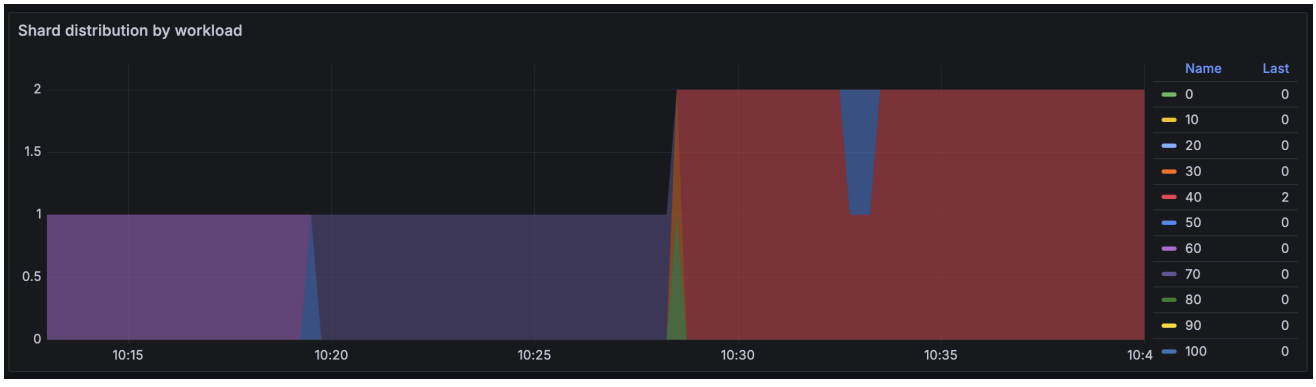
Нам необходимо включить партиционирование по нагрузке для таблицы `kv_test`:

1. Во встроенном UI выберите базу данных.
2. Откройте вкладку **Query**.
3. Выполните следующий запрос:

```
ALTER TABLE kv_test SET (
    AUTO_PARTITIONING_BY_LOAD = ENABLED
);
```

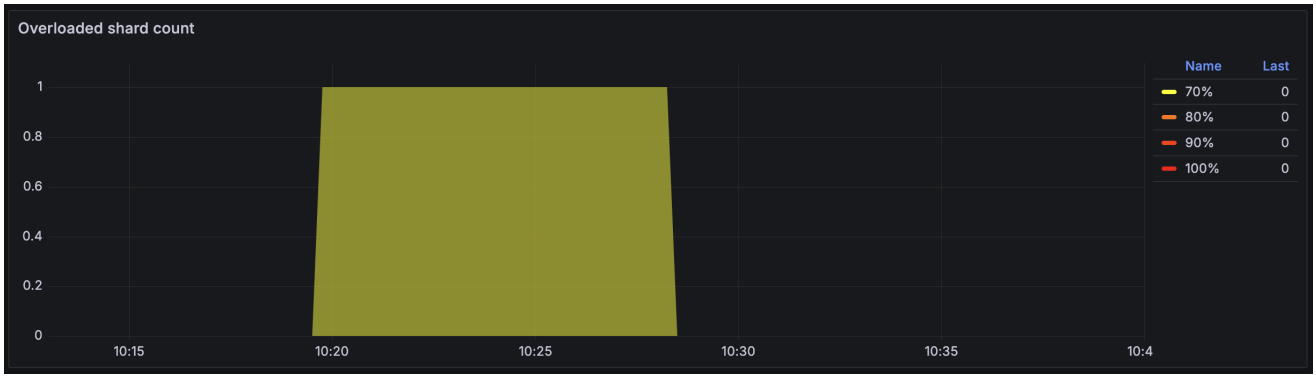
Результат

После включения автоматического партиционирования для таблицы `kv_test` перегруженный data shard разделился на два.



См. описание графика

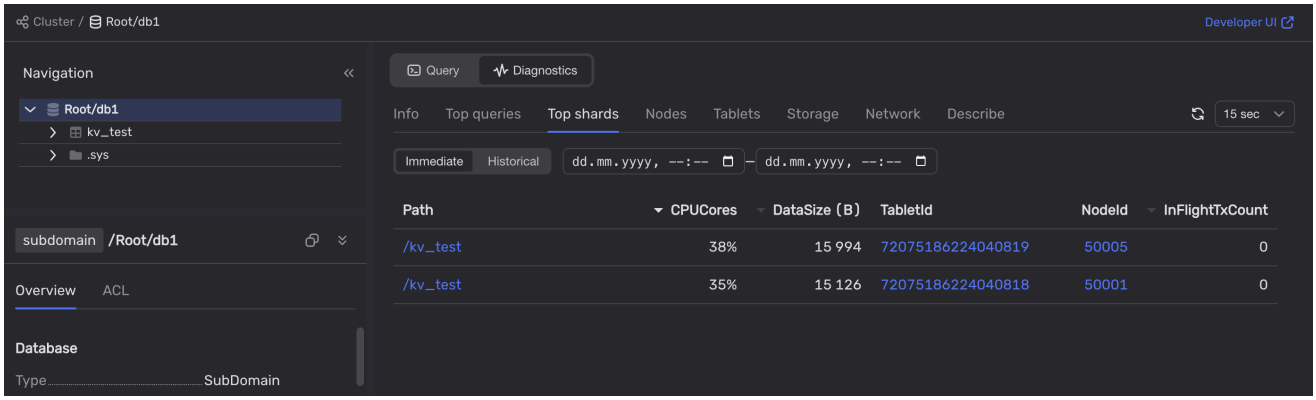
График показывает, что количество data shard'ов выросло примерно в `10:28:00`. Судя по цвету групп, их нагрузка не превышает 40%.



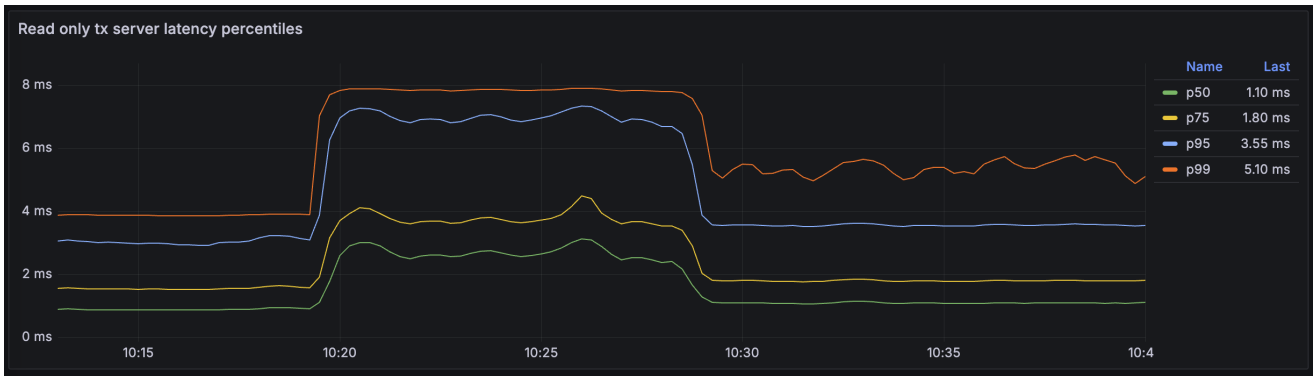
См. описание графика

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:

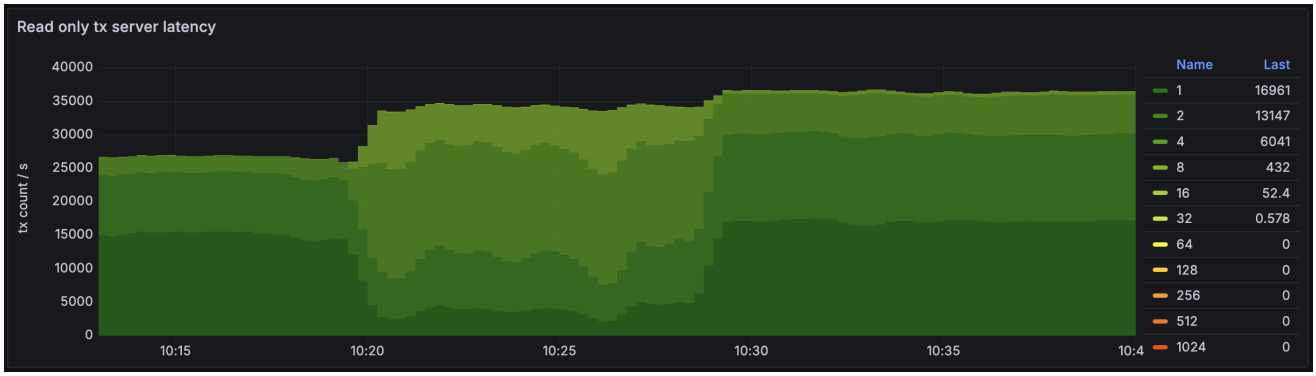


Давайте убедимся, что задержки транзакций вернулись к прежним значениям:



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

ере был запущен один узел

ормацию см. в статье Нагрузочное

```
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

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 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), ...)?

Это лучше записывать через JOIN с константной таблицей:

```
$keys = AsList(
    AsStruct(1 AS Key1, "One" AS Key2),
    AsStruct(2 AS Key1, "Three" AS Key2),
    AsStruct(4 AS Key1, "One" AS Key2)
);

SELECT t.* FROM AS_TABLE($keys) AS k
INNER JOIN table1 AS t
ON t.Key1 = k.Key1 AND t.Key2 = k.Key2;
```

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.

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 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), ...)?

Это лучше записывать через JOIN с константной таблицей:

```
$keys = AsList(
    AsStruct(1 AS Key1, "One" AS Key2),
    AsStruct(2 AS Key1, "Three" AS Key2),
    AsStruct(4 AS Key1, "One" AS Key2)
);

SELECT t.* FROM AS_TABLE($keys) AS k
INNER JOIN table1 AS t
ON t.Key1 = k.Key1 AND t.Key2 = k.Key2;
```

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.

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	Ссылка	Список изменений
v24.4			
v.24.4.4.12	03.06.25	https://github.com/ydb-platform/ydb/tree/24.4.4.12	См. список
v.24.4.4.2	15.04.25	https://github.com/ydb-platform/ydb/tree/24.4.4.2	См. список
v24.3			
v.24.3.15.5	06.02.25	https://github.com/ydb-platform/ydb/tree/24.3.15.5	См. список
v.24.3.11.14	09.01.25	https://github.com/ydb-platform/ydb/tree/24.3.11.14	См. список
v.24.3.11.13	24.12.24	https://github.com/ydb-platform/ydb/tree/24.3.11.13	См. список
v24.2			
v.24.2.7	20.08.24	https://github.com/ydb-platform/ydb/tree/24.2.7	См. список
v24.1			
v.24.1.18	31.07.24	https://github.com/ydb-platform/ydb/tree/24.1.18	См. список
v23.4			
v.23.4.11	14.05.24	https://github.com/ydb-platform/ydb/tree/23.4.11	См. список
v23.3			

v.23.3.17	14.12.23	https://github.com/ydb-platform/ydb/tree/23.3.17	
v.23.3.13	12.10.23	https://github.com/ydb-platform/ydb/tree/23.3.13	См. список
v23.2			
v.23.2.12	14.08.23	https://github.com/ydb-platform/ydb/tree/23.2.12	См. список

педующих целях:

го функции 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

Версия	Дата выпуска	Скачать	Список изменений
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

Версия	Дата выпуска	Скачать	Список изменений

Версия	Дата выпуска	Скачать	Список изменений

ольные суммы | См. список

Версия	Дата выпуска	Скачать	Список изменений

ольные суммы | См. список

Версия	Дата выпуска	Скачать	Список изменений
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

вой подсистемой кластера YDB.

Для использования утилиты установите ydb-dstool.

Загрузка YDB Ops

YDB Ops (`ydbops`) — утилита командной строки для управления кластером YDB.

Linux

Версия	Дата выпуска	Скачать	Список изменений
Версия	Дата выпуска	Скачать	Список изменений
Версия	Дата выпуска	Скачать	Список изменений
v0.0.14	09/12/2024	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	Контрольные суммы
v0.14	30.11.2024	ydb-ansible-0.14.zip	ydb-ansible-0.14.txt
v0.10	01.08.2024	ydb-ansible-0.10.zip	ydb-ansible-0.10.txt
v0.9	20.07.2024	ydb-ansible-0.9.zip	ydb-ansible-0.9.txt
v0.8	10.07.2024	ydb-ansible-0.8.zip	ydb-ansible-0.7.txt
v0.7	03.05.2024	ydb-ansible-0.7.zip	ydb-ansible-0.7.txt

Видеозаписи

Видеозаписи с конференций и вебинаров. 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.

2025

YDB Topics — корпоративный брокер сообщений от Яндекса

General

YDB Topics — совместимый с Apache Kafka корпоративный брокер сообщений, являющийся частью СУБД Яндекса YDB. На вебинаре рассказали, 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.

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. Рассматривается 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.

2024

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

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

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) 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

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

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

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

2023

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

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 геораспределённой 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

2022

Интервью с Андреем Фомичевым

Overview

Андрей Фомичев (Руководитель YDB) рассказал о том, как он пришел в разработку баз данных, с чего начиналась YDB, 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.

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 исполняется 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 они спроектировали две системы подсчёта пользователей: общее число просмотров видео и количество текущих зрителей лайв-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, что нужно делать, чтобы их поведение было привычным для пользователя и 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 решали или митигировали, а также про вынесенные уроки и принятые решения. Доклад будет интересен разработчикам, интересующимся высоконагруженными системами.

Слайды

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

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 casesPractice

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

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

2020

Serverless: публичный запуск YDB

Overview

Сервис YDB вышел в общий доступ. Теперь можно выбрать режим работы с выделенными серверами или бессерверный режим.

В докладе:

- Выход сервиса YDB в общественный доступ.
- Запуск нового продукта YDB Serverless.
- Новые возможности YDB.
- Ценообразование YDB и YDB Serverless.

2019

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

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

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 двухфазной фиксации и детерминированных транзакций.

Слайды

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:

- Как реализован ACID механизм.
- Какие есть уровни изоляции.
- Как работают распределенные транзакции.
- Что такое таблетки и как они устроены.

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

2018

Три доклада о YDB с about:cloud

OverviewUse casesDatabase 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.

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 докладе рассказ 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

Статьи о YDB. 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.

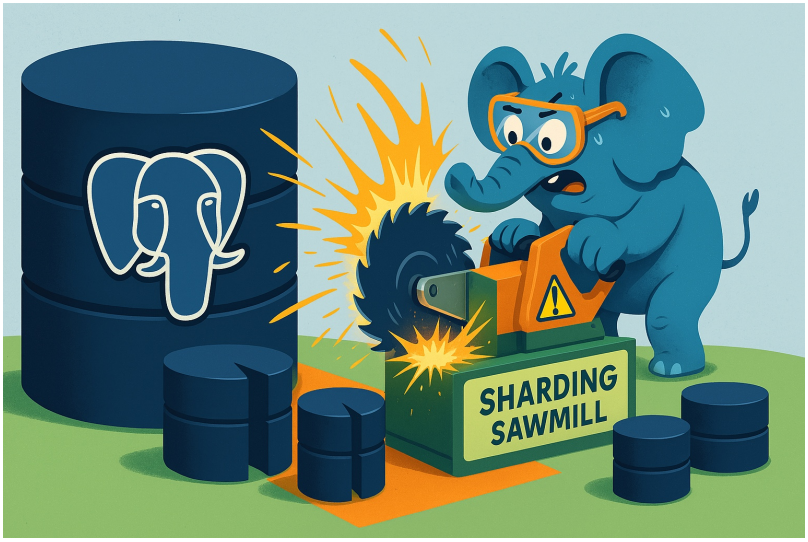
General

– материалы общего направления.

2025

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

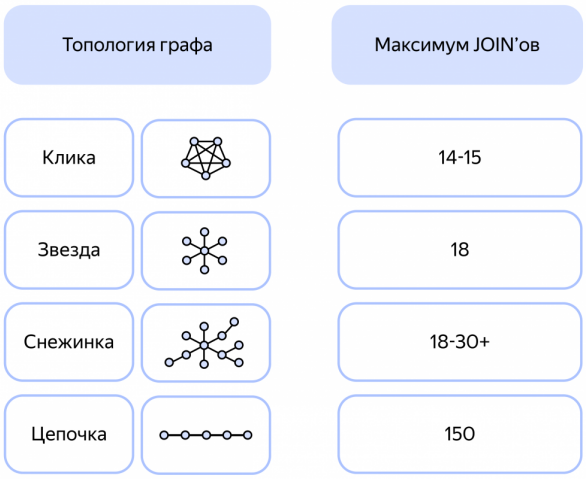
General

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

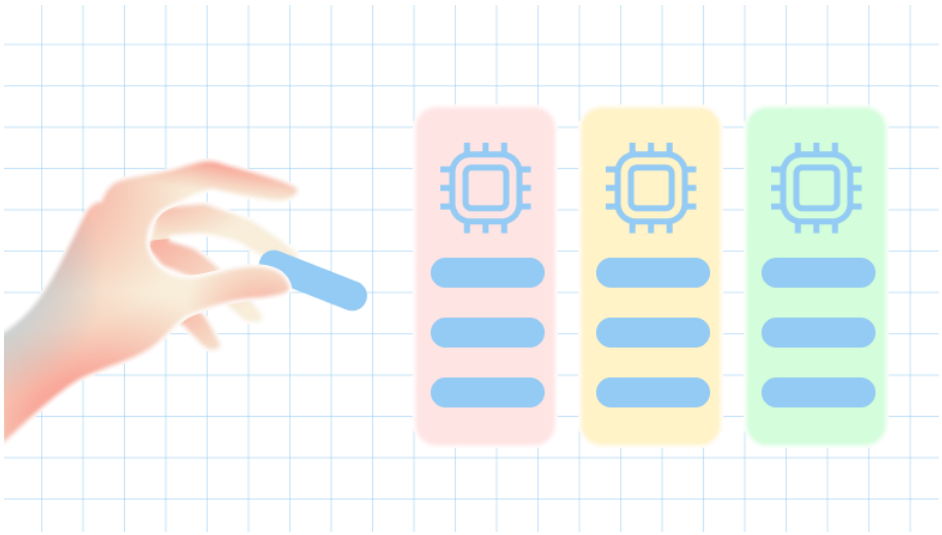


В статье рассматривается назначение, принципы работы и внутренняя реализация стоимостного (cost-based) оптимизатора запросов в гибридной базе данных YDB. Автор делится историей разработки собственной СУБД в Яндексе, объясняет отличия OLTP- и OLAP-нагрузок и обосновывает необходимость сложного оптимизатора 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. Автор 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

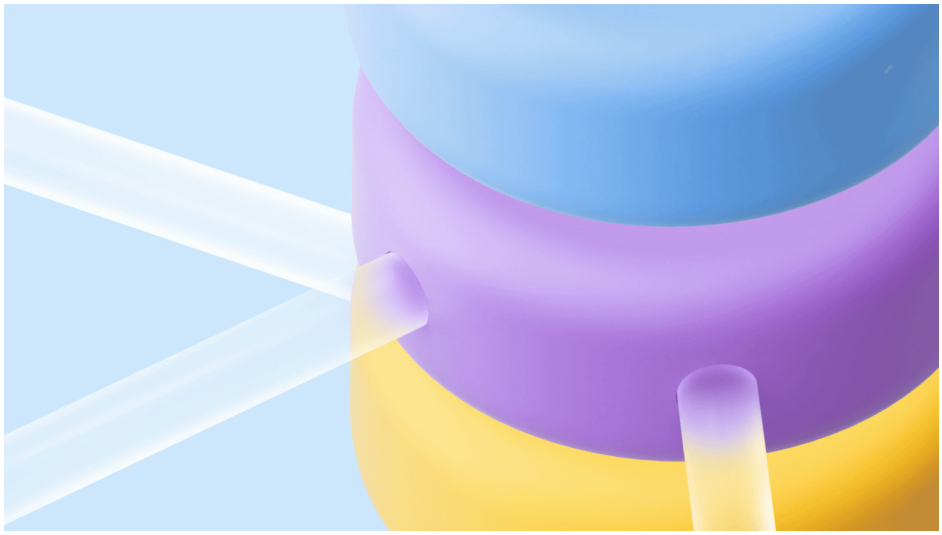
Practice

Статья представляет собой подробный туториал по использованию Kafka API для работы с потоками данных Yandex Data Streams (топиками YDB) в облачной инфраструктуре Yandex Cloud. Даётся пошаговая инструкция по созданию необходимых облачных ресурсов — базы данных, потоков, S3-бакета, сервисных аккаунтов и настройке доступа, а также приведены 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



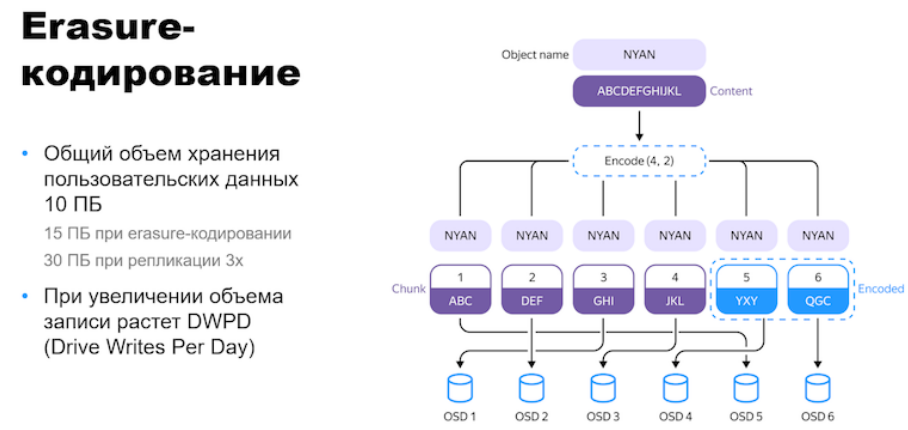
В статье студент рассказывает о практическом опыте крупного вклада в проект YDB — переводе парсера языка запросов YQL с ANTLR3 на ANTLR4. Автор объясняет, почему задача была технически сложной, какие трудности возникли при миграции инструментов и шаблонов, а также как удалось аккуратно внедрить изменения, 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.

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?

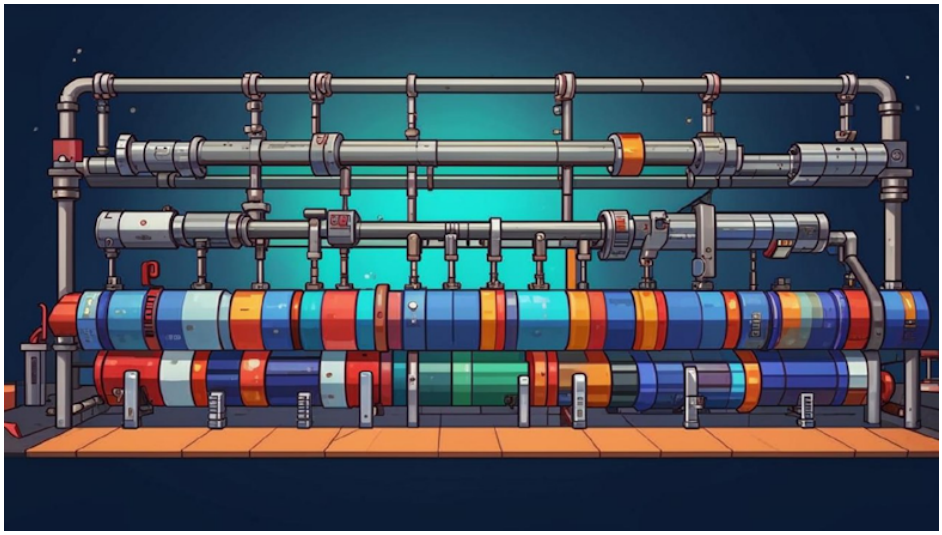
General

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

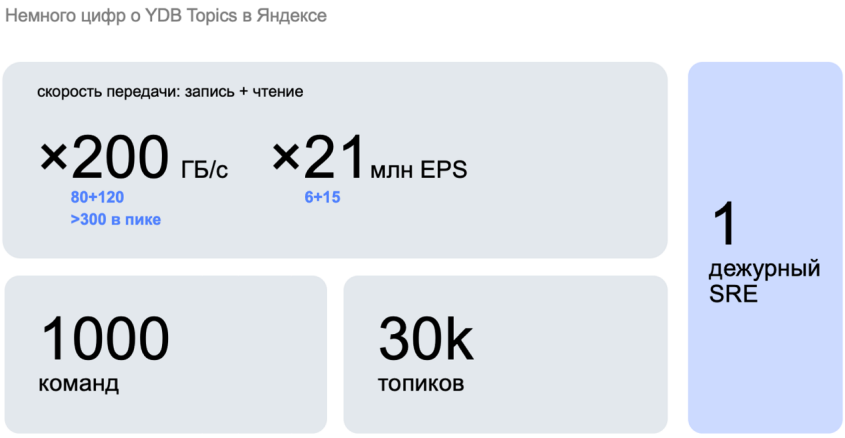
Database internals

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

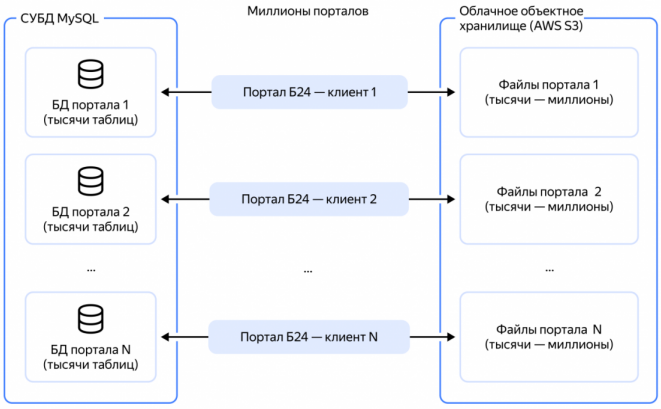
Practice

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

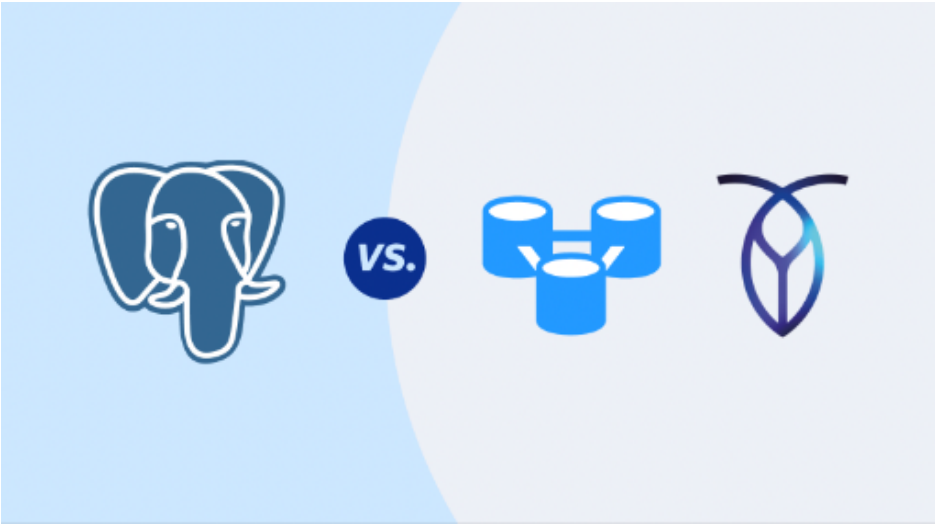


В статье на реальном кейсе компании Битрикс24 описан опыт миграции масштабируемой схемы резервного копирования и инкрементальной репликации файлов с AWS S3 в облако Yandex Cloud. Автор поэтапно разбирает эволюцию архитектуры, методы инкрементального и realtime-бэкапа, подходы к отложенному удалению файлов 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

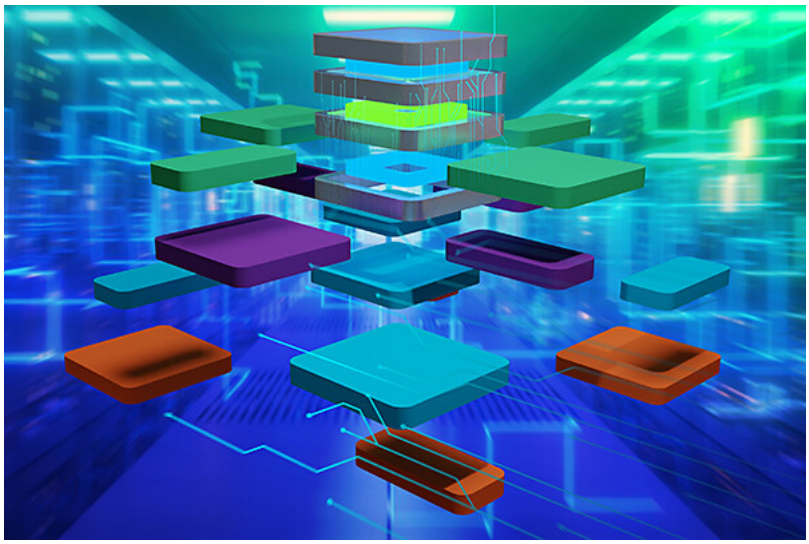
Database internals

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.

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:

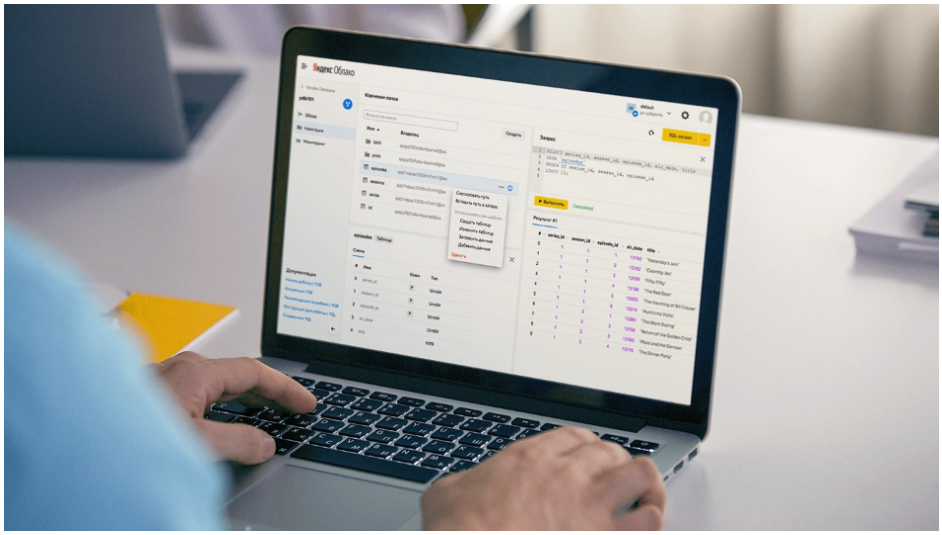
- Почему вообще YDB пошла в Open Source?
- Какими возможностями обладает YDB?
- Как развивался проект?
- В каких сценариях YDB показывает себя сопоставимо с конкурентами, а в каких их превосходит?

Ознакомиться со статьёй можно на сайте OSP.

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.

2025

sysadmins №54. YDB

General

В подкасте linkmeup руководитель проектного офиса YDB Антон Коваленко 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. YDB

Список изменений YDB Server

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 отказоустойчивость 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 `KQP` 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'e, 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 `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 фолловерами 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 фолловерах 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 YDB CLI

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'a 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.