Общество с ограниченной ответственностью «Платформа Данных Селена» 123022, г. Москва, вн.тер.г. муниципальный округ Пресненский, ул. 2-я Звенигородская, д. 13, стр. 37 ИНН 9703195303, КПП 770301001, ОГРН 1247700725491

Платформа Данных Селена

РУКОВОДСТВО РАЗРАБОТЧИКА

Версия программы 3.0

Москва
2026

Аннотация

Данное руководство — практический справочник разработчика по Платформе Данных Селена (далее Selena). Здесь собраны сведения, к которым чаще всего приходится обращаться в процессе написания запросов и интеграций: типы данных, SQL-функции и операторы, системные ограничения и переменные, коды ошибок, а также шаблоны SQL-инструкций и описания параметров.

© ООО «Платформа Данных Селена», 2026

Оглавление

1 Обзор типов данных

1.1 Numeric
1.2 String
1.3 Типы данных даты и времени
1.4 Semi-structured
1.5 Other

2 Функции

2.1 Java UDF
2.2 Лямбда-выражения
2.3 Оконные функции
2.4 CAST
2.5 Hive Bitmap UDF
2.6 Python UDF
2.7 Aggregate
2.8 Array
2.9 Binary
2.10 Bit
2.11 Bitmap
2.12 Condition
2.13 Cryptographic
2.14 Функции даты и времени
2.15 Dictionary
2.16 Hash
2.17 JSON
2.18 Pattern matching
2.19 Map
2.20 Math
2.21 Percentile
2.22 Scalar
2.23 Spatial
2.24 String
2.25 Struct
2.26 Table
2.27 Utility
meta-functions

3 SQL statements

3.1 User account management
3.2 Cluster management
3.3 Loading and unloading
3.4 Catalog

© ООО «Платформа Данных Селена», 2026

3.5 Database
3.6 Resource
3.7 Table, bucket, partition, and index
3.8 View
3.9 Materialized View
3.10 Function
3.11 CBO statistics
3.12 Backup and restore
3.13 Translate Trino SQL dictionary
3.14 Генерируемые столбцы
3.15 Ключевые слова
3.16 Подготовленные операторы (Prepared statements)

4 Коды ошибок

5 Convert_tz

6 Шаблон SQL-инструкции

7 Системные ограничения

8 Системные переменные

9 AWS IAM policies

10 HTTP SQL API

11 Information Schema

11.3 Analyze_status
11.4 Applicable_roles
11.5 Be_bvars
11.6 Be_cloud_native_compactions
11.7 Be_compactions
11.8 Be_configs
11.9 Be_logs
11.10 Be_metrics
11.11 Be_tablets
11.12 Be_threads
11.13 Be_txns
11.14 Character_sets
11.15 Collations
11.16 Column_privileges
11.17 Column_stats_usage
11.18 Columns
11.19 Engines
11.20 Events
11.21 Fe_metrics

11.22 Fe_tablet_schedules
11.23 Global_variables
11.24 Key_column_usage
11.25 Load_tracking_logs
11.26 Loads
11.27 Materialized_views
11.28 Partitions
11.29 Partitions_meta
11.30 Pipe_files
11.31 Pipes
11.32 Recyclebin_catalogs
11.33 Referential_constraints
11.34 Routine_load_jobs
11.35 Routines
11.36 Schema_privileges
11.37 Schemata
11.38 Session_variables
11.39 Statistics
11.40 Stream_loads
11.41 Table_constraints
11.42 Table_privileges
11.43 Tables
11.44 Tables_config
11.45 Task_runs
11.46 Tasks
11.47 Triggers
11.48 User_privileges
11.49 Verbose_session_variables
11.50 Views
11.51 Warehouse_metrics
11.52 Warehouse_queries

12 Системная метабаза данных

12.3 Fe_locks
12.4 Grants_to_roles
12.5 Grants_to_users
12.6 Object_dependencies
12.7 Policy_references
12.8 Role_edges

13 Шаблон для описания параметров FE/BE и переменных

14 Пользовательские переменные

1 Обзор типов данных

Selena поддерживает следующие типы данных: Числовые, Строковые, Дата и время, Полуструктурированные и Другие. Вы можете определить столбцы этих типов при создании таблицы.

1.1 Numeric

BIGINT

BIGINT — это 8-байтовое целое число со знаком. Диапазон значений: [-9223372036854775808, 9223372036854775807].

BOOLEAN

BOOL, BOOLEAN

Как и в TINYINT, 0 означает false, а 1 означает true.

DECIMAL

DECIMAL(P[,S]) — это значение с фиксированной точкой высокой точности. `P` обозначает общее количество значащих цифр (precision). `S` обозначает максимальное количество десятичных знаков (scale).

Если `P` не указан, значение по умолчанию — 10. Если `S` не указан, значение по умолчанию — 0.

- Для версий до v2.0.0 Selena поддерживает DECIMAL128 на основе Fast DECIMAL.
- Для версий v2.0.0 и более поздних Selena поддерживает DECIMAL256.

Возможности

Fast DECIMAL

Управляется динамическим параметром FE `enable_decimal_v3`, Fast DECIMAL включен по умолчанию.

Для Fast DECIMAL диапазон `P` составляет [1,38], а диапазон `S` — [0, P]. Значение `S` по умолчанию — 0.

Fast DECIMAL использует целые числа переменной ширины для представления десятичных чисел.

- Decimal(P ≤ 18, S)
  - LogicalType: Decimal64
  - Stored as: int64
  - Delegate LogicalType: BIGINT
- Decimal(P > 18 & P ≤ 38, S)
  - LogicalType: Decimal128
  - Stored as: int128
  - Delegate LogicalType: LARGEINT

Начиная с версии v1.5.2, Selena поддерживает элементы Fast Decimal в ARRAY, MAP и STRUCT.

DECIMAL256

Selena вводит DECIMAL256 начиная с версии v2.0.0.

Для DECIMAL256 диапазон `P` составляет (38,76], а диапазон `S` — [0, P]. Значение `S` по умолчанию — 0.

DECIMAL256 расширяет верхний предел точности до 76 бит. Его числовая ёмкость в 10^{38} раз больше, чем у DECIMAL128, что снижает вероятность переполнения до незначительного уровня.

DECIMAL256 использует ту же стратегию для представления десятичных чисел, что и Fast DECIMAL. В дополнение к двум упомянутым выше типам десятичных чисел он поддерживает:

- Decimal(P > 38 & P ≤ 76, S)
  - LogicalType: Decimal256
  - Stored as: int256
  - Delegate LogicalType: INT_256

Фактический представимый диапазон DECIMAL256 — от `-57896044618658097711785492504343953926634992332820282019728792003956564819968`

до `57896044618658097711785492504343953926634992332820282019728792003956564819967`.

Операции приведения типов

DECIMAL256 поддерживает двунаправленное преобразование типов между всеми перечисленными ниже типами:

Исходный тип	Целевые типы
TYPE_DECIMAL256	BOOL
TYPE_DECIMAL256	TYPE_TINYINT, TYPE_SMALLINT, TYPE_INT, TYPE_BIGINT, TYPE_LARGEINT
TYPE_DECIMAL256	TYPE_FLOAT, TYPE_DOUBLE
TYPE_DECIMAL256	TYPE_VARCHAR

Агрегатные функции

В настоящее время DECIMAL256 поддерживает следующие агрегатные функции: `COUNT`, `COUNT DISTINCT`, `SUM`, `SUM DISTINCT`, `AVG`, `AVG DISTINCT`, `MAX`, `MIN` и `ABS`.

Ограничения

DECIMAL256 имеет следующие ограничения:

- В настоящее время DECIMAL256 не поддерживает автоматическое увеличение точности для десятичных операций.

  Например, `DECIMAL128 * DECIMAL128` не будет автоматически увеличен до DECIMAL256. Функции DECIMAL256 применяются только тогда, когда `DECIMAL256` явно указан в качестве операнда либо в операторе CREATE TABLE, либо с использованием CAST (`SELECT CAST(p38s10 as DECIMAL(70, 30))`).

- DECIMAL256 не поддерживает оконные функции.

- Агрегатные таблицы не поддерживают тип DECIMAL256.

Примеры

Пример Fast DECIMAL

Определение столбцов DECIMAL при создании таблицы.

```
CREATE TABLE decimalDemo (
    pk BIGINT(20) NOT NULL COMMENT "",
    account DECIMAL(20,10) COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(pk)
COMMENT "OLAP"
DISTRIBUTED BY HASH(pk);

INSERT INTO decimalDemo VALUES
(1,3.141592656),
(2,21.638378),
(3,4873.6293048479);

SELECT * FROM decimalDemo;
+------+-----------------+
| pk   | account         |
+------+-----------------+
|    1 |    3.1415926560 |
|    3 | 4873.6293048479 |
|    2 |   21.6383780000 |
+------+-----------------+
```

Пример DECIMAL256

Создание таблицы со столбцом типа DECIMAL256.

```
CREATE TABLE test_decimal256(
    p50s48 DECIMAL(50, 48) COMMENT ""
);
```

Вставка различных значений:

```
INSERT INTO test_decimal256(p50s48) SELECT 1.222222;
```

Этот оператор выполняется успешно, значения записываются корректно.

```
INSERT INTO test_decimal256(p50s48) SELECT
11111111111111111111111111111111111111111111.222222;
```

Этот оператор выполняется успешно, значения записываются как NULL. Поскольку целая часть и десятичная часть (которая имеет 44 цифры) с указанным scale (который равен `48`) вместе превышают максимальное целое число, которое может быть представлено 256 битами, FE напрямую преобразует значение в NULL.

```
INSERT INTO test_decimal256(p50s48) SELECT 333;
```

Этот оператор вернёт ошибку `ERROR 1064 (HY000): Insert has filtered data`. Поскольку целая часть и десятичная часть с указанным scale (который равен `48`) вместе превышают максимальное значение, представленное 50-разрядным целым числом, BE возвращает ошибку.

Примечания по использованию

Вы можете установить системную переменную `sql_mode` в `ERROR_IF_OVERFLOW`, чтобы система возвращала ошибку вместо NULL в случае арифметического переполнения.

Ключевые слова

decimal, decimalv3, fast decimal, decimal128, decimal256

DOUBLE

DOUBLE — это 8-байтовое число с плавающей точкой.

FLOAT

FLOAT

4-байтное число с плавающей точкой

INT

INT - это 4-байтное знаковое целое число. Диапазон значений: [-2147483648, 2147483647].

LARGEINT

LARGEINT - это 16-байтное знаковое целое число. Диапазон значений: [-2^127 + 1, 2^127 - 1].

SMALLINT

SMALLINT - это 2-байтное знаковое целое число. Диапазон значений: [-32768, 32767].

TINYINT

TINYINT - это 1-байтное знаковое целое число. Диапазон значений: [-128, 127].

1.2 String

BINARY/VARBINARY

BINARY(M)

VARBINARY(M)

Начиная с версии v1.5.2, Selena поддерживает тип данных BINARY/VARBINARY, который используется для хранения бинарных данных. Максимальная поддерживаемая длина такая же, как у VARCHAR [1, 1048576]. Единица измерения: байты. Если `M` не указан, по умолчанию используется 1048576. Бинарные типы данных содержат байтовые строки, тогда как символьные типы данных содержат символьные строки.

BINARY является псевдонимом VARBINARY. Использование такое же, как у VARBINARY.

Ограничения и примечания по использованию

- Столбцы VARBINARY поддерживаются в таблицах Duplicate Key, Primary Key и Unique Key. Они не поддерживаются в Aggregate-таблицах.
- Столбцы VARBINARY не могут использоваться в качестве ключей partition, ключей bucket или столбцов измерений в таблицах Duplicate Key, Primary Key и Unique Key. Они не могут использоваться в предложениях ORDER BY, GROUP BY и JOIN.
- BINARY(M)/VARBINARY(M) не дополняются справа в случае несогласованной длины.

Примеры

Создание столбца типа VARBINARY

При создании таблицы используйте ключевое слово `VARBINARY` для указания столбца `j` как столбца VARBINARY.

```
CREATE TABLE `test_binary` (
    `id` INT(11) NOT NULL COMMENT "",
    `j`  VARBINARY NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`id`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`id`)
PROPERTIES (
    "replication_num" = "3",
    "storage_format" = "DEFAULT"
);

mysql> DESC test_binary;
+-------+-----------+------+-------+---------+-------+
| Field | Type      | Null | Key   | Default | Extra |
+-------+-----------+------+-------+---------+-------+
| id    | int       | NO   | true  | NULL    |       |
| j     | varbinary | YES  | false | NULL    |       |
+-------+-----------+------+-------+---------+-------+
2 rows in set (0.01 sec)
```

Загрузка данных и хранение в типе BINARY

Selena поддерживает следующие способы загрузки данных и хранения их в типе BINARY.

- Способ 1: Используйте INSERT INTO для записи данных в константный столбец типа BINARY (например, столбец `j`), где константный столбец имеет префикс `x''`.

```
INSERT INTO test_binary (id, j) VALUES (1, x'abab');
INSERT INTO test_binary (id, j) VALUES (2, x'baba');
INSERT INTO test_binary (id, j) VALUES (3, x'010102');
INSERT INTO test_binary (id, j) VALUES (4, x'0000');
```

- Способ 2: Используйте функцию to_binary для преобразования данных VARCHAR в бинарные данные.

```
INSERT INTO test_binary select 5, to_binary('abab', 'hex');
INSERT INTO test_binary select 6, to_binary('abab', 'base64');
INSERT INTO test_binary select 7, to_binary('abab', 'utf8');
```

- Способ 3: Используйте Broker Load для загрузки файла Parquet или ORC и сохранения файла как данных BINARY. Подробнее см. Broker Load.
    - Для файлов Parquet конвертируйте `parquet::Type::type::BYTE_ARRAY` в `TYPE_VARBINARY` напрямую.
    - Для файлов ORC конвертируйте `orc::BINARY` в `TYPE_VARBINARY` напрямую.
- Способ 4: Используйте Stream Load для загрузки CSV-файла и сохранения файла как данных `BINARY`. Подробнее см. Загрузка данных CSV.
    - CSV-файл использует hex-формат для бинарных данных. Убедитесь, что входное бинарное значение является допустимым hex-значением.
    - Тип `BINARY` поддерживается только в CSV-файлах. JSON-файлы не поддерживают тип `BINARY`.

  Например, `t1` - это таблица со столбцом VARBINARY `b`.

```
CREATE TABLE `t1` (
`k` int(11) NOT NULL COMMENT "",
`v` int(11) NOT NULL COMMENT "",
`b` varbinary
) ENGINE = OLAP
DUPLICATE KEY(`k`)
PARTITION BY RANGE(`v`) (
PARTITION p1 VALUES [("-2147483648"), ("0")),
PARTITION p2 VALUES [("0"), ("10")),
PARTITION p3 VALUES [("10"), ("20")))
DISTRIBUTED BY HASH(`k`)
PROPERTIES ("replication_num" = "1");

-- csv-файл
-- cat temp_data
0,0,ab

-- Загрузка CSV-файла с помощью Stream Load.
curl --location-trusted -u <username>:<password> -T temp_data -XPUT -H
column_separator:, -H label:xx
http://172.17.0.1:8131/api/test_mv/t1/_stream_load

-- Запрос загруженных данных.
mysql> select * from t1;
+------+------+------------+
| k    | v    | xx         |
+------+------+------------+
|    0 |    0 | 0xAB       |
+------+------+------------+
1 rows in set (0.11 sec)
```

Запрос и обработка данных BINARY

Selena поддерживает запрос и обработку данных BINARY, а также использование функций и операторов BINARY. В этом примере используется таблица `test_binary`.

Примечание: Если вы добавите опцию `--binary-as-hex` при доступе к Selena из клиента MySQL, бинарные данные будут отображаться в hex-нотации.

```
mysql> select * from test_binary;
+------+------------+
| id   | j          |
+------+------------+
|    1 | 0xABAB     |
|    2 | 0xBABA     |
|    3 | 0x010102   |
|    4 | 0x0000     |
|    5 | 0xABAB     |
|    6 | 0xABAB     |
|    7 | 0x61626162 |
+------+------------+
7 rows in set (0.08 sec)
```

Пример 1: Просмотр бинарных данных с помощью функции hex.

```
mysql> select id, hex(j) from test_binary;
+------+----------+
| id   | hex(j)   |
+------+----------+
|    1 | ABAB     |
|    2 | BABA     |
|    3 | 010102   |
|    4 | 0000     |
|    5 | ABAB     |
|    6 | ABAB     |
|    7 | 61626162 |
+------+----------+
7 rows in set (0.02 sec)
```

Пример 2: Просмотр бинарных данных с помощью функции to_base64.

```
mysql> select id, to_base64(j) from test_binary;
+------+--------------+
| id   | to_base64(j) |
+------+--------------+
|    1 | q6s=         |
|    2 | uro=         |
|    3 | AQEC         |
|    4 | AAA=         |
|    5 | q6s=         |
|    6 | q6s=         |
|    7 | YWJhYg==     |
+------+--------------+
7 rows in set (0.01 sec)
```

Пример 3: Просмотр бинарных данных с помощью функции from_binary.

```
mysql> select id, from_binary(j, 'hex') from test_binary;
+------+-----------------------+
| id   | from_binary(j, 'hex') |
+------+-----------------------+
|    1 | ABAB                  |
|    2 | BABA                  |
|    3 | 010102                |
|    4 | 0000                  |
|    5 | ABAB                  |
|    6 | ABAB                  |
|    7 | 61626162              |
+------+-----------------------+
7 rows in set (0.01 sec)
```

CHAR

CHAR(M)

Строка фиксированной длины, `M` представляет длину строки фиксированной длины. Единица измерения: байты. Диапазон `M`: [1, 255].

STRING

STRING

Строка переменной длины. Максимальная длина составляет 65533 байта.

VARCHAR

VARCHAR(M)

Строка переменной длины. `M` указывает длину строки. Значение по умолчанию - `1`. Единица измерения: байты.

- В версиях до Selena 2.1 диапазон значений `M` составляет [1, 65533].
- [Preview] В Selena 2.1 и более поздних версиях диапазон значений `M` составляет [1, 1048576].

Примеры

Создайте таблицу и укажите тип столбца как VARCHAR.

```
CREATE TABLE varcharDemo (
    pk INT COMMENT "range [-2147483648, 2147483647]",
    pd_type VARCHAR(20) COMMENT "range char(m),m in (1-65533) "
) ENGINE=OLAP
DUPLICATE KEY(pk)
COMMENT "OLAP"
DISTRIBUTED BY HASH(pk)
```

1.3 Типы данных даты и времени

DATE

Тип DATE. Диапазон значений: ['0000-01-01', '9999-12-31'], формат по умолчанию: `YYYY-MM-DD`.

ПОДСКАЗКА

- SQL-функции даты и времени
- Типы данных даты и времени

Примеры

Пример 1: Указать столбец типа DATE при создании таблицы.

```
CREATE TABLE dateDemo (
    pk INT COMMENT "range [-2147483648, 2147483647]",
    make_time DATE NOT NULL COMMENT "YYYY-MM-DD"
) ENGINE=OLAP
DUPLICATE KEY(pk)
COMMENT "OLAP"
DISTRIBUTED BY HASH(pk)
```

Пример 2: Преобразовать значение DATETIME в значение DATE.

```
mysql> SELECT DATE('2003-12-31 01:02:03');
-> '2003-12-31'
```

Для получения дополнительной информации см. функцию date.

DATETIME

Тип даты и времени. Диапазон значений: ['0000-01-01 00:00:00', '9999-12-31 23:59:59'].

ПОДСКАЗКА

- SQL-функции даты и времени
- Типы данных даты и времени

Формат вывода: `YYYY-MM-DD HH:MM:SS`.

Начиная с версии v1.5.2, DATETIME поддерживает точность в миллисекундах и микросекундах. Формат вывода: `YYYY-MM-DD HH:MM:SS.ffffff`.

1.4 Semi-structured

ARRAY

ARRAY, как расширенный тип базы данных, поддерживается в различных системах баз данных, таких как PostgreSQL, ClickHouse и Snowflake. ARRAY широко используется в таких сценариях, как A/B-тесты, анализ пользовательских тегов и профилирование пользователей. Selena поддерживает вложенность многомерных массивов, срезы массивов, сравнение и фильтрацию.

Определение столбцов ARRAY

Вы можете определить столбец ARRAY при создании таблицы.

```
-- Определение одномерного массива.
ARRAY<type>

-- Определение вложенного массива.
ARRAY<ARRAY<type>>

-- Определение столбца массива как NOT NULL.
ARRAY<type> NOT NULL
```

`type` указывает типы данных элементов в массиве. Selena поддерживает следующие типы элементов: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, VARCHAR, CHAR, DATETIME, DATE, JSON, ARRAY (начиная с v1.5.2), MAP (начиная с v1.5.2) и STRUCT (начиная с v1.5.2).

Элементы в массиве по умолчанию допускают значение null, например, `[null, 1 ,2]`. Вы не можете указать элементы в массиве как NOT NULL. Однако вы можете указать столбец ARRAY как NOT NULL при создании таблицы, как в третьем примере в следующем фрагменте кода.

Примеры:

```
-- Определите c1 как одномерный массив с типом элемента INT.
create table t0(
  c0 INT,
  c1 ARRAY<INT>
)
duplicate key(c0)
distributed by hash(c0);

-- Определите c1 как вложенный массив с типом элемента VARCHAR.
create table t1(
  c0 INT,
  c1 ARRAY<ARRAY<VARCHAR(10)>>
)
duplicate key(c0)
distributed by hash(c0);

-- Определите c1 как столбец массива NOT NULL.
create table t2(
  c0 INT,
  c1 ARRAY<INT> NOT NULL
)
duplicate key(c0)
distributed by hash(c0);
```

Ограничения

При создании столбцов ARRAY в таблицах Selena применяются следующие ограничения:

- В версиях до v1.5.2 вы можете создавать столбцы ARRAY только в таблицах Duplicate Key. Начиная с v1.5.2 вы также можете создавать столбцы ARRAY в таблицах других типов (Primary Key, Unique Key, Aggregate). Обратите внимание, что в Aggregate-таблице вы можете создать столбец ARRAY только тогда, когда функция, используемая для агрегации данных в этом столбце, - replace() или replace_if_not_null(). Подробнее см. Aggregate-таблица.
- Столбцы ARRAY не могут использоваться в качестве ключевых столбцов.
- Столбцы ARRAY не могут использоваться в качестве ключей partition (включены в PARTITION BY) или ключей bucket (включены в DISTRIBUTED BY).
- Массив может иметь максимум 14 уровней вложенности.

Создание массивов в SQL

Массивы можно создавать в SQL с помощью квадратных скобок `[]`, где каждый элемент массива разделяется запятой (`,`).

```
mysql> select [1, 2, 3] as numbers;

+---------+
| numbers |
+---------+
| [1,2,3] |
+---------+

mysql> select ["apple", "orange", "pear"] as fruit;

+---------------------------+
| fruit                     |
+---------------------------+
| ["apple","orange","pear"] |
+---------------------------+

mysql> select [true, false] as booleans;

+----------+
| booleans |
+----------+
| [1,0]    |
+----------+
```

Selena автоматически определяет типы данных, если массив состоит из элементов нескольких типов:

```
mysql> select [1, 1.2] as floats;
+---------+
| floats  |
+---------+
| [1.0,1.2] |
+---------+

mysql> select [12, "100"];

+--------------+
| [12,'100']   |
+--------------+
| ["12","100"] |
+--------------+
```

Вы можете использовать угловые скобки (`<>`) для указания объявленного типа массива.

```
mysql> select ARRAY<float>[1, 2];

+-----------------------+
| ARRAY<float>[1.0,2.0] |
+-----------------------+
| [1,2]                 |
+-----------------------+

mysql> select ARRAY<INT>["12", "100"];

+------------------------+
| ARRAY<int(11)>[12,100] |
+------------------------+
| [12,100]               |
+------------------------+
```

NULL может быть включен в элемент.

```
mysql> select [1, NULL];

+----------+
| [1,NULL] |
+----------+
| [1,null] |
+----------+
```

Для пустого массива вы можете использовать угловые скобки для указания объявленного типа, или вы можете написать [] напрямую, чтобы Selena определила тип на основе контекста. Если Selena не может определить тип, она сообщит об ошибке.

```
mysql> select [];

+------+
| []   |
+------+
| []   |
+------+

mysql> select ARRAY<VARCHAR(10)>[];

+-----------------------------------+
| ARRAY<unknown type: NULL_TYPE>[] |
+-----------------------------------+
| []                                |
+-----------------------------------+

mysql> select array_append([], 10);

+----------------------+
| array_append([], 10) |
+----------------------+
| [10]                 |
+----------------------+
```

Загрузка данных Array

Selena поддерживает загрузку данных Array тремя способами:

- INSERT INTO подходит для загрузки небольших объемов данных для тестирования.

- Broker Load подходит для загрузки файлов ORC или Parquet с большими объемами данных.
- Stream Load и Routine Load подходят для загрузки CSV-файлов с большими объемами данных.

Использование INSERT INTO для загрузки массивов

Вы можете использовать INSERT INTO для загрузки небольших объемов данных столбец за столбцом или выполнить ETL над данными перед их загрузкой.

```
create table t0(
c0 INT,
c1 ARRAY<INT>
)
duplicate key(c0)
distributed by hash(c0);

INSERT INTO t0 VALUES(1, [1,2,3]);
```

Использование Broker Load для загрузки массивов из файлов ORC или Parquet

Тип массива в Selena соответствует структуре list в файлах ORC и Parquet, что избавляет от необходимости указывать разные типы данных в Selena. Подробнее о загрузке данных см. Broker load.

Использование Stream Load или Routine Load для загрузки массивов в формате CSV

Массивы в CSV-файлах по умолчанию разделяются запятой. Вы можете использовать Stream Load или Routine Load для загрузки текстовых файлов CSV или данных CSV в Kafka.

Запрос данных ARRAY

Вы можете получить доступ к элементам массива с помощью `[]` и индексов, начиная с `1`.

```
mysql> select [1,2,3][1];

+------------+
| [1,2,3][1] |
+------------+
|          1 |
+------------+
1 row in set (0.00 sec)
```

Если индекс равен 0 или отрицательному числу, **ошибка не выводится и возвращается NULL**.

```
mysql> select [1,2,3][0];

+------------+
| [1,2,3][0] |
+------------+
|       NULL |
+------------+
1 row in set (0.01 sec)
```

Если индекс превышает длину массива (количество элементов в массиве), **будет возвращен NULL**.

```
mysql> select [1,2,3][4];

+------------+
| [1,2,3][4] |
+------------+
|       NULL |
+------------+
1 row in set (0.01 sec)
```

Для многомерных массивов к элементам можно обращаться **рекурсивно**.

```
mysql(ARRAY)> select [[1,2],[3,4]][2];

+------------------+
| [[1,2],[3,4]][2] |
+------------------+
| [3,4]            |
+------------------+
1 row in set (0.00 sec)

mysql> select [[1,2],[3,4]][2][1];

+---------------------+
| [[1,2],[3,4]][2][1] |
+---------------------+
|                   3 |
+---------------------+
1 row in set (0.01 sec)
```

JSON

Начиная с версии 2.2.0, Selena поддерживает JSON. В этой статье представлены основные концепции JSON, а также описано, как Selena создает столбцы типа JSON, загружает и запрашивает данные JSON, а также конструирует и обрабатывает данные JSON с помощью JSON-функций и операторов.

Что такое JSON

JSON - это легковесный формат обмена данными. Данные типа JSON являются полуструктурированными и поддерживают древовидную структуру. Данные JSON иерархичны, гибки, легко читаются и обрабатываются, и широко используются в сценариях хранения и анализа данных. JSON поддерживает такие типы данных, как NUMBER, STRING, BOOLEAN, ARRAY, OBJECT и значения NULL.

Для получения дополнительной информации о JSON обратитесь к официальному сайту JSON. Для синтаксиса ввода и вывода данных JSON обратитесь к спецификации JSON RFC 7159.

Selena поддерживает хранение и эффективный запрос и анализ данных JSON. Selena использует кодирование в бинарном формате для хранения данных JSON вместо прямого хранения входного текста, что снижает затраты на парсинг при вычислении данных и запросах, тем самым повышая эффективность запросов.

Использование данных JSON

Создание столбцов типа JSON

При создании таблицы укажите столбец `j` как тип JSON с помощью ключевого слова `JSON`.

```
CREATE TABLE `tj` (
    `id` INT(11) NOT NULL COMMENT "",
    `j`  JSON NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`id`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`id`)
PROPERTIES (
    "replication_num" = "3",
    "storage_format" = "DEFAULT"
);
```

Загрузка данных и хранение в типе JSON

Selena поддерживает следующие методы загрузки данных и их хранения в типе JSON.

- Способ 1: Используйте `INSERT INTO` для записи данных в столбец типа JSON (например, столбец `j`).

```
INSERT INTO tj (id, j) VALUES (1, parse_json('{"a": 1, "b": true}'));
INSERT INTO tj (id, j) VALUES (2, parse_json('{"a": 2, "b": false}'));
INSERT INTO tj (id, j) VALUES (3, parse_json('{"a": 3, "b": true}'));
INSERT INTO tj (id, j) VALUES (4, json_object('a', 4, 'b', false));
```

> Функция PARSE_JSON может конструировать данные типа JSON на основе данных строкового типа. Функция JSON_OBJECT может конструировать данные типа JSON-объект, позволяя преобразовывать существующие таблицы в тип JSON. Подробнее см. PARSE_JSON и JSON_OBJECT.

- Способ 2: Используйте Stream Load для импорта JSON-файлов и хранения их в типе JSON. О методах импорта см. Импорт данных JSON.
    - Чтобы импортировать и сохранить JSON-объект в корневом узле JSON-файла как тип JSON, установите `jsonpaths` в `$`.

- Чтобы импортировать и сохранить значение JSON-объекта в JSON-файле как тип JSON, установите `jsonpaths` в `$.a` (где `a` представляет ключ). Подробнее о JSON path-выражениях см. JSON path.
- Способ 3: Используйте Broker Load для импорта Parquet-файлов и хранения их в типе JSON. О методах импорта см. Broker Load.

Преобразование типов данных поддерживается при импорте следующим образом:

Тип данных в Parquet-файле	Преобразованный тип данных JSON
Целочисленные типы (INT8, INT16, INT32, INT64, UINT8, UINT16, UINT32, UINT64)	JSON Number
Типы с плавающей точкой (FLOAT, DOUBLE)	JSON Number
BOOLEAN	JSON Boolean
STRING	JSON String
MAP	JSON Object
STRUCT	JSON Object
LIST	JSON Array
UNION, TIMESTAMP и другие типы	Не поддерживается

- Способ 4: Используйте Routine Load для непрерывного потребления данных в формате JSON из Kafka и импорта их в Selena.

Запрос и обработка данных типа JSON

Selena поддерживает запрос и обработку данных типа JSON, а также использование JSON-функций и операторов.

В этом примере используется таблица `tj` для иллюстрации.

```
mysql> select * from tj;
+------+----------------------+
| id   |          j           |
+------+----------------------+
| 1    | {"a": 1, "b": true}  |
| 2    | {"a": 2, "b": false} |
| 3    | {"a": 3, "b": true}  |
| 4    | {"a": 4, "b": false} |
+------+----------------------+
```

Пример 1: Фильтрация данных в столбце типа JSON, удовлетворяющих условию `id=1`.

```
mysql> select * from tj where id = 1;
+------+----------------------+
| id   |           j          |
+------+----------------------+
| 1    | {"a": 1, "b": true} |
+------+----------------------+
```

Пример 2: Фильтрация данных в таблице на основе столбца типа JSON.

> В следующем примере `j->'a'` возвращает данные типа JSON. Вы можете сравнить с первым примером, который выполняет неявное преобразование данных; или использовать функцию CAST для преобразования данных типа JSON в INT для сравнения.

```
mysql> select * from tj where j->'a' = 1;
+------+----------------------+
| id   | j                    |
+------+----------------------+
|    1 | {"a": 1, "b": true} |
+------+----------------------+

mysql> select * from tj where cast(j->'a' as INT) = 1;
+------+----------------------+
|   id |          j           |
+------+----------------------+
|   1  | {"a": 1, "b": true} |
+------+----------------------+
```

Пример 3: Фильтрация данных в таблице на основе столбца типа JSON (вы можете использовать функцию CAST для преобразования столбца типа JSON в тип BOOLEAN).

```
mysql> select * from tj where cast(j->'b' as boolean);
+------+----------------------+
|  id  |           j          |
+------+----------------------+
| 1    | {"a": 1, "b": true} |
| 3    | {"a": 3, "b": true} |
+------+----------------------+
```

Пример 4: Фильтрация данных в столбце типа JSON, удовлетворяющих условию, и выполнение числовых операций.

```
mysql> select cast(j->'a' as int) from tj where cast(j->'b' as boolean);
+-----------------------+
|  CAST(j->'a' AS INT)  |
+-----------------------+
|          3            |
|          1            |
+-----------------------+

mysql> select sum(cast(j->'a' as int)) from tj where cast(j->'b' as
boolean);
+----------------------------+
| sum(CAST(j->'a' AS INT))  |
+----------------------------+
|              4             |
+----------------------------+
```

Пример 5: Сортировка на основе столбца типа JSON.

```
mysql> select * from tj
       where j->'a' <= 3
       order by cast(j->'a' as int);
+------+----------------------+
| id   | j                    |
+------+----------------------+
|    1 | {"a": 1, "b": true}  |
|    2 | {"a": 2, "b": false} |
|    3 | {"a": 3, "b": true}  |
|    4 | {"a": 4, "b": false} |
+------+----------------------+
```

JSON-функции и операторы

JSON-функции и операторы можно использовать для конструирования и обработки данных JSON. Подробную информацию см. в JSON-функции и операторы.

JSON-массивы

JSON может содержать вложенные данные, такие как Objects, Arrays или другие типы данных JSON, вложенные в Array. Selena предоставляет богатый набор функций и

операторов для обработки этих сложных вложенных структур данных JSON. Далее будет рассмотрено, как обрабатывать данные JSON, содержащие массивы.

Предположим, что в таблице `events` есть JSON-поле `event_data` со следующим содержимым:

```
{
  "user": "Alice",
  "actions": [
    {"type": "click", "timestamp": "2024-03-17T10:00:00Z", "quantity": 1},
    {"type": "view", "timestamp": "2024-03-17T10:05:00Z", "quantity": 2},
    {"type": "purchase", "timestamp": "2024-03-17T10:10:00Z", "quantity":
3}
  ]
}
```

Следующие примеры демонстрируют несколько распространенных сценариев анализа JSON-массивов:

1. Извлечение элементов массива: Извлечение определенных полей, таких как type, timestamp и т.д., из массива actions и выполнение операций проекции.
2. Разворачивание массива: Использование функции `json_each` для разворачивания вложенного JSON-массива в многострочную, многостолбцовую табличную структуру для последующего анализа.
3. Вычисления с массивами: Использование Array-функций для фильтрации, преобразования и агрегации элементов массива, например, подсчет количества операций определенного типа.

1. Извлечение элементов из JSON-массива

Чтобы извлечь вложенный элемент из JSON-массива, вы можете использовать следующий синтаксис:

- Возвращаемый тип по-прежнему JSON-массив, и вы можете использовать выражение CAST для преобразования типа.

```
MySQL > SELECT json_query(event_data, '$.actions[*].type') as json_array
FROM events;
+-------------------------------+
| json_array                    |
+-------------------------------+
| ["click", "view", "purchase"] |
+-------------------------------+

MySQL > SELECT cast(json_query(event_data, '$.actions[*].type') as
array<string>) array_string FROM events;
+-----------------------------+
| array_string                |
+-----------------------------+
| ["click","view","purchase"] |
+-----------------------------+
```

2. Разворачивание с помощью json_each

Selena предоставляет функцию `json_each` для разворачивания JSON-массивов, преобразуя их в несколько строк данных. Например:

```
MySQL > select value from events, json_each(event_data->'actions');
+-------------------------------------------------------------------------
+
| value
|
+-------------------------------------------------------------------------
+
| {"quantity": 1, "timestamp": "2024-03-17T10:00:00Z", "type": "click"}
|
| {"quantity": 2, "timestamp": "2024-03-17T10:05:00Z", "type": "view"}
|
| {"quantity": 3, "timestamp": "2024-03-17T10:10:00Z", "type": "purchase"}
|
+-------------------------------------------------------------------------
+
```

Чтобы извлечь поля type и timestamp отдельно:

```
MySQL > select value->'timestamp', value->'type' from events,
json_each(event_data->'actions');
+------------------------+---------------+
| value->'timestamp'     | value->'type' |
+------------------------+---------------+
| "2024-03-17T10:00:00Z" | "click"       |
| "2024-03-17T10:05:00Z" | "view"        |
| "2024-03-17T10:10:00Z" | "purchase"    |
+------------------------+---------------+
```

После этого данные JSON-массива становятся знакомой реляционной моделью, что позволяет использовать обычные функции для анализа.

3. Использование Array-функций для фильтрации и вычислений

Selena также поддерживает функции, связанные с ARRAY, которые можно использовать совместно с JSON-функциями для более эффективных запросов. Комбинируя эти функции, вы можете фильтровать, преобразовывать и агрегировать данные JSON-массивов. Следующий пример демонстрирует, как использовать эти функции:

```
MySQL >
WITH step1 AS (
 SELECT cast(event_data->'actions' as ARRAY<JSON>) as docs
   FROM events
)
SELECT array_filter(doc -> get_json_string(doc, 'type') = 'click', docs) as
clicks
FROM step1
+-----------------------------------------------------------------------------
-+
| clicks
|
+-----------------------------------------------------------------------------
-+
| ['{"quantity": 1, "timestamp": "2024-03-17T10:00:00Z", "type": "click"}']
|
+-----------------------------------------------------------------------------
-+
```

Кроме того, вы можете комбинировать другие Array-функции для выполнения агрегационных вычислений над элементами массива:

```
MySQL >
WITH step1 AS (
 SELECT cast(event_data->'actions' as ARRAY<JSON>) as docs
   FROM events
), step2 AS (
    SELECT array_filter(doc -> get_json_string(doc, 'type') = 'click',
docs) as clicks
    FROM step1
)
SELECT array_sum(
            array_map(doc -> get_json_double(doc, 'quantity'), clicks)
            ) as click_amount
FROM step2
+--------------+
| click_amount |
+--------------+
| 1.0          |
+--------------+
```

Ограничения и особенности

- Максимальная поддерживаемая длина данных типа JSON в настоящее время составляет 16 МБ.
- Предложения ORDER BY, GROUP BY и JOIN не поддерживают ссылки на столбцы типа JSON. Если вам нужно на них ссылаться, вы можете заранее использовать функцию CAST для преобразования столбцов типа JSON в другие SQL-типы. О конкретных методах преобразования см. Преобразование типов JSON.
- Столбцы типа JSON могут существовать в таблицах Duplicate Key, Primary Key и Unique Key, но не в Aggregate-таблицах.
- Столбцы типа JSON не поддерживаются в качестве ключей partition, ключей bucket или столбцов измерений в таблицах Duplicate Key, Primary Key и Unique Key, и не могут использоваться в предложениях JOIN, GROUP BY или ORDER BY.
- Selena поддерживает использование операторов `<`, `<=`, `>`, `>=`, `=`, `!=` для запроса данных JSON, но не поддерживает оператор IN.

MAP

MAP - это сложный тип данных, который хранит набор пар ключ-значение, например, `{a:1, b:2, c:3}`. Ключи в map должны быть уникальными. Вложенный map может содержать до 14 уровней вложенности.

Тип данных MAP поддерживается начиная с версии v1.5.2. В v1.5.2 вы можете определять столбцы MAP при создании таблицы Selena, загружать данные MAP в эту таблицу и запрашивать данные MAP.

Начиная с v1.5.2, Selena поддерживает запрос сложных типов данных MAP и STRUCT из озер данных. Вы можете использовать внешние каталоги, предоставляемые Selena, для запроса данных MAP и STRUCT из Apache Hive, Apache Hudi и Apache Iceberg. Вы можете запрашивать данные только из файлов ORC и Parquet. Подробнее об использовании внешних каталогов для запроса внешних источников данных см. Обзор каталогов и темы, связанные с требуемым типом каталога.

Синтаксис

```
MAP<key_type,value_type>
```

- `key_type`: тип данных ключа. Ключ должен быть примитивного типа, поддерживаемого Selena, такого как числовой, строковый или дата. Он не может быть типа HLL, JSON, ARRAY, MAP, BITMAP или STRUCT.
- `value_type`: тип данных значения. Значение может быть любого поддерживаемого типа.

Ключи и значения **по умолчанию допускают null**.

Определение столбца MAP в Selena

Вы можете определить столбец MAP при создании таблицы и загружать данные MAP в этот столбец.

```sql
-- Определение одномерного map.
CREATE TABLE t0(
  c0 INT,
  c1 MAP<INT,INT>
)
DUPLICATE KEY(c0);

-- Определение вложенного map.
CREATE TABLE t1(
  c0 INT,
  c1 MAP<DATE, MAP<VARCHAR(10), INT>>
)
DUPLICATE KEY(c0);

-- Определение NOT NULL map.
CREATE TABLE t2(
  c0 INT,
  c1 MAP<INT,DATETIME> NOT NULL
)
DUPLICATE KEY(c0);
```

Столбцы с типом MAP имеют следующие ограничения:

- Не могут использоваться в качестве ключевых столбцов в таблице. Они могут использоваться только как столбцы значений.
- Не могут использоваться в качестве столбцов ключа partition (столбцы после PARTITION BY) в таблице.
- Не могут использоваться в качестве столбцов bucket (столбцы после DISTRIBUTED BY) в таблице.

Создание map в SQL

Map можно создать в SQL, используя два следующих синтаксиса:

- `map{key_expr:value_expr, ...}`: Элементы map разделяются запятой (`,`), а ключи и значения разделяются двоеточием (`:`), например, `map{a:1, b:2, c:3}`.

- `map(key_expr, value_expr ...)`: Выражения ключей и значений должны быть в парах, например, `map(a,1,b,2,c,3)`.

Selena может определить типы данных ключей и значений из всех входных ключей и значений.

```
select map{1:1, 2:2, 3:3} as numbers;
select map(1,1,2,2,3,3) as numbers; -- Возвращает {1:1,2:2,3:3}.
select map{1:"apple", 2:"orange", 3:"pear"} as fruit;
select map(1, "apple", 2, "orange", 3, "pear") as fruit; -- Возвращает
{1:"apple",2:"orange",3:"pear"}.
select map{true:map{3.13:"abc"}, false:map{}} as nest;
select map(true, map(3.13, "abc"), false, map{}) as nest; -- Возвращает {1:
{3.13:"abc"},0:{}}.
```

Если ключи или значения имеют разные типы, Selena автоматически определяет соответствующий тип (супертип).

```
select map{1:2.2, 1.2:21} as floats_floats; -- Возвращает
{1.0:2.2,1.2:21.0}.
select map{12:"a", "100":1, NULL:NULL} as string_string; -- Возвращает
{"12":"a","100":"1",null:null}.
```

Вы также можете определить тип данных с помощью `<>` при создании map. Входные ключи или значения должны иметь возможность приведения к указанным типам.

```
select map<FLOAT,INT>{1:2}; -- Возвращает {1:2}.
select map<INT,INT>{"12": "100"}; -- Возвращает {12:100}.
```

Ключи и значения допускают null.

```
select map{1:NULL};
```

Создание пустых map.

```
select map{} as empty_map;
select map() as empty_map; -- Возвращает {}.
```

Загрузка данных MAP в Selena

Вы можете загрузить данные map в Selena двумя способами: INSERT INTO и загрузка ORC/Parquet.

Обратите внимание, что Selena удаляет дублирующиеся ключи каждого map при загрузке данных MAP.

INSERT INTO

```
CREATE TABLE t0(
  c0 INT,
  c1 MAP<INT,INT>
)
DUPLICATE KEY(c0);

INSERT INTO t0 VALUES(1, map{1:2,3:NULL});
```

Загрузка данных MAP из файлов ORC и Parquet

Тип данных MAP в Selena соответствует структуре map в формате ORC или Parquet. Дополнительная спецификация не требуется. Вы можете загрузить данные MAP из файлов ORC или Parquet, следуя инструкциям в загрузка ORC/Parquet.

Доступ к данным MAP

Пример 1: Запрос столбца MAP `c1` из таблицы `t0`.

```
mysql> select c1 from t0;
+--------------+
| c1           |
+--------------+
| {1:2,3:null} |
+--------------+
```

Пример 2: Используйте оператор `[ ]` для получения значений из map по ключу или используйте функцию `element_at(any_map, any_key)`.

Следующий пример запрашивает значение, соответствующее ключу `1`.

```
mysql> select map{1:2,3:NULL}[1];
+-----------------------+
| map(1, 2, 3, NULL)[1] |
+-----------------------+
|                     2 |
+-----------------------+

mysql> select element_at(map{1:2,3:NULL},1);
+--------------------+
| map{1:2,3:NULL}[1] |
+--------------------+
|                  2 |
+--------------------+
```

Если ключ не существует в map, возвращается `NULL`.

Следующий пример запрашивает значение, соответствующее ключу 2, который не существует.

```
mysql> select map{1:2,3:NULL}[2];
+-----------------------+
| map(1, 2, 3, NULL)[2] |
+-----------------------+
|                  NULL |
+-----------------------+
```

Пример 3: Запрос многомерных map **рекурсивно**.

Следующий пример сначала запрашивает значение, соответствующее ключу `1`, которое равно `map{2:1}`, а затем рекурсивно запрашивает значение, соответствующее ключу `2` в `map{2:1}`.

```
mysql> select map{1:map{2:1},3:NULL}[1][2];

+----------------------------------+
| map(1, map(2, 1), 3, NULL)[1][2] |
+----------------------------------+
|                                1 |
+----------------------------------+
```

Ссылки

- Map-функции
- element_at

STRUCT

STRUCT широко используется для выражения сложных типов данных. Он представляет собой коллекцию элементов (также называемых полями) с различными типами данных, например, `<a INT, b STRING>`.

Имена полей в struct должны быть уникальными. Поля могут быть примитивных типов данных (таких как числовой, строковый или дата) или сложных типов данных (таких как ARRAY или MAP).

Поле внутри struct также может быть другим STRUCT, ARRAY или MAP, что позволяет создавать вложенные структуры данных, например, `STRUCT<a INT, b STRUCT<c INT, d INT>, c MAP<INT, INT>, d ARRAY<INT>>`.

Тип данных STRUCT поддерживается начиная с версии v1.5.2. В v1.5.2 вы можете определять столбцы STRUCT при создании таблицы Selena, загружать данные STRUCT в эту таблицу и запрашивать данные MAP.

Начиная с v1.5.2, Selena поддерживает запрос сложных типов данных MAP и STRUCT из озер данных. Вы можете использовать внешние каталоги, предоставляемые Selena, для запроса данных MAP и STRUCT из Apache Hive, Apache Hudi и Apache Iceberg. Вы можете запрашивать данные только из файлов ORC и Parquet. Подробнее об использовании внешних каталогов для запроса внешних источников данных см. Обзор каталогов и темы, связанные с требуемым типом каталога.

Синтаксис

```
STRUCT<name, type>
```

- `name`: имя поля, такое же, как имя столбца, определенное в операторе CREATE TABLE.
- `type`: тип поля. Может быть любым поддерживаемым типом.

Определение столбца STRUCT в Selena

Вы можете определить столбец STRUCT при создании таблицы и загружать данные STRUCT в этот столбец.

```
-- Определение одномерного struct.
CREATE TABLE t0(
  c0 INT,
  c1 STRUCT<a INT, b INT>
)
DUPLICATE KEY(c0);

-- Определение сложного struct.
CREATE TABLE t1(
  c0 INT,
  c1 STRUCT<a INT, b STRUCT<c INT, d INT>, c MAP<INT, INT>, d ARRAY<INT>>
)
DUPLICATE KEY(c0);

-- Определение NOT NULL struct.
CREATE TABLE t2(
  c0 INT,
  c1 STRUCT<a INT, b INT> NOT NULL
)
DUPLICATE KEY(c0);
```

Столбцы с типом STRUCT имеют следующие ограничения:

- Не могут использоваться в качестве ключевых столбцов в таблице. Они могут использоваться только как столбцы значений.
- Не могут использоваться в качестве столбцов ключа partition (после PARTITION BY) в таблице.
- Не могут использоваться в качестве столбцов bucket (после DISTRIBUTED BY) в таблице.
- Поддерживают только функцию replace() при использовании в качестве столбца значений в Aggregate-таблице.

Создание struct в SQL

STRUCT можно создать в SQL с помощью следующих функций: row, struct и named_struct. struct() является псевдонимом row().

- `row` и `struct` поддерживают безымянный struct. Вам не нужно указывать имена полей. Selena автоматически генерирует имена столбцов, такие как `col1`, `col2`...
- `named_struct` поддерживает именованный struct. Выражения имен и значений должны быть в парах.

Selena автоматически определяет тип struct на основе входных значений.

```
select row(1, 2, 3, 4) as numbers; -- Возвращает
{"col1":1,"col2":2,"col3":3,"col4":4}.
select row(1, 2, null, 4) as numbers; -- Возвращает
{"col1":1,"col2":2,"col3":null,"col4":4}.
select row(null) as nulls; -- Возвращает {"col1":null}.
select struct(1, 2, 3, 4) as numbers; -- Возвращает
{"col1":1,"col2":2,"col3":3,"col4":4}.
select named_struct('a', 1, 'b', 2, 'c', 3, 'd', 4) as numbers; --
Возвращает {"a":1,"b":2,"c":3,"d":4}.
```

Загрузка данных STRUCT

Вы можете загрузить данные STRUCT в Selena двумя способами: INSERT INTO и загрузка ORC/Parquet.

Обратите внимание, что Selena автоматически приводит тип данных к соответствующему типу STRUCT. При загрузке данных в столбец именованного struct убедитесь, что имена полей внутри struct находятся в том же порядке, что и при создании таблицы.

INSERT INTO

```
CREATE TABLE t0(
  c0 INT,
  c1 STRUCT<a INT, b INT>
)
DUPLICATE KEY(c0);

INSERT INTO t0 VALUES(1, row(1, 1));

SELECT * FROM t0;
+------+---------------+
| c0   | c1            |
+------+---------------+
|    1 | {"a":1,"b":1} |
+------+---------------+
```

Загрузка данных STRUCT из файлов ORC/Parquet

Тип данных STRUCT в Selena соответствует структуре вложенных столбцов в формате ORC или Parquet. Дополнительная спецификация не требуется. Вы можете загрузить данные STRUCT из файлов ORC или Parquet, следуя инструкциям в загрузка ORC/Parquet.

Доступ к полям STRUCT

Для запроса подполя struct вы можете использовать оператор точка (`.`) для запроса значения по имени поля, или использовать `[]` для вызова значения по его индексу.

```
mysql> select named_struct('a', 1, 'b', 2, 'c', 3, 'd', 4).a;
+-------------------------------------------------+
| named_struct('a', 1, 'b', 2, 'c', 3, 'd', 4).a |
+-------------------------------------------------+
| 1                                               |
+-------------------------------------------------+

mysql> select row(1, 2, 3, 4).col1;
+------------------------+
| row(1, 2, 3, 4).col1  |
+------------------------+
| 1                      |
+------------------------+

mysql> select row(2, 4, 6, 8)[2];
+--------------------+
| row(2, 4, 6, 8)[2] |
+--------------------+
|                  4 |
+--------------------+

mysql> select row(map{'a':1}, 2, 3, 4)[1];
+------------------------------+
| row(map{'a':1}, 2, 3, 4)[1] |
+------------------------------+
| {"a":1}                      |
+------------------------------+
```

1.5 Other

BITMAP

BITMAP часто используется для ускорения подсчета уникальных значений (count distinct). Он похож на HyperLogLog (HLL), но обеспечивает более точный подсчет уникальных значений. BITMAP потребляет больше памяти и дисковых ресурсов. Он поддерживает агрегацию только данных типа INT. Если вы хотите применить bitmap к строковым данным, необходимо сопоставить данные с помощью словаря низкой кардинальности.

В этой теме приводится простой пример создания столбца BITMAP и использования bitmap-функций для агрегации данных этого столбца. Подробные определения функций или дополнительные Bitmap-функции см. в разделе "Bitmap-функции".

Создание таблицы

- Создайте Aggregate-таблицу, в которой тип данных столбца `user_id` - BITMAP, и функция bitmap_union() используется для агрегации данных.

```
CREATE TABLE `pv_bitmap` (
`dt` int(11) NULL COMMENT "",
`page` varchar(10) NULL COMMENT "",
`user_id` bitmap BITMAP_UNION NULL COMMENT ""
) ENGINE=OLAP
AGGREGATE KEY(`dt`, `page`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`dt`);
```

- Создайте Primary Key-таблицу, в которой тип данных столбца `userid` - BITMAP.

```
CREATE TABLE primary_bitmap (
`tagname` varchar(65533) NOT NULL COMMENT "Tag name",
`tagvalue` varchar(65533) NOT NULL COMMENT "Tag value",
`userid` bitmap NOT NULL COMMENT "User ID")
ENGINE=OLAP
PRIMARY KEY(`tagname`, `tagvalue`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`tagname`);
```

Перед вставкой данных в столбцы BITMAP необходимо сначала использовать функцию to_bitmap() для преобразования данных.

Подробнее об использовании BITMAP, например, о загрузке данных BITMAP в таблицу, см. bitmap.

HLL (HyperLogLog)

HLL используется для приблизительного подсчета уникальных значений.

HLL позволяет разрабатывать программы на основе алгоритма HyperLogLog. Он используется для хранения промежуточных результатов вычислений HyperLogLog. Его можно использовать только в качестве типа столбца значений таблицы. HLL уменьшает объем данных через агрегацию для ускорения процесса запроса. В оценочных результатах может быть отклонение в 1%.

Столбец HLL генерируется на основе импортированных данных или данных из других столбцов. При импорте данных функция hll_hash указывает, какой столбец будет использоваться для генерации столбца HLL. HLL часто используется для замены COUNT DISTINCT и быстрого вычисления уникальных просмотров (UV) с rollup.

Объем хранения, используемый HLL, определяется уникальными значениями в хеш-значении. Объем хранения варьируется в зависимости от трех условий:

- HLL пуст. В HLL не вставлено ни одного значения, и стоимость хранения минимальна - 80 байт.
- Количество уникальных хеш-значений в HLL меньше или равно 160. Максимальная стоимость хранения составляет 1360 байт (80 + 160 * 8 = 1360).
- Количество уникальных хеш-значений в HLL больше 160. Стоимость хранения фиксирована и составляет 16 464 байта (80 + 16 * 1024 = 16464).

В реальных бизнес-сценариях объем данных и распределение данных влияют на использование памяти при запросах и точность приблизительного результата. Необходимо учитывать эти два фактора:

- Объем данных: HLL возвращает приблизительное значение. Больший объем данных дает более точный результат. Меньший объем данных приводит к большему отклонению.
- Распределение данных: В случае большого объема данных и столбца измерения с высокой кардинальностью для GROUP BY вычисление данных будет использовать больше памяти. HLL не рекомендуется в этой ситуации. Он рекомендуется при выполнении count distinct без группировки или GROUP BY по столбцам измерений с низкой кардинальностью.
- Гранулярность запроса: Если вы запрашиваете данные с большой гранулярностью запроса, рекомендуется использовать Aggregate-таблицу или материализованное представление для предварительной агрегации данных для уменьшения объема данных.

Связанные функции

- HLL_UNION_AGG(hll): Это агрегатная функция, используемая для оценки кардинальности всех данных, удовлетворяющих условиям. Также может использоваться для анализа функций. Она поддерживает только окно по умолчанию и не поддерживает предложение window.

- HLL_RAW_AGG(hll): Это агрегатная функция, используемая для агрегации полей типа hll и возвращающая тип hll.
- HLL_CARDINALITY(hll): Эта функция используется для оценки кардинальности одного столбца hll.
- HLL_HASH(column_name): Генерирует тип столбца HLL и используется для вставки или импорта. См. инструкции по использованию импорта.
- HLL_EMPTY: Генерирует пустой столбец HLL и используется для заполнения значений по умолчанию при вставке или импорте. См. инструкции по использованию импорта.

Примеры

1. Создайте таблицу со столбцами HLL `set1` и `set2`.

```
create table test(
dt date,
id int,
name char(10),
province char(10),
os char(1),
set1 hll hll_union,
set2 hll hll_union)
distributed by hash(id);
```

2. Загрузите данные с помощью Stream Load.

```
a. Используйте столбцы таблицы для генерации столбца HLL.
curl --location-trusted -uname:password -T data -H "label:load_1" \
    -H "columns:dt, id, name, province, os, set1=hll_hash(id),
set2=hll_hash(name)"
http://host/api/test_db/test/_stream_load

b. Используйте столбцы данных для генерации столбца HLL.
curl --location-trusted -uname:password -T data -H "label:load_1" \
    -H "columns:dt, id, name, province, sex, cuid, os,
set1=hll_hash(cuid), set2=hll_hash(os)"
http://host/api/test_db/test/_stream_load
```

3. Агрегируйте данные одним из трех способов: (Без агрегации прямой запрос к базовой таблице может быть таким же медленным, как использование approx_count_distinct)

```
-- a. Создайте rollup для агрегации столбца HLL.
alter table test add rollup test_rollup(dt, set1);

-- b. Создайте другую таблицу для вычисления uv и вставьте в нее данные

create table test_uv(
dt date,
id int
uv_set hll hll_union)
distributed by hash(id);

insert into test_uv select dt, id, set1 from test;

-- c. Создайте другую таблицу для вычисления UV. Вставьте данные и
сгенерируйте столбец HLL, тестируя другие столбцы через hll_hash.

create table test_uv(
dt date,
id int,
id_set hll hll_union)
distributed by hash(id);

insert into test_uv select dt, id, hll_hash(id) from test;
```

4. Запросите данные. Столбцы HLL не поддерживают прямой запрос к их исходным значениям. Их можно запрашивать с помощью соответствующих функций.

```
a. Вычислите общий UV.
select HLL_UNION_AGG(uv_set) from test_uv;

b. Вычислите UV для каждого дня.
select dt, HLL_CARDINALITY(uv_set) from test_uv;

c. Вычислите агрегированное значение set1 в таблице test.
select dt, HLL_CARDINALITY(uv) from (select dt, HLL_RAW_AGG(set1) as uv
from test group by dt) tmp;
select dt, HLL_UNION_AGG(set1) as uv from test group by dt;
```

2 Функции

Selena предлагает богатый набор функций для облегчения запросов и анализа данных. Помимо часто используемых функций, Selena поддерживает полуструктурированные функции, такие как функции ARRAY, JSON, MAP и STRUCT. Также поддерживаются функции высшего порядка Lambda. Если эти функции не удовлетворяют вашим бизнес-требованиям, вы можете использовать Java UDF для компиляции функций. Selena также предоставляет Hive Bitmap UDF. Вы можете генерировать данные Bitmap в Hive, а затем загружать Bitmap в Selena. Вы также можете экспортировать данные Bitmap, сгенерированные в Selena, в Hive для использования другими системами.

2.1 Java UDF

Начиная с версии 2.2.0 вы можете компилировать пользовательские функции (UDF) для удовлетворения своих бизнес-потребностей, используя язык программирования Java.

Начиная с версии 3.0, Selena поддерживает глобальные UDF, и вам нужно только включить ключевое слово `GLOBAL` в соответствующие SQL-выражения (CREATE/SHOW/DROP).

В этом разделе описано, как разрабатывать и использовать различные UDF.

В настоящее время Selena поддерживает скалярные UDF, пользовательские агрегатные функции (UDAF), пользовательские оконные функции (UDWF) и пользовательские табличные функции (UDTF).

Предварительные требования

- Вы установили Apache Maven, чтобы создавать и компилировать Java-проекты.
- Вы установили JDK 17 на своих серверах.
- Функция Java UDF включена. Вы можете установить параметр конфигурации FE `enable_udf` в значение `true` в файле конфигурации FE **fe/conf/fe.conf**, чтобы включить эту функцию, а затем перезапустить узлы FE, чтобы настройки вступили в силу. Для получения дополнительной информации см. Настройка параметров.

Разработка и использование UDF

Вам нужно создать проект Maven и скомпилировать нужную UDF, используя язык программирования Java.

Шаг 1: Создание проекта Maven

Создайте проект Maven со следующей базовой структурой каталогов:

```
project
|--pom.xml
|--src
|  |--main
|  |  |--java
|  |  |--resources
|  |--test
|--target
```

Шаг 2: Добавление зависимостей

Добавьте следующие зависимости в файл **pom.xml**:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>udf</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.76</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-dependency-plugin</artifactId>
                <version>2.10</version>
                <executions>
                    <execution>
                        <id>copy-dependencies</id>
                        <phase>package</phase>
                        <goals>
                            <goal>copy-dependencies</goal>
                        </goals>
                        <configuration>

<outputDirectory>${project.build.directory}/lib</outputDirectory>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
            <plugin>
```

```
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-assembly-plugin</artifactId>
                <version>3.3.0</version>
                <executions>
                    <execution>
                        <id>make-assembly</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
                <configuration>
                    <descriptorRefs>
                        <descriptorRef>jar-with-
dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Шаг 3: Компиляция UDF

Используйте язык программирования Java для компиляции UDF.

Компиляция скалярной UDF

Скалярная UDF работает с одной строкой данных и возвращает одно значение. При использовании скалярной UDF в запросе каждая строка соответствует одному значению в результирующем наборе. Типичные скалярные функции включают `UPPER`, `LOWER`, `ROUND` и `ABS`.

Предположим, что значения поля в ваших JSON-данных являются JSON-строками, а не JSON-объектами. При использовании SQL-выражения для извлечения JSON-строк вам нужно выполнить `GET_JSON_STRING` дважды, например, `GET_JSON_STRING(GET_JSON_STRING('{"key":"{\\"k0\\":\\"v0\\"}"}', "$.key"), "$.k0")`.

Чтобы упростить SQL-выражение, вы можете скомпилировать скалярную UDF, которая может напрямую извлекать JSON-строки, например, `MY_UDF_JSON_GET('{"key":"`

`{\\"k0\\":\\"v0\\"}"}', "$.key.k0")`.

```
package com.selena.udf.sample;

import com.alibaba.fastjson.JSONPath;

public class UDFJsonGet {
    public final String evaluate(String obj, String key) {
        if (obj == null || key == null) return null;
        try {
            // Библиотека JSONPath может полностью развернуть, даже если
значения поля являются JSON-строками.
            return JSONPath.read(obj, key).toString();
        } catch (Exception e) {
            return null;
        }
    }
}
```

Пользовательский класс должен реализовать метод, описанный в следующей таблице.

> **ПРИМЕЧАНИЕ**
>
> Типы данных параметров запроса и возвращаемых параметров в методе должны совпадать с теми, которые объявлены в выражении `CREATE FUNCTION`, которое будет выполнено на Шаге 6, и соответствовать сопоставлению, приведённому в разделе "Сопоставление типов данных SQL и Java" этого документа.

Метод	Описание
TYPE1 evaluate(TYPE2, ...)	Запускает UDF. Метод evaluate() требует уровня доступа public.

Компиляция UDAF

UDAF работает с несколькими строками данных и возвращает одно значение. Типичные агрегатные функции включают `SUM`, `COUNT`, `MAX` и `MIN`, которые агрегируют несколько строк данных, указанных в каждом предложении GROUP BY, и возвращают одно значение.

Предположим, что вы хотите скомпилировать UDAF с именем `MY_SUM_INT`. В отличие от встроенной агрегатной функции `SUM`, которая возвращает значения типа BIGINT, функция

`MY_SUM_INT` поддерживает только параметры запроса и возвращаемые параметры типа INT.

```
package com.selena.udf.sample;

public class SumInt {
    public static class State {
        int counter = 0;
        public int serializeLength() { return 4; }
    }

    public State create() {
        return new State();
    }

    public void destroy(State state) {
    }

    public final void update(State state, Integer val) {
        if (val != null) {
            state.counter+= val;
        }
    }

    public void serialize(State state, java.nio.ByteBuffer buff) {
        buff.putInt(state.counter);
    }

    public void merge(State state, java.nio.ByteBuffer buffer) {
        int val = buffer.getInt();
        state.counter += val;
    }

    public Integer finalize(State state) {
        return state.counter;
    }
}
```

Пользовательский класс должен реализовать методы, описанные в следующей таблице.

> **ПРИМЕЧАНИЕ**

> Типы данных параметров запроса и возвращаемых параметров в методах должны совпадать с теми, которые объявлены в выражении `CREATE FUNCTION`, которое будет выполнено на Шаге 6, и соответствовать сопоставлению, приведённому в разделе "Сопоставление типов данных SQL и Java" этого документа.

Метод	Описание
State create()	Создаёт состояние.
void destroy(State)	Уничтожает состояние.
void update(State, ...)	Обновляет состояние. Помимо первого параметра `State`, вы также можете указать один или несколько параметров запроса в объявлении UDF.
void serialize(State, ByteBuffer)	Сериализует состояние в байтовый буфер.
void merge(State, ByteBuffer)	Десериализует состояние из байтового буфера и объединяет байтовый буфер с состоянием в качестве первого параметра.
TYPE finalize(State)	Получает окончательный результат UDF из состояния.

Во время компиляции вы также должны использовать класс буфера `java.nio.ByteBuffer` и локальную переменную `serializeLength`, которые описаны в следующей таблице.

Класс и локальная переменная	Описание
java.nio.ByteBuffer()	Класс буфера, который хранит промежуточные результаты. Промежуточные результаты могут быть сериализованы или десериализованы при передаче между узлами для выполнения. Поэтому вы также должны использовать переменную `serializeLength` для указания длины, разрешённой для десериализации промежуточных результатов.
serializeLength()	Длина, разрешённая для десериализации промежуточных результатов. Единица измерения: байты. Установите эту локальную переменную в значение типа INT. Например, `State { int counter = 0; public int serializeLength() { return 4; }}` указывает, что промежуточные результаты имеют тип INT, а длина для десериализации составляет 4 байта. Вы можете настроить эти параметры в соответствии с вашими бизнес-требованиями. Например, если вы хотите указать тип данных промежуточных результатов как LONG и длину для десериализации как 8 байт, передайте `State { long counter = 0; public int serializeLength() { return 8; }}`.

Обратите внимание на следующие моменты при десериализации промежуточных результатов, хранящихся в классе `java.nio.ByteBuffer`:

- Метод remaining(), зависящий от класса `ByteBuffer`, не может быть вызван для десериализации состояния.
- Метод clear() не может быть вызван для класса `ByteBuffer`.
- Значение `serializeLength` должно совпадать с длиной записанных данных. В противном случае при сериализации и десериализации будут сгенерированы неверные результаты.

Компиляция UDWF

В отличие от обычных агрегатных функций, UDWF работает с набором из нескольких строк, которые вместе называются окном, и возвращает значение для каждой строки. Типичная оконная функция включает предложение `OVER`, которое разделяет строки на несколько наборов. Она выполняет вычисление для каждого набора строк и возвращает значение для каждой строки.

Предположим, что вы хотите скомпилировать UDWF с именем `MY_WINDOW_SUM_INT`. В отличие от встроенной агрегатной функции `SUM`, которая возвращает значения типа BIGINT, функция `MY_WINDOW_SUM_INT` поддерживает только параметры запроса и возвращаемые параметры типа INT.

```java
package com.selena.udf.sample;

public class WindowSumInt {
    public static class State {
        int counter = 0;
        public int serializeLength() { return 4; }
        @Override
        public String toString() {
            return "State{" +
                    "counter=" + counter +
                    '}';
        }
    }

    public State create() {
        return new State();
    }

    public void destroy(State state) {

    }

    public void update(State state, Integer val) {
        if (val != null) {
            state.counter+=val;
        }
    }

    public void serialize(State state, java.nio.ByteBuffer buff) {
        buff.putInt(state.counter);
    }

    public void merge(State state, java.nio.ByteBuffer buffer) {
        int val = buffer.getInt();
        state.counter += val;
    }

    public Integer finalize(State state) {
        return state.counter;
    }

    public void reset(State state) {
        state.counter = 0;
    }
```

```
    public void windowUpdate(State state,
                             int peer_group_start, int peer_group_end,
                             int frame_start, int frame_end,
                             Integer[] inputs) {
        for (int i = (int)frame_start; i < (int)frame_end; ++i) {
            state.counter += inputs[i];
        }
    }
}
```

Пользовательский класс должен реализовать методы, требуемые для UDAF (поскольку UDWF является специальной агрегатной функцией), и метод windowUpdate(), описанный в следующей таблице.

> **ПРИМЕЧАНИЕ**
>
> Типы данных параметров запроса и возвращаемых параметров в методе должны совпадать с теми, которые объявлены в выражении `CREATE FUNCTION`, которое будет выполнено на Шаге 6, и соответствовать сопоставлению, приведённому в разделе "Сопоставление типов данных SQL и Java" этого документа.

Метод	Описание
void windowUpdate(State state, int, int, int , int, ...)	Обновляет данные окна. Для получения дополнительной информации о UDWF см. Оконные функции. Каждый раз, когда вы вводите строку в качестве входных данных, этот метод получает информацию об окне и соответственно обновляет промежуточные результаты. • `peer_group_start`: начальная позиция текущего partition. `PARTITION BY` используется в предложении OVER для указания столбца partition. Строки с одинаковыми значениями в столбце partition считаются находящимися в одном partition. • `peer_group_end`: конечная позиция текущего partition. • `frame_start`: начальная позиция текущего фрейма окна. Предложение фрейма окна указывает диапазон вычислений, который охватывает текущую строку и строки, находящиеся на указанном расстоянии от текущей строки. Например, `ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING` указывает диапазон вычислений, охватывающий текущую строку, предыдущую строку перед текущей строкой и следующую строку после текущей строки. • `frame_end`: конечная позиция текущего фрейма окна. • `inputs`: данные, которые вводятся в окно. Данные представляют собой пакет массива, который поддерживает только определённые типы данных. В этом примере вводятся значения INT, и пакет массива — Integer[].

Компиляция UDTF

UDTF читает одну строку данных и возвращает несколько значений, которые можно рассматривать как таблицу. Табличные функции обычно используются для преобразования строк в столбцы.

> **ПРИМЕЧАНИЕ**
>
> Selena позволяет UDTF возвращать таблицу, состоящую из нескольких строк и одного столбца.

Предположим, что вы хотите скомпилировать UDTF с именем `MY_UDF_SPLIT`. Функция `MY_UDF_SPLIT` позволяет использовать пробелы в качестве разделителей и поддерживает параметры запроса и возвращаемые параметры типа STRING.

```
package com.selena.udf.sample;

public class UDFSplit{
    public String[] process(String in) {
        if (in == null) return null;
        return in.split(" ");
    }
}
```

Метод, определённый пользовательским классом, должен соответствовать следующим требованиям:

> **ПРИМЕЧАНИЕ**
>
> Типы данных параметров запроса и возвращаемых параметров в методе должны совпадать с теми, которые объявлены в выражении `CREATE FUNCTION`, которое будет выполнено на Шаге 6, и соответствовать сопоставлению, приведённому в разделе "Сопоставление типов данных SQL и Java" этого документа.

Метод	Описание
TYPE[] process()	Запускает UDTF и возвращает массив.

Шаг 4: Упаковка Java-проекта

Выполните следующую команду для упаковки Java-проекта:

```
mvn package
```

Следующие JAR-файлы будут сгенерированы в папке **target**: **udf-1.0-SNAPSHOT.jar** и **udf-1.0-SNAPSHOT-jar-with-dependencies.jar**.

Шаг 5: Загрузка Java-проекта

Загрузите JAR-файл **udf-1.0-SNAPSHOT-jar-with-dependencies.jar** на HTTP-сервер, который работает и доступен для всех FE и BE в вашем cluster Selena. Затем выполните следующую команду для развёртывания файла:

```
mvn deploy
```

Вы можете настроить простой HTTP-сервер с помощью Python и загрузить JAR-файл на этот HTTP-сервер.

> **ПРИМЕЧАНИЕ**
>
> На Шаге 6 FE проверят JAR-файл, содержащий код для UDF, и вычислят контрольную сумму, а BE загрузят и выполнят JAR-файл.

Шаг 6: Создание UDF в Selena

Selena позволяет создавать UDF в двух типах пространств имён: пространства имён баз данных и глобальные пространства имён.

- Если у вас нет требований к видимости или изоляции для UDF, вы можете создать её как глобальную UDF. Тогда вы можете ссылаться на глобальную UDF, используя имя функции без включения имён каталога и базы данных в качестве префиксов к имени функции.
- Если у вас есть требования к видимости или изоляции для UDF, или если вам нужно создать одну и ту же UDF в разных базах данных, вы можете создать её в каждой отдельной базе данных. Таким образом, если ваша сессия подключена к целевой базе данных, вы можете ссылаться на UDF, используя имя функции. Если ваша сессия подключена к другому каталогу или базе данных, отличной от целевой, вам нужно ссылаться на UDF, включая имена каталога и базы данных в качестве префиксов к имени функции, например, `catalog.database.function`.

> **ВНИМАНИЕ**
>
> Перед созданием и использованием глобальной UDF вы должны связаться с системным администратором, чтобы получить необходимые разрешения. Для получения дополнительной информации см. GRANT.

После загрузки JAR-пакета вы можете создавать UDF в Selena. Для глобальной UDF вы должны включить ключевое слово `GLOBAL` в выражение создания.

Синтаксис

```
CREATE [GLOBAL][AGGREGATE | TABLE] FUNCTION function_name
(arg_type [, ...])
RETURNS return_type
PROPERTIES ("key" = "value" [, ...])
```

Параметры

Параметр	Обязательный	Описание
GLOBAL	Нет	Создавать ли глобальную UDF, поддерживается с версии 3.0.
AGGREGATE	Нет	Создавать ли UDAF или UDWF.
TABLE	Нет	Создавать ли UDTF. Если ни `AGGREGATE`, ни `TABLE` не указаны, создаётся скалярная функция.
function_name	Да	Имя функции, которую вы хотите создать. Вы можете включить имя базы данных в этот параметр, например, `db1.my_func`. Если `function_name` включает имя базы данных, UDF создаётся в этой базе данных. В противном случае UDF создаётся в текущей базе данных. Имя новой функции и её параметры не должны совпадать с существующим именем в целевой базе данных. В противном случае функция не может быть создана. Создание успешно, если имя функции совпадает, но параметры отличаются.
arg_type	Да	Тип аргумента функции. Добавленный аргумент может быть представлен как `, ...`. Для поддерживаемых типов данных см. Сопоставление типов данных SQL и Java.

Параметр	Обязательный	Описание
return_type	Да	Тип возвращаемого значения функции. Для поддерживаемых типов данных см. Java UDF.
PROPERTIES	Да	Свойства функции, которые зависят от типа создаваемой UDF.

Создание скалярной UDF

Выполните следующую команду для создания скалярной UDF, которую вы скомпилировали в предыдущем примере:

```
CREATE [GLOBAL] FUNCTION MY_UDF_JSON_GET(string, string)
RETURNS string
PROPERTIES (
    "symbol" = "com.selena.udf.sample.UDFJsonGet",
    "type" = "SelenaJar",
    "file" = "http://http_host:http_port/udf-1.0-SNAPSHOT-jar-with-
dependencies.jar"
);
```

Параметр	Описание
symbol	Имя класса для проекта Maven, к которому принадлежит UDF. Значение этого параметра имеет формат `<имя_пакета>.<имя_класса>`.
type	Тип UDF. Установите значение `SelenaJar`, которое указывает, что UDF является функцией на основе Java.
file	HTTP URL, с которого вы можете загрузить JAR-файл, содержащий код для UDF. Значение этого параметра имеет формат `http://<http_server_ip>:<http_server_port>/<jar_package_name>`.
isolation	(Необязательно) Чтобы разделять экземпляры функций между выполнениями UDF и поддерживать статические переменные, установите значение "shared".

Создание UDAF

Выполните следующую команду для создания UDAF, которую вы скомпилировали в предыдущем примере:

```
CREATE [GLOBAL] AGGREGATE FUNCTION MY_SUM_INT(INT)
RETURNS INT
PROPERTIES
(
    "symbol" = "com.selena.udf.sample.SumInt",
    "type" = "SelenaJar",
    "file" = "http://http_host:http_port/udf-1.0-SNAPSHOT-jar-with-
dependencies.jar"
);
```

Описания параметров в PROPERTIES такие же, как в разделе Создание скалярной UDF.

Создание UDWF

Выполните следующую команду для создания UDWF, которую вы скомпилировали в предыдущем примере:

```
CREATE [GLOBAL] AGGREGATE FUNCTION MY_WINDOW_SUM_INT(Int)
RETURNS Int
properties
(
    "analytic" = "true",
    "symbol" = "com.selena.udf.sample.WindowSumInt",
    "type" = "SelenaJar",
    "file" = "http://http_host:http_port/udf-1.0-SNAPSHOT-jar-with-
dependencies.jar"
);
```

`analytic`: Является ли UDF оконной функцией. Установите значение `true`. Описания других свойств такие же, как в разделе Создание скалярной UDF.

Создание UDTF

Выполните следующую команду для создания UDTF, которую вы скомпилировали в предыдущем примере:

```
CREATE [GLOBAL] TABLE FUNCTION MY_UDF_SPLIT(string)
RETURNS string
properties
(
    "symbol" = "com.selena.udf.sample.UDFSplit",
    "type" = "SelenaJar",
    "file" = "http://http_host:http_port/udf-1.0-SNAPSHOT-jar-with-
dependencies.jar"
);
```

Описания параметров в PROPERTIES такие же, как в разделе Создание скалярной UDF.

Шаг 7: Использование UDF

После создания UDF вы можете тестировать и использовать её в соответствии с вашими бизнес-потребностями.

Использование скалярной UDF

Выполните следующую команду для использования скалярной UDF, которую вы создали в предыдущем примере:

```
SELECT MY_UDF_JSON_GET('{"key":"{\\"in\\":2}"}', '$.key.in');
```

Использование UDAF

Выполните следующую команду для использования UDAF, которую вы создали в предыдущем примере:

```
SELECT MY_SUM_INT(col1);
```

Использование UDWF

Выполните следующую команду для использования UDWF, которую вы создали в предыдущем примере:

```
SELECT MY_WINDOW_SUM_INT(intcol)
            OVER (PARTITION BY intcol2
                  ORDER BY intcol3
                  ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
FROM test_basic;
```

Использование UDTF

Выполните следующую команду для использования UDTF, которую вы создали в предыдущем примере:

```
-- Предположим, у вас есть таблица с именем t1, и информация о её столбцах
a, b и c1 следующая:
SELECT t1.a,t1.b,t1.c1 FROM t1;
> output:
1,2.1,"hello world"
2,2.2,"hello UDTF."

-- Выполните функцию MY_UDF_SPLIT().
SELECT t1.a,t1.b, MY_UDF_SPLIT FROM t1, MY_UDF_SPLIT(t1.c1);
> output:
1,2.1,"hello"
1,2.1,"world"
2,2.2,"hello"
2,2.2,"UDTF."
```

> **ПРИМЕЧАНИЕ**
>
> - Первый `MY_UDF_SPLIT` в приведённом выше фрагменте кода является псевдонимом столбца, возвращаемого вторым `MY_UDF_SPLIT`, который является функцией.
> - Вы не можете использовать `AS t2(f1)` для указания псевдонимов таблицы и её столбцов, которые будут возвращены.

Просмотр UDF

Выполните следующую команду для запроса UDF:

```
SHOW [GLOBAL] FUNCTIONS;
```

Для получения дополнительной информации см. SHOW FUNCTIONS.

Удаление UDF

Выполните следующую команду для удаления UDF:

```
DROP [GLOBAL] FUNCTION <function_name>(arg_type [, ...]);
```

Для получения дополнительной информации см. DROP FUNCTION.

Сопоставление типов данных SQL и Java

> **ПРИМЕЧАНИЕ**
>
> В настоящее время для скалярных UDF поддерживаются только невложенные типы параметров/возвращаемых значений ARRAY и MAP.

Тип SQL	Тип Java
BOOLEAN	java.lang.Boolean
TINYINT	java.lang.Byte
SMALLINT	java.lang.Short
INT	java.lang.Integer
BIGINT	java.lang.Long
FLOAT	java.lang.Float
DOUBLE	java.lang.Double
STRING/VARCHAR	java.lang.String
ARRAY	java.util.List
Map	java.util.Map

Настройки параметров

Настройте следующую переменную окружения в файле **be/conf/be.conf** каждой виртуальной машины Java (JVM) в вашем cluster Selena для управления использованием памяти.

```
JAVA_OPTS="-Xmx12G"
```

FAQ

Могу ли я использовать статические переменные при создании UDF? Влияют ли статические переменные разных UDF друг на друга?

Да, вы можете использовать статические переменные при компиляции UDF. Статические переменные разных UDF изолированы друг от друга и не влияют друг на друга, даже если UDF имеют классы с одинаковыми именами.

2.2 Лямбда-выражения

Лямбда-выражения — это анонимные функции, которые можно передавать в качестве параметров в функции высшего порядка SQL. Лямбда-выражения позволяют разрабатывать более краткий, элегантный и расширяемый код.

Лямбда-выражения записываются с помощью оператора `->`, который читается как "переходит к". Слева от `->` располагаются входные параметры (если есть), а справа — выражение.

Начиная с версии 2.5, Selena поддерживает использование лямбда-выражений в следующих функциях высшего порядка SQL: array_map(), array_filter(), array_sum() и array_sortby().

Синтаксис

```
parameter -> expression
```

Параметры

- `parameter`: входные параметры для лямбда-выражения, которые могут принимать ноль, один или несколько параметров. Два или более входных параметра заключаются в круглые скобки.

- `expression`: простое выражение, которое ссылается на `parameter`. Выражение должно быть допустимым для входных параметров.

Возвращаемое значение

Тип возвращаемого значения определяется типом результата `expression`.

Примечания по использованию

Практически все скалярные функции можно использовать в теле лямбда-выражения. Однако есть несколько исключений:

- Подзапросы не поддерживаются, например, `x -> 5 + (SELECT 3)`.
- Агрегатные функции не поддерживаются, например, `x -> min(y)`.
- Оконные функции не поддерживаются.
- Табличные функции не поддерживаются.
- Коррелированные столбцы не могут встречаться в лямбда-функциях.

Примеры

Простые примеры лямбда-выражений:

```
-- Не принимает параметров и возвращает 5.
() -> 5
-- Принимает x и возвращает значение (x + 2).
x -> x + 2
-- Принимает x и y и возвращает их сумму.
(x, y) -> x + y
-- Принимает x и применяет функцию к x.
x -> COALESCE(x, 0)
x -> day(x)
x -> split(x,",")
x -> if(x>0,"positive","negative")
```

Пример использования лямбда-выражений в функциях высшего порядка:

```
select array_map((x,y,z) -> x + y, [1], [2], [4]);
+-----------------------------------------------+
| array_map((x, y, z) -> x + y, [1], [2], [4]) |
+-----------------------------------------------+
| [3]                                           |
+-----------------------------------------------+
1 row in set (0.01 sec)
```

2.3 Оконные функции

Введение

Оконная функция — это особый класс встроенных функций. Подобно агрегатным функциям, она также выполняет вычисления над несколькими входными строками для получения одного значения данных. Разница в том, что оконная функция обрабатывает входные данные в пределах определённого окна, а не использует метод "group by". Данные в каждом окне могут быть отсортированы и сгруппированы с использованием предложения over(). Оконная функция **вычисляет отдельное значение для каждой строки**, а не одно значение для каждой группы. Эта гибкость позволяет пользователям добавлять дополнительные столбцы в предложение select и дополнительно фильтровать результирующий набор. Оконная функция может появляться только в списке select и во внешней позиции предложения. Она применяется в конце запроса, то есть после выполнения операций `join`, `where` и `group by`. Оконная функция часто используется для анализа трендов, вычисления выбросов и выполнения анализа распределения по корзинам на больших данных.

Использование

Синтаксис

```
function(args) OVER([partition_by_clause] [order_by_clause]
[order_by_clause window_clause])
partition_by_clause ::= PARTITION BY expr [, expr ...]
order_by_clause ::= ORDER BY expr [ASC | DESC] [, expr [ASC | DESC] ...]
```

Предложение PARTITION BY

Предложение Partition By аналогично Group By. Оно группирует входные строки по одному или нескольким указанным столбцам. Строки с одинаковым значением группируются вместе.

Предложение ORDER BY

Предложение `Order By` в основном такое же, как внешнее `Order By`. Оно определяет порядок входных строк. Если указано `Partition By`, `Order By` определяет порядок внутри каждой группы Partition. Единственное отличие в том, что `Order By n` (n —

положительное целое число) в предложении `OVER` эквивалентно отсутствию операции, тогда как `n` во внешнем `Order By` указывает на сортировку по n-му столбцу.

Пример:

Этот пример показывает добавление столбца id в список select со значениями 1, 2, 3 и т.д., отсортированными по столбцу `date_and_time` в таблице events.

```
SELECT row_number() OVER (ORDER BY date_and_time) AS id,
    c1, c2, c3, c4
FROM events;
```

Предложение Window

Предложение window используется для указания диапазона строк для операций (предшествующие и следующие строки относительно текущей строки). Оно поддерживает следующие синтаксисы: AVG(), COUNT(), FIRST_VALUE(), LAST_VALUE() и SUM(). Для MAX() и MIN() предложение window может указывать начало до `UNBOUNDED PRECEDING`.

Синтаксис:

```
ROWS BETWEEN [ { m | UNBOUNDED } PRECEDING | CURRENT ROW] [ AND [CURRENT
ROW | { UNBOUNDED | n } FOLLOWING] ]
RANGE BETWEEN [ { m | UNBOUNDED } PRECEDING | CURRENT ROW] [ AND [CURRENT
ROW | { UNBOUNDED | n } FOLLOWING] ]
```

> **ПРИМЕЧАНИЕ**
>
> **Ограничение оконного фрейма ARRAY_AGG():** При использовании ARRAY_AGG() как оконной функции поддерживаются только фреймы RANGE. Фреймы ROWS НЕ поддерживаются. Например:
>
> ```
> -- Поддерживается: фрейм RANGE
> array_agg(col) OVER (PARTITION BY x ORDER BY y RANGE BETWEEN UNBOUNDED
> PRECEDING AND CURRENT ROW)
>
> -- НЕ поддерживается: фрейм ROWS (вызовет ошибку)
> array_agg(col) OVER (PARTITION BY x ORDER BY y ROWS BETWEEN 1
> PRECEDING AND 1 FOLLOWING)
> ```

Пример таблицы для оконных функций

В этом разделе создаётся примерная таблица `scores`. Вы можете использовать эту таблицу для тестирования многих оконных функций ниже.

```
CREATE TABLE `scores` (
    `id` int(11) NULL,
    `name` varchar(11) NULL,
    `subject` varchar(11) NULL,
    `score` int(11) NULL
  )
DISTRIBUTED BY HASH(`score`) BUCKETS 10;

INSERT INTO `scores` VALUES
  (1, "lily", "math", NULL),
  (1, "lily", "english", 100),
  (1, "lily", "physics", 60),
  (2, "tom", "math", 80),
  (2, "tom", "english", 98),
  (2, "tom", "physics", NULL),
  (3, "jack", "math", 95),
  (3, "jack", "english", NULL),
  (3, "jack", "physics", 99),
  (4, "amy", "math", 80),
  (4, "amy", "english", 92),
  (4, "amy", "physics", 99),
  (5, "mike", "math", 70),
  (5, "mike", "english", 85),
  (5, "mike", "physics", 85),
  (6, "amber", "math", 92),
  (6, "amber", NULL, 90),
  (6, "amber", "physics", 100);
```

Примеры функций

В этом разделе описаны оконные функции, поддерживаемые в Selena.

AVG()

Вычисляет среднее значение поля в заданном окне. Эта функция игнорирует значения NULL.

Синтаксис:

```
AVG([DISTINCT] expr) [OVER (*analytic_clause*)]
```

`DISTINCT` поддерживается начиная с Selena v2.0.0. При указании AVG() вычисляет среднее только уникальных значений в окне.

> **ПРИМЕЧАНИЕ**
>
> **Ограничение оконного фрейма:** При использовании AVG(DISTINCT) как оконной функции поддерживаются только фреймы RANGE. Фреймы ROWS НЕ поддерживаются.

Примеры:

В следующем примере используются данные акций.

```
CREATE TABLE stock_ticker (
    stock_symbol  STRING,
    closing_price DECIMAL(8,2),
    closing_date  DATETIME
)
DUPLICATE KEY(stock_symbol)
COMMENT "OLAP"
DISTRIBUTED BY HASH(closing_date);

INSERT INTO stock_ticker VALUES
    ("JDR", 12.86, "2014-10-02 00:00:00"),
    ("JDR", 12.89, "2014-10-03 00:00:00"),
    ("JDR", 12.94, "2014-10-04 00:00:00"),
    ("JDR", 12.55, "2014-10-05 00:00:00"),
    ("JDR", 14.03, "2014-10-06 00:00:00"),
    ("JDR", 14.75, "2014-10-07 00:00:00"),
    ("JDR", 13.98, "2014-10-08 00:00:00")
;
```

Вычисление средней цены закрытия для текущей строки и каждой строки до и после неё.

```
select stock_symbol, closing_date, closing_price,
    avg(closing_price)
        over (partition by stock_symbol
              order by closing_date
              rows between 1 preceding and 1 following
        ) as moving_average
from stock_ticker;
```

Вывод:

```
+--------------+---------------------+---------------+----------------+
| stock_symbol | closing_date        | closing_price | moving_average |
+--------------+---------------------+---------------+----------------+
| JDR          | 2014-10-02 00:00:00 |         12.86 |    12.87500000 |
| JDR          | 2014-10-03 00:00:00 |         12.89 |    12.89666667 |
| JDR          | 2014-10-04 00:00:00 |         12.94 |    12.79333333 |
| JDR          | 2014-10-05 00:00:00 |         12.55 |    13.17333333 |
| JDR          | 2014-10-06 00:00:00 |         14.03 |    13.77666667 |
| JDR          | 2014-10-07 00:00:00 |         14.75 |    14.25333333 |
| JDR          | 2014-10-08 00:00:00 |         13.98 |    14.36500000 |
+--------------+---------------------+---------------+----------------+
```

Например, `12.87500000` в первой строке — это среднее значение цен закрытия на "2014-10-02" (`12.86`), предыдущий день "2014-10-01" (null) и следующий день "2014-10-03" (`12.89`).

Пример 2: Использование AVG(DISTINCT) по всему окну

Вычисление среднего уникальных оценок по всем строкам:

```
SELECT id, subject, score,
    AVG(DISTINCT score) OVER () AS distinct_avg
FROM test_scores;
```

Вывод:

```
+----+---------+-------+-------------+
| id | subject | score | distinct_avg|
+----+---------+-------+-------------+
|  1 | math    |    80 |       85.00 |
|  2 | math    |    85 |       85.00 |
|  3 | math    |    80 |       85.00 |
|  4 | english |    90 |       85.00 |
|  5 | english |    85 |       85.00 |
|  6 | english |    90 |       85.00 |
+----+---------+-------+-------------+
```

Среднее уникальных значений равно 85.00 ((80 + 85 + 90) / 3).

Пример 3: Использование AVG(DISTINCT) по фреймированному окну с фреймом RANGE

Вычисление среднего уникальных оценок в каждом разделе предмета с использованием фрейма RANGE:

```
SELECT id, subject, score,
    AVG(DISTINCT score) OVER (
        PARTITION BY subject
        ORDER BY score
        RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    ) AS distinct_avg
FROM test_scores;
```

Вывод:

```
+----+---------+-------+-------------+
| id | subject | score | distinct_avg|
+----+---------+-------+-------------+
|  1 | math    |    80 |       80.00 |
|  3 | math    |    80 |       80.00 |
|  2 | math    |    85 |       82.50 |
|  5 | english |    85 |       85.00 |
|  4 | english |    90 |       87.50 |
|  6 | english |    90 |       87.50 |
+----+---------+-------+-------------+
```

Для каждой строки функция вычисляет среднее уникальных оценок от начала раздела до текущего значения оценки строки включительно.

ARRAY_AGG()

Агрегирует значения (включая значения NULL) в окне в массив. Вы можете использовать необязательное предложение `ORDER BY` для сортировки элементов внутри массива.

Эта функция поддерживается начиная с v1.5.2.

> **ПОДСКАЗКА**
>
> **Важное ограничение:** ARRAY_AGG() как оконная функция **поддерживает только оконные фреймы RANGE**. Оконные фреймы ROWS НЕ поддерживаются. Если оконный фрейм не указан, используется значение по умолчанию `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`.

Синтаксис:

```
ARRAY_AGG([DISTINCT] expr [ORDER BY expr [ASC | DESC]])
OVER([partition_by_clause] [order_by_clause] [window_clause])
```

Параметры:

- `expr`: Выражение для агрегации. Может быть столбцом любого поддерживаемого типа данных.
- `DISTINCT`: Необязательно. Удаляет дублирующиеся значения из результирующего массива.
- `ORDER BY`: Необязательно. Указывает порядок элементов внутри массива.

Возвращаемое значение:

Возвращает ARRAY, содержащий все значения в окне.

Примечания по использованию:

- **Фреймы ROWS НЕ поддерживаются.** Только фреймы RANGE могут использоваться с ARRAY_AGG() как оконной функцией. Использование фреймов ROWS приведёт к ошибке.

- Значения NULL включаются в результирующий массив.
- При указании `DISTINCT` дублирующиеся значения удаляются из массива.
- При указании `ORDER BY` внутри ARRAY_AGG() элементы в результирующем массиве сортируются соответственно.

Примеры:

Этот пример использует данные из Примера таблицы `scores`.

Пример 1: Базовый ARRAY_AGG() по окну

Сбор всех оценок в каждом разделе предмета:

```
SELECT *,
    array_agg(score)
        OVER (
            PARTITION BY subject
            ORDER BY score
        ) AS score_array
FROM scores
WHERE subject = 'math';
```

Вывод:

```
+------+-------+---------+-------+----------------------+
| id   | name  | subject | score | score_array          |
+------+-------+---------+-------+----------------------+
|    1 | lily  | math    |  NULL | [null]               |
|    5 | mike  | math    |    70 | [null,70]            |
|    2 | tom   | math    |    80 | [null,70,80,80]      |
|    4 | amy   | math    |    80 | [null,70,80,80]      |
|    6 | amber | math    |    92 | [null,70,80,80,92]   |
|    3 | jack  | math    |    95 | [null,70,80,80,92,95]|
+------+-------+---------+-------+----------------------+
```

Примечание: Строки с одинаковым значением `score` (tom и amy оба имеют 80) получают один и тот же массив из-за семантики фрейма RANGE.

Пример 2: ARRAY_AGG(DISTINCT) по окну

Сбор уникальных оценок в каждом разделе предмета:

```
SELECT *,
    array_agg(DISTINCT score)
        OVER (
            PARTITION BY subject
            ORDER BY score
        ) AS distinct_scores
FROM scores
WHERE subject = 'math';
```

Вывод:

```
+------+-------+---------+-------+-------------------+
| id   | name  | subject | score | distinct_scores   |
+------+-------+---------+-------+-------------------+
|    1 | lily  | math    |  NULL | [null]            |
|    5 | mike  | math    |    70 | [null,70]         |
|    2 | tom   | math    |    80 | [null,70,80]      |
|    4 | amy   | math    |    80 | [null,70,80]      |
|    6 | amber | math    |    92 | [null,70,80,92]   |
|    3 | jack  | math    |    95 | [null,70,80,92,95]|
+------+-------+---------+-------+-------------------+
```

Пример 3: ARRAY_AGG() с ORDER BY

Сбор оценок, отсортированных по убыванию внутри массива:

```
SELECT *,
    array_agg(score ORDER BY score DESC)
        OVER (
            PARTITION BY subject
        ) AS scores_desc
FROM scores
WHERE subject = 'math';
```

Вывод:

```
+------+-------+---------+-------+----------------------+
| id   | name  | subject | score | scores_desc          |
+------+-------+---------+-------+----------------------+
|    1 | lily  | math    |  NULL | [95,92,80,80,70,null]|
|    5 | mike  | math    |    70 | [95,92,80,80,70,null]|
|    2 | tom   | math    |    80 | [95,92,80,80,70,null]|
|    4 | amy   | math    |    80 | [95,92,80,80,70,null]|
|    6 | amber | math    |    92 | [95,92,80,80,70,null]|
|    3 | jack  | math    |    95 | [95,92,80,80,70,null]|
+------+-------+---------+-------+----------------------+
```

Пример 4: ARRAY_AGG() с фреймом RANGE

Сбор всех оценок во всём разделе с использованием RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING:

```
SELECT *,
    array_agg(score)
        OVER (
            PARTITION BY subject
            ORDER BY score
            RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
        ) AS all_scores
FROM scores
WHERE subject = 'math';
```

Вывод:

```
+------+-------+---------+-------+----------------------+
| id   | name  | subject | score | all_scores           |
+------+-------+---------+-------+----------------------+
|    1 | lily  | math    |  NULL | [null,70,80,80,92,95]|
|    5 | mike  | math    |    70 | [null,70,80,80,92,95]|
|    2 | tom   | math    |    80 | [null,70,80,80,92,95]|
|    4 | amy   | math    |    80 | [null,70,80,80,92,95]|
|    6 | amber | math    |    92 | [null,70,80,80,92,95]|
|    3 | jack  | math    |    95 | [null,70,80,80,92,95]|
+------+-------+---------+-------+----------------------+
```

Пример 5: Сбор имён, разделённых по диапазонам оценок

Используя таблицу `stock_ticker`, сбор символов акций в скользящем окне:

```
SELECT
    stock_symbol,
    closing_date,
    closing_price,
    array_agg(closing_price)
        OVER (
            PARTITION BY stock_symbol
            ORDER BY closing_date
        ) AS price_history
FROM stock_ticker;
```

Вывод:

```
+--------------+---------------------+---------------+---------------------
------------------+
| stock_symbol | closing_date        | closing_price | price_history
|
+--------------+---------------------+---------------+---------------------
------------------+
| JDR          | 2014-10-02 00:00:00 |         12.86 | [12.86]
|
| JDR          | 2014-10-03 00:00:00 |         12.89 | [12.86,12.89]
|
| JDR          | 2014-10-04 00:00:00 |         12.94 | [12.86,12.89,12.94]
|
| JDR          | 2014-10-05 00:00:00 |         12.55 |
[12.86,12.89,12.94,12.55]             |
| JDR          | 2014-10-06 00:00:00 |         14.03 |
[12.86,12.89,12.94,12.55,14.03]       |
| JDR          | 2014-10-07 00:00:00 |         14.75 |
[12.86,12.89,12.94,12.55,14.03,14.75] |
| JDR          | 2014-10-08 00:00:00 |         13.98 |
[12.86,12.89,12.94,12.55,14.03,14.75,13.98] |
+--------------+---------------------+---------------+---------------------
------------------+
```

Пример 6: Неверное использование - фрейм ROWS (вызовет ошибку)

Следующий запрос завершится неудачей, поскольку фреймы ROWS не поддерживаются:

```
-- Это вызовет ошибку!
SELECT *,
    array_agg(score)
        OVER (
            PARTITION BY subject
            ORDER BY score
            ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING  -- НЕ ПОДДЕРЖИВАЕТСЯ!
        ) AS score_array
FROM scores;
```

Сообщение об ошибке:

```
ERROR: array_agg as window function does not support ROWS frame type.
Please use RANGE frame instead.
```

COUNT()

Вычисляет общее количество строк, соответствующих указанным условиям в заданном окне.

Синтаксис:

```
COUNT([DISTINCT] expr) [OVER (analytic_clause)]
```

`DISTINCT` поддерживается начиная с Selena v2.0.0. При указании COUNT() подсчитывает только уникальные значения в окне.

> **ПРИМЕЧАНИЕ**
>
> **Ограничение оконного фрейма:** При использовании COUNT(DISTINCT) как оконной функции поддерживаются только фреймы RANGE. Фреймы ROWS НЕ поддерживаются. Например:
>
> ```
> -- Поддерживается: фрейм RANGE
> count(distinct col) OVER (PARTITION BY x ORDER BY y RANGE BETWEEN
> UNBOUNDED PRECEDING AND CURRENT ROW)
>
> -- НЕ поддерживается: фрейм ROWS (вызовет ошибку)
> count(distinct col) OVER (PARTITION BY x ORDER BY y ROWS BETWEEN 1
> PRECEDING AND 1 FOLLOWING)
> ```

Примеры:

Подсчёт появлений оценок по математике, которые больше 90, от текущей строки до первой строки в разделе математики. Этот пример использует данные из Примера таблицы `scores`.

```
select *,
    count(score)
        over (
            partition by subject
            order by score
            rows between unbounded preceding and current row
        ) as 'score_count'
from scores where subject in ('math') and score > 90;
```

```
+------+-------+---------+-------+-------------+
| id   | name  | subject | score | score_count |
+------+-------+---------+-------+-------------+
|    6 | amber | math    |    92 |           1 |
|    3 | jack  | math    |    95 |           2 |
+------+-------+---------+-------+-------------+
```

Пример 2: Использование COUNT(DISTINCT) по всему окну

Подсчёт уникальных оценок по всем строкам:

```sql
CREATE TABLE test_scores (
    id INT,
    subject VARCHAR(20),
    score INT
) DISTRIBUTED BY HASH(id);

INSERT INTO test_scores VALUES
    (1, 'math', 80),
    (2, 'math', 85),
    (3, 'math', 80),
    (4, 'english', 90),
    (5, 'english', 85),
    (6, 'english', 90);
```

```sql
SELECT id, subject, score,
    COUNT(DISTINCT score) OVER () AS distinct_count
FROM test_scores;
```

Вывод:

```
+----+---------+-------+---------------+
| id | subject | score | distinct_count|
+----+---------+-------+---------------+
|  1 | math    |    80 |             4 |
|  2 | math    |    85 |             4 |
|  3 | math    |    80 |             4 |
|  4 | english |    90 |             4 |
|  5 | english |    85 |             4 |
|  6 | english |    90 |             4 |
+----+---------+-------+---------------+
```

Количество уникальных значений равно 4 (значения: 80, 85, 90 и NULL, если есть).

Пример 3: Использование COUNT(DISTINCT) по фреймированному окну с фреймом RANGE

Подсчёт уникальных оценок в каждом разделе предмета с использованием фрейма RANGE:

```
SELECT id, subject, score,
    COUNT(DISTINCT score) OVER (
        PARTITION BY subject
        ORDER BY score
        RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    ) AS distinct_count
FROM test_scores;
```

Вывод:

```
+----+---------+-------+---------------+
| id | subject | score | distinct_count|
+----+---------+-------+---------------+
|  1 | math    |    80 |             1 |
|  3 | math    |    80 |             1 |
|  2 | math    |    85 |             2 |
|  5 | english |    85 |             1 |
|  4 | english |    90 |             2 |
|  6 | english |    90 |             2 |
+----+---------+-------+---------------+
```

Для каждой строки функция подсчитывает уникальные оценки от начала раздела до текущего значения оценки строки включительно.

CUME_DIST()

Функция CUME_DIST() вычисляет кумулятивное распределение значения в разделе или окне, указывая его относительную позицию как процент в разделе. Часто используется для вычисления распределения наивысших или наименьших значений в группе.

- Если данные отсортированы по возрастанию, эта функция вычисляет процент значений, меньших или равных значению в текущей строке.
- Если данные отсортированы по убыванию, эта функция вычисляет процент значений, больших или равных значению в текущей строке.

Кумулятивное распределение находится в диапазоне от 0 до 1. Это полезно для вычисления процентилей и анализа распределения данных.

Эта функция поддерживается начиная с v1.5.2.

Синтаксис:

```
CUME_DIST() OVER (partition_by_clause order_by_clause)
```

- `partition_by_clause`: необязательно. Если это предложение не указано, весь результирующий набор обрабатывается как один раздел.
- `order_by_clause`: **Эта функция должна использоваться с ORDER BY для сортировки строк раздела в желаемом порядке.**

CUME_DIST() содержит значения NULL и рассматривает их как наименьшие значения.

Примеры:

Следующий пример показывает кумулятивное распределение каждой оценки в каждой группе `subject`. Этот пример использует данные из Примера таблицы `scores`.

```
SELECT *,
    cume_dist()
      OVER (
        PARTITION BY subject
        ORDER BY score
      ) AS cume_dist
FROM scores;
+------+-------+---------+-------+---------------------+
| id   | name  | subject | score | cume_dist           |
+------+-------+---------+-------+---------------------+
|    6 | amber | NULL    |    90 |                   1 |
|    3 | jack  | english |  NULL |                 0.2 |
|    5 | mike  | english |    85 |                 0.4 |
|    4 | amy   | english |    92 |                 0.6 |
|    2 | tom   | english |    98 |                 0.8 |
|    1 | lily  | english |   100 |                   1 |
|    1 | lily  | math    |  NULL | 0.16666666666666666 |
|    5 | mike  | math    |    70 |  0.3333333333333333 |
|    2 | tom   | math    |    80 |  0.6666666666666666 |
|    4 | amy   | math    |    80 |  0.6666666666666666 |
|    6 | amber | math    |    92 |  0.8333333333333334 |
|    3 | jack  | math    |    95 |                   1 |
|    2 | tom   | physics |  NULL | 0.16666666666666666 |
|    1 | lily  | physics |    60 |  0.3333333333333333 |
|    5 | mike  | physics |    85 |                 0.5 |
|    4 | amy   | physics |    99 |  0.8333333333333334 |
|    3 | jack  | physics |    99 |  0.8333333333333334 |
|    6 | amber | physics |   100 |                   1 |
+------+-------+---------+-------+---------------------+
```

- Для `cume_dist` в первой строке группа `NULL` имеет только одну строку, и только эта строка сама соответствует условию "меньше или равно текущей строке". Кумулятивное распределение равно 1.
- Для `cume_dist` во второй строке группа `english` имеет пять строк, и только эта строка сама (NULL) соответствует условию "меньше или равно текущей строке". Кумулятивное распределение равно 0.2.
- Для `cume_dist` в третьей строке группа `english` имеет пять строк, и две строки (85 и NULL) соответствуют условию "меньше или равно текущей строке". Кумулятивное распределение равно 0.4.

DENSE_RANK()

Функция DENSE_RANK() используется для представления рангов. В отличие от RANK(), DENSE_RANK() **не имеет пропусков** в номерах. Например, если есть две 1, третий номер DENSE_RANK() всё равно будет 2, тогда как третий номер RANK() будет 3.

Синтаксис:

```
DENSE_RANK() OVER(partition_by_clause order_by_clause)
```

Примеры:

Следующий пример показывает ранжирование оценок по математике (отсортированных по убыванию). Этот пример использует данные из Примера таблицы `scores`.

```
select *,
    dense_rank()
        over (
            partition by subject
            order by score desc
        ) as `rank`
from scores where subject in ('math');
```

```
+------+-------+---------+-------+------+
| id   | name  | subject | score | rank |
+------+-------+---------+-------+------+
|    3 | jack  | math    |    95 |    1 |
|    6 | amber | math    |    92 |    2 |
|    2 | tom   | math    |    80 |    3 |
|    4 | amy   | math    |    80 |    3 |
|    5 | mike  | math    |    70 |    4 |
|    1 | lily  | math    |  NULL |    5 |
+------+-------+---------+-------+------+
```

Результирующие данные имеют две строки с оценкой 80. Они все имеют ранг 3. Ранг для следующей оценки 70 равен 4. Это показывает, что DENSE_RANK() **не имеет пропусков** в номерах.

FIRST_VALUE()

FIRST_VALUE() возвращает **первое** значение диапазона окна.

Синтаксис:

```
FIRST_VALUE(expr [IGNORE NULLS]) OVER(partition_by_clause order_by_clause
[window_clause])
```

`IGNORE NULLS` поддерживается начиная с v1.5.2. Используется для определения, исключаются ли значения NULL `expr` из расчёта. По умолчанию значения NULL включаются, что означает, что NULL возвращается, если первое значение в отфильтрованном результате равно NULL. Если вы указываете IGNORE NULLS, возвращается первое ненулевое значение в отфильтрованном результате. Если все значения равны NULL, NULL возвращается даже при указании IGNORE NULLS.

Типы ARRAY поддерживаются начиная с Selena v1.5.2. Вы можете использовать FIRST_VALUE() со столбцами ARRAY для получения первого значения массива в окне.

Примеры:

Возврат первого значения `score` для каждого участника в каждой группе (по убыванию), группировка по `subject`. Этот пример использует данные из Примера таблицы `scores`.

```
select *,
    first_value(score IGNORE NULLS)
        over (
            partition by subject
            order by score desc
        ) as first
from scores;
```

```
+------+-------+---------+-------+-------+
| id   | name  | subject | score | first |
+------+-------+---------+-------+-------+
|    1 | lily  | english |   100 |   100 |
|    2 | tom   | english |    98 |   100 |
|    4 | amy   | english |    92 |   100 |
|    5 | mike  | english |    85 |   100 |
|    3 | jack  | english |  NULL |   100 |
|    6 | amber | physics |   100 |   100 |
|    3 | jack  | physics |    99 |   100 |
|    4 | amy   | physics |    99 |   100 |
|    5 | mike  | physics |    85 |   100 |
|    1 | lily  | physics |    60 |   100 |
|    2 | tom   | physics |  NULL |   100 |
|    6 | amber | NULL    |    90 |    90 |
|    3 | jack  | math    |    95 |    95 |
|    6 | amber | math    |    92 |    95 |
|    2 | tom   | math    |    80 |    95 |
|    4 | amy   | math    |    80 |    95 |
|    5 | mike  | math    |    70 |    95 |
|    1 | lily  | math    |  NULL |    95 |
+------+-------+---------+-------+-------+
```

Пример 2: Использование FIRST_VALUE() с типами ARRAY

Создание таблицы со столбцами ARRAY:

```
CREATE TABLE test_array_value (
    col_1 INT,
    arr1 ARRAY<INT>
) DISTRIBUTED BY HASH(col_1);

INSERT INTO test_array_value (col_1, arr1) VALUES
    (1, [1, 11]),
    (2, [2, 22]),
    (3, [3, 33]),
    (4, NULL),
    (5, [5, 55]);
```

Запрос данных с использованием FIRST_VALUE() с типами ARRAY:

```
SELECT col_1, arr1,
    FIRST_VALUE(arr1) OVER (ORDER BY col_1) AS first_array
FROM test_array_value;
```

Вывод:

```
+-------+--------+------------+
| col_1 | arr1   | first_array|
+-------+--------+------------+
|     1 | [1,11] | [1,11]     |
|     2 | [2,22] | [1,11]     |
|     3 | [3,33] | [1,11]     |
|     4 | NULL   | [1,11]     |
|     5 | [5,55] | [1,11]     |
+-------+--------+------------+
```

Первое значение массива `[1,11]` возвращается для всех строк в окне.

LAST_VALUE()

LAST_VALUE() возвращает **последнее** значение диапазона окна. Это противоположность FIRST_VALUE().

Синтаксис:

```
LAST_VALUE(expr [IGNORE NULLS]) OVER(partition_by_clause order_by_clause
[window_clause])
```

`IGNORE NULLS` поддерживается начиная с v1.5.2. Используется для определения, исключаются ли значения NULL `expr` из расчёта. По умолчанию значения NULL включаются, что означает, что NULL возвращается, если последнее значение в отфильтрованном результате равно NULL. Если вы указываете IGNORE NULLS, возвращается последнее ненулевое значение в отфильтрованном результате. Если все значения равны NULL, NULL возвращается даже при указании IGNORE NULLS.

По умолчанию LAST_VALUE() вычисляет `rows between unbounded preceding and current row`, что сравнивает текущую строку со всеми предшествующими строками. Если

вы хотите показать только одно значение для каждого раздела, используйте `rows between unbounded preceding and unbounded following` после ORDER BY.

Типы ARRAY поддерживаются начиная с Selena v1.5.2. Вы можете использовать LAST_VALUE() со столбцами ARRAY для получения последнего значения массива в окне.

Примеры:

Возврат последней `score` для каждого участника в группе (по убыванию), группировка по `subject`. Этот пример использует данные из Примера таблицы `scores`.

```
select *,
    last_value(score IGNORE NULLS)
        over (
            partition by subject
            order by score desc
            rows between unbounded preceding and unbounded following
        ) as last
from scores;
```

```
+------+-------+---------+-------+------+
| id   | name  | subject | score | last |
+------+-------+---------+-------+------+
|    1 | lily  | english |   100 |   85 |
|    2 | tom   | english |    98 |   85 |
|    4 | amy   | english |    92 |   85 |
|    5 | mike  | english |    85 |   85 |
|    3 | jack  | english |  NULL |   85 |
|    6 | amber | physics |   100 |   60 |
|    3 | jack  | physics |    99 |   60 |
|    4 | amy   | physics |    99 |   60 |
|    5 | mike  | physics |    85 |   60 |
|    1 | lily  | physics |    60 |   60 |
|    2 | tom   | physics |  NULL |   60 |
|    6 | amber | NULL    |    90 |   90 |
|    3 | jack  | math    |    95 |   70 |
|    6 | amber | math    |    92 |   70 |
|    2 | tom   | math    |    80 |   70 |
|    4 | amy   | math    |    80 |   70 |
|    5 | mike  | math    |    70 |   70 |
|    1 | lily  | math    |  NULL |   70 |
+------+-------+---------+-------+------+
```

Пример 2: Использование LAST_VALUE() с типами ARRAY

Использование той же таблицы из Примера 2 FIRST_VALUE():

```
SELECT col_1, arr1,
    LAST_VALUE(arr1) OVER (
        ORDER BY col_1
        ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
    ) AS last_array
FROM test_array_value;
```

Вывод:

```
+-------+--------+-----------+
| col_1 | arr1   | last_array|
+-------+--------+-----------+
|     1 | [1,11] | [5,55]    |
|     2 | [2,22] | [5,55]    |
|     3 | [3,33] | [5,55]    |
|     4 | NULL   | [5,55]    |
|     5 | [5,55] | [5,55]    |
+-------+--------+-----------+
```

Последнее значение массива `[5,55]` возвращается для всех строк в окне.

LAG()

Возвращает значение строки, которая отстаёт от текущей строки на `offset` строк. Эта функция часто используется для сравнения значений между строками и фильтрации данных.

`LAG()` можно использовать для запроса данных следующих типов:

- Числовые: TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, DECIMAL
- Строковые: CHAR, VARCHAR
- Даты: DATE, DATETIME
- BITMAP и HLL поддерживаются начиная с Selena v1.5.2.
- Типы ARRAY поддерживаются начиная с Selena v1.5.2.

Синтаксис:

```
LAG(expr [IGNORE NULLS] [, offset[, default]])
OVER([<partition_by_clause>] [<order_by_clause>])
```

Параметры:

- `expr`: поле, которое вы хотите вычислить.
- `offset`: смещение. Должно быть **положительным целым числом**. Если этот параметр не указан, по умолчанию используется 1.
- `default`: значение по умолчанию, возвращаемое, если совпадающая строка не найдена. Если этот параметр не указан, по умолчанию используется NULL. `default` поддерживает любое выражение, тип которого совместим с `expr`.

- `IGNORE NULLS` поддерживается начиная с v1.5.2. Используется для определения, включаются ли значения NULL `expr` в результат. По умолчанию значения NULL включаются при подсчёте `offset` строк, что означает, что NULL возвращается, если значение целевой строки равно NULL. См. Пример 1. Если вы указываете IGNORE NULLS, значения NULL игнорируются при подсчёте `offset` строк, и система продолжает искать `offset` ненулевых значений. Если `offset` ненулевых значений не может быть найдено, возвращается NULL или `default` (если указано). См. Пример 2.

Пример 1: IGNORE NULLS не указан

Создание таблицы и вставка значений:

```
CREATE TABLE test_tbl (col_1 INT, col_2 INT)
DISTRIBUTED BY HASH(col_1);

INSERT INTO test_tbl VALUES
    (1, NULL),
    (2, 4),
    (3, NULL),
    (4, 2),
    (5, NULL),
    (6, 7),
    (7, 6),
    (8, 5),
    (9, NULL),
    (10, NULL);
```

Запрос данных из этой таблицы, где `offset` равен 2, что означает обход двух предыдущих строк; `default` равен 0, что означает, что 0 возвращается, если совпадающие строки не найдены.

Вывод:

```
SELECT col_1, col_2, LAG(col_2,2,0) OVER (ORDER BY col_1)
FROM test_tbl ORDER BY col_1;
+-------+-------+----------------------------------------------+
| col_1 | col_2 | lag(col_2, 2, 0) OVER (ORDER BY col_1 ASC ) |
+-------+-------+----------------------------------------------+
|     1 |  NULL |                                            0 |
|     2 |     4 |                                            0 |
|     3 |  NULL |                                         NULL |
|     4 |     2 |                                            4 |
|     5 |  NULL |                                         NULL |
|     6 |     7 |                                            2 |
|     7 |     6 |                                         NULL |
|     8 |     5 |                                            7 |
|     9 |  NULL |                                            6 |
|    10 |  NULL |                                            5 |
+-------+-------+----------------------------------------------+
```

Для первых двух строк предыдущие две строки не существуют, и возвращается значение по умолчанию 0.

Для NULL в строке 3 значение на две строки назад равно NULL, и возвращается NULL, потому что значения NULL разрешены.

Пример 2: IGNORE NULLS указан

Использование предыдущей таблицы и настроек параметров.

```
SELECT col_1, col_2, LAG(col_2 IGNORE NULLS,2,0) OVER (ORDER BY col_1)
FROM test_tbl ORDER BY col_1;
+-------+-------+---------------------------------------------+
| col_1 | col_2 | lag(col_2, 2, 0) OVER (ORDER BY col_1 ASC ) |
+-------+-------+---------------------------------------------+
|     1 |  NULL |                                           0 |
|     2 |     4 |                                           0 |
|     3 |  NULL |                                           0 |
|     4 |     2 |                                           0 |
|     5 |  NULL |                                           4 |
|     6 |     7 |                                           4 |
|     7 |     6 |                                           2 |
|     8 |     5 |                                           7 |
|     9 |  NULL |                                           6 |
|    10 |  NULL |                                           6 |
+-------+-------+---------------------------------------------+
```

Для строк с 1 по 4 система не может найти два ненулевых значения для каждой из них в предыдущих строках, и возвращается значение по умолчанию 0.

Для значения 6 в строке 7 значение на две строки назад равно NULL, и NULL игнорируется, потому что указан IGNORE NULLS. Система продолжает искать ненулевые значения, и возвращается 2 из строки 4.

Пример 3: Установка значения по умолчанию в LAG() равным имени столбца

Использование предыдущей таблицы и настроек параметров.

```
SELECT col_1, col_2, LAG(col_2 ,2,col_1) OVER (ORDER BY col_1)
FROM test_tbl ORDER BY col_1;
+-------+-------+-------------------------------------------------+
| col_1 | col_2 | lag(col_2, 2, col_1) OVER (ORDER BY col_1 ASC ) |
+-------+-------+-------------------------------------------------+
|     1 |  NULL |                                               1 |
|     2 |     4 |                                               2 |
|     3 |  NULL |                                            NULL |
|     4 |     2 |                                               4 |
|     5 |  NULL |                                            NULL |
|     6 |     7 |                                               2 |
|     7 |     6 |                                            NULL |
|     8 |     5 |                                               7 |
|     9 |  NULL |                                               6 |
|    10 |  NULL |                                               5 |
+-------+-------+-------------------------------------------------+
```

Как видите, для строк 1 и 2 нет двух ненулевых значений при сканировании назад, поэтому возвращается значение col_1 текущей строки по умолчанию.

Все остальные строки ведут себя так же, как в Примере 1.

Пример 4: Использование LAG() с типами ARRAY

Создание таблицы со столбцами ARRAY:

```
CREATE TABLE test_array_value (
    col_1 INT,
    arr1 ARRAY<INT>,
    arr2 ARRAY<INT> NOT NULL
) DISTRIBUTED BY HASH(col_1);

INSERT INTO test_array_value (col_1, arr1, arr2) VALUES
    (1, [1, 11], [101, 111]),
    (2, [2, 22], [102, 112]),
    (3, [3, 33], [103, 113]),
    (4, NULL,    [104, 114]),
    (5, [5, 55], [105, 115]),
    (6, [6, 66], [106, 116]);
```

Запрос данных с использованием LAG() с типами ARRAY:

```
SELECT col_1, arr1, LAG(arr1, 2, arr2) OVER (ORDER BY col_1) AS lag_result
FROM test_array_value;
```

Вывод:

```
+-------+--------+-------------+
| col_1 | arr1   | lag_result  |
+-------+--------+-------------+
|     1 | [1,11] | [101,111]   |
|     2 | [2,22] | [102,112]   |
|     3 | [3,33] | [1,11]      |
|     4 | NULL   | [2,22]      |
|     5 | [5,55] | [3,33]      |
|     6 | [6,66] | NULL        |
+-------+--------+-------------+
```

Для первых двух строк предыдущие две строки не существуют, поэтому возвращается значение по умолчанию из `arr2`.

LEAD()

Возвращает значение строки, которая опережает текущую строку на `offset` строк. Эта функция часто используется для сравнения значений между строками и фильтрации данных.

Типы данных, которые можно запрашивать с помощью `LEAD()`, такие же, как поддерживаемые LAG().

Синтаксис:

```
LEAD(expr [IGNORE NULLS] [, offset[, default]])
OVER([<partition_by_clause>] [<order_by_clause>])
```

Параметры:

- `expr`: поле, которое вы хотите вычислить.
- `offset`: смещение. Должно быть положительным целым числом. Если этот параметр не указан, по умолчанию используется 1.

- `default`: значение по умолчанию, возвращаемое, если совпадающая строка не найдена. Если этот параметр не указан, по умолчанию используется NULL. `default` поддерживает любое выражение, тип которого совместим с `expr`.
- `IGNORE NULLS` поддерживается начиная с v1.5.2. Используется для определения, включаются ли значения NULL `expr` в результат. По умолчанию значения NULL включаются при подсчёте `offset` строк, что означает, что NULL возвращается, если значение целевой строки равно NULL. См. Пример 1. Если вы указываете IGNORE NULLS, значения NULL игнорируются при подсчёте `offset` строк, и система продолжает искать `offset` ненулевых значений. Если `offset` ненулевых значений не может быть найдено, возвращается NULL или `default` (если указано). См. Пример 2.

Пример 1: IGNORE NULLS не указан

Создание таблицы и вставка значений:

```
CREATE TABLE test_tbl (col_1 INT, col_2 INT)
DISTRIBUTED BY HASH(col_1);

INSERT INTO test_tbl VALUES
    (1, NULL),
    (2, 4),
    (3, NULL),
    (4, 2),
    (5, NULL),
    (6, 7),
    (7, 6),
    (8, 5),
    (9, NULL),
    (10, NULL);
```

Запрос данных из этой таблицы, где `offset` равен 2, что означает обход двух последующих строк; `default` равен 0, что означает, что 0 возвращается, если совпадающие строки не найдены.

Вывод:

```
SELECT col_1, col_2, LEAD(col_2,2,0) OVER (ORDER BY col_1)
FROM test_tbl ORDER BY col_1;
+-------+-------+-----------------------------------------------+
| col_1 | col_2 | lead(col_2, 2, 0) OVER (ORDER BY col_1 ASC ) |
+-------+-------+-----------------------------------------------+
|     1 |  NULL |                                          NULL |
|     2 |     4 |                                             2 |
|     3 |  NULL |                                          NULL |
|     4 |     2 |                                             7 |
|     5 |  NULL |                                             6 |
|     6 |     7 |                                             5 |
|     7 |     6 |                                          NULL |
|     8 |     5 |                                          NULL |
|     9 |  NULL |                                             0 |
|    10 |  NULL |                                             0 |
+-------+-------+-----------------------------------------------+
```

Для первой строки значение на две строки вперёд равно NULL, и возвращается NULL, потому что значения NULL разрешены.

Для последних двух строк последующие две строки не существуют, и возвращается значение по умолчанию 0.

Пример 2: IGNORE NULLS указан

Использование предыдущей таблицы и настроек параметров.

```
SELECT col_1, col_2, LEAD(col_2 IGNORE NULLS,2,0) OVER (ORDER BY col_1)
FROM test_tbl ORDER BY col_1;
+-------+-------+-----------------------------------------------+
| col_1 | col_2 | lead(col_2, 2, 0) OVER (ORDER BY col_1 ASC ) |
+-------+-------+-----------------------------------------------+
|     1 |  NULL |                                             2 |
|     2 |     4 |                                             7 |
|     3 |  NULL |                                             7 |
|     4 |     2 |                                             6 |
|     5 |  NULL |                                             6 |
|     6 |     7 |                                             5 |
|     7 |     6 |                                             0 |
|     8 |     5 |                                             0 |
|     9 |  NULL |                                             0 |
|    10 |  NULL |                                             0 |
+-------+-------+-----------------------------------------------+
```

Для строк с 7 по 10 система не может найти два ненулевых значения в последующих строках, и возвращается значение по умолчанию 0.

Для первой строки значение на две строки вперёд равно NULL, и NULL игнорируется, потому что указан IGNORE NULLS. Система продолжает искать второе ненулевое значение, и возвращается 2 из строки 4.

Пример 3: Установка значения по умолчанию в LEAD() равным имени столбца

Использование предыдущей таблицы и настроек параметров.

```
SELECT col_1, col_2, LEAD(col_2 ,2,col_1) OVER (ORDER BY col_1)
FROM test_tbl ORDER BY col_1;
+-------+-------+-----------------------------------------------+
| col_1 | col_2 | lead(col_2, 2, col_1) OVER (ORDER BY col_1 ASC ) |
+-------+-------+-----------------------------------------------+
|     1 |  NULL |                                          NULL |
|     2 |     4 |                                             2 |
|     3 |  NULL |                                          NULL |
|     4 |     2 |                                             7 |
|     5 |  NULL |                                             6 |
|     6 |     7 |                                             5 |
|     7 |     6 |                                          NULL |
|     8 |     5 |                                          NULL |
|     9 |  NULL |                                             9 |
|    10 |  NULL |                                            10 |
+-------+-------+-----------------------------------------------+
```

Как видите, для строк 9 и 10 нет двух ненулевых значений при сканировании вперёд, поэтому возвращается значение col_1 текущей строки по умолчанию.

Все остальные строки ведут себя так же, как в Примере 1.

Пример 4: Использование LEAD() с типами ARRAY

Использование той же таблицы из Примера 4 LAG():

```
SELECT col_1, arr1, LEAD(arr1, 2, arr2) OVER (ORDER BY col_1) AS
lead_result
FROM test_array_value;
```

Вывод:

```
+-------+--------+-------------+
| col_1 | arr1   | lead_result |
+-------+--------+-------------+
|     1 | [1,11] | [3,33]      |
|     2 | [2,22] | NULL        |
|     3 | [3,33] | [5,55]      |
|     4 | NULL   | [6,66]      |
|     5 | [5,55] | [105,115]   |
|     6 | [6,66] | [106,116]   |
+-------+--------+-------------+
```

Для последних двух строк последующие две строки не существуют, поэтому возвращается значение по умолчанию из `arr2`.

MAX()

Возвращает максимальное значение указанных строк в текущем окне.

Синтаксис:

```
MAX(expr) [OVER (analytic_clause)]
```

Примеры:

Вычисление максимального значения строк от первой строки до строки после текущей строки. Этот пример использует данные из Примера таблицы `scores`.

```
select *,
    max(scores)
        over (
            partition by subject
            order by score
            rows between unbounded preceding and 1 following
        ) as max
from scores
where subject in ('math');
```

```
+------+-------+---------+-------+------+
| id   | name  | subject | score | max  |
+------+-------+---------+-------+------+
|    1 | lily  | math    |  NULL |   70 |
|    5 | mike  | math    |    70 |   80 |
|    2 | tom   | math    |    80 |   80 |
|    4 | amy   | math    |    80 |   92 |
|    6 | amber | math    |    92 |   95 |
|    3 | jack  | math    |    95 |   95 |
+------+-------+---------+-------+------+
```

Следующий пример вычисляет максимальную оценку среди всех строк для предмета `math`.

```
select *,
    max(score)
        over (
            partition by subject
            order by score
            rows between unbounded preceding and unbounded following
        ) as max
from scores
where subject in ('math');
```

Начиная с Selena 2.4, вы можете указать диапазон строк как `rows between n preceding and n following`, что означает, что вы можете захватить `n` строк до текущей строки и `n` строк после текущей строки.

Пример оператора:

```
select *,
    max(score)
        over (
            partition by subject
            order by score
            rows between 3 preceding and 2 following) as max
from scores
where subject in ('math');
```

MIN()

Возвращает минимальное значение указанных строк в текущем окне.

Синтаксис:

```
MIN(expr) [OVER (analytic_clause)]
```

Примеры:

Вычисление наименьшей оценки среди всех строк для предмета математика. Этот пример использует данные из Примера таблицы `scores`.

```
select *,
    min(score)
        over (
            partition by subject
            order by score
            rows between unbounded preceding and unbounded following)
            as min
from scores
where subject in ('math');
```

```
+------+-------+---------+-------+------+
| id   | name  | subject | score | min  |
+------+-------+---------+-------+------+
|    1 | lily  | math    |  NULL |   70 |
|    5 | mike  | math    |    70 |   70 |
|    2 | tom   | math    |    80 |   70 |
|    4 | amy   | math    |    80 |   70 |
|    6 | amber | math    |    92 |   70 |
|    3 | jack  | math    |    95 |   70 |
+------+-------+---------+-------+------+
```

Начиная с Selena 2.4, вы можете указать диапазон строк как `rows between n preceding and n following`, что означает, что вы можете захватить `n` строк до текущей строки и `n` строк после текущей строки.

Пример оператора:

```
select *,
    min(score)
        over (
            partition by subject
            order by score
            rows between 3 preceding and 2 following) as max
from scores
where subject in ('math');
```

NTILE()

Функция NTILE() делит отсортированные строки в разделе на указанное количество `num_buckets` как можно более равномерно, сохраняет разделённые строки в соответствующих корзинах, начиная с 1 `[1, 2, ..., num_buckets]`, и возвращает номер корзины, в которой находится каждая строка.

О размере корзины:

- Если количество строк делится на указанное число `num_buckets` ровно, все корзины будут одинакового размера.
- Если количество строк не делится на указанное число `num_buckets` ровно, будут корзины двух разных размеров. Разница между размерами равна 1. Корзины с большим количеством строк будут перечислены перед корзинами с меньшим количеством строк.

Синтаксис:

```
NTILE (num_buckets) OVER (partition_by_clause order_by_clause)
```

`num_buckets`: Количество корзин для создания. Значение должно быть константным положительным целым числом, максимум `2^63 - 1`.

Предложение window не допускается в функции NTILE().

Функция NTILE() возвращает данные типа BIGINT.

Примеры:

Следующий пример делит все строки в разделе на две корзины. Этот пример использует данные из Примера таблицы `scores`.

```
select *,
    ntile(2)
        over (
            partition by subject
            order by score
        ) as bucket_id
from scores;
```

Вывод:

```
+------+-------+---------+-------+-----------+
| id   | name  | subject | score | bucket_id |
+------+-------+---------+-------+-----------+
|    6 | amber | NULL    |    90 |         1 |
|    1 | lily  | math    |  NULL |         1 |
|    5 | mike  | math    |    70 |         1 |
|    2 | tom   | math    |    80 |         1 |
|    4 | amy   | math    |    80 |         2 |
|    6 | amber | math    |    92 |         2 |
|    3 | jack  | math    |    95 |         2 |
|    3 | jack  | english |  NULL |         1 |
|    5 | mike  | english |    85 |         1 |
|    4 | amy   | english |    92 |         1 |
|    2 | tom   | english |    98 |         2 |
|    1 | lily  | english |   100 |         2 |
|    2 | tom   | physics |  NULL |         1 |
|    1 | lily  | physics |    60 |         1 |
|    5 | mike  | physics |    85 |         1 |
|    3 | jack  | physics |    99 |         2 |
|    4 | amy   | physics |    99 |         2 |
|    6 | amber | physics |   100 |         2 |
+------+-------+---------+-------+-----------+
```

Как показано в приведённом выше примере, когда `num_buckets` равно `2`:

- Для первой строки этот раздел имеет только эту запись, и она назначается только в одну корзину.

- Для строк со 2 по 7 раздел имеет 6 записей, и первые 3 записи назначаются в корзину 1, а другие 3 записи назначаются в корзину 2.

PERCENT_RANK()

Вычисляет относительный ранг строки в результирующем наборе как процент.

PERCENT_RANK() вычисляется по следующей формуле, где `Rank` представляет ранг текущей строки в разделе.

```
(Rank - 1)/(Rows in partition - 1)
```

Возвращаемые значения находятся в диапазоне от 0 до 1. Эта функция полезна для вычисления процентилей и анализа распределения данных. Поддерживается начиная с v1.5.2.

Синтаксис:

```
PERCENT_RANK() OVER (partition_by_clause order_by_clause)
```

Эта функция должна использоваться с ORDER BY для сортировки строк раздела в желаемом порядке.

Примеры:

Следующий пример показывает относительный ранг каждой `score` в группе `math`. Этот пример использует данные из Примера таблицы `scores`.

```
SELECT *,
    PERCENT_RANK()
        OVER (
            PARTITION BY subject
            ORDER BY score
        ) AS `percent_rank`
FROM scores where subject in ('math');
```

```
+------+-------+---------+-------+--------------+
| id   | name  | subject | score | percent_rank |
+------+-------+---------+-------+--------------+
|    1 | lily  | math    |  NULL |            0 |
|    5 | mike  | math    |    70 |          0.2 |
|    2 | tom   | math    |    80 |          0.4 |
|    4 | amy   | math    |    80 |          0.4 |
|    6 | amber | math    |    92 |          0.8 |
|    3 | jack  | math    |    95 |            1 |
+------+-------+---------+-------+--------------+
```

RANK()

Функция RANK() используется для представления рангов. В отличие от DENSE_RANK(), RANK() **имеет пропуски** в номерах. Например, если появляются две 1, третий номер RANK() будет 3 вместо 2.

Синтаксис:

```
RANK() OVER(partition_by_clause order_by_clause)
```

Примеры:

Ранжирование оценок по математике в группе. Этот пример использует данные из Примера таблицы `scores`.

```
select *,
    rank() over(
        partition by subject
        order by score desc
        ) as `rank`
from scores where subject in ('math');
```

```
+------+-------+---------+-------+------+
| id   | name  | subject | score | rank |
+------+-------+---------+-------+------+
|    3 | jack  | math    |    95 |    1 |
|    6 | amber | math    |    92 |    2 |
|    4 | amy   | math    |    80 |    3 |
|    2 | tom   | math    |    80 |    3 |
|    5 | mike  | math    |    70 |    5 |
|    1 | lily  | math    |  NULL |    6 |
+------+-------+---------+-------+------+
```

Результирующие данные имеют две строки с оценкой 80. Они все имеют ранг 3. Ранг для следующей оценки 70 равен 5.

ROW_NUMBER()

Возвращает непрерывно увеличивающееся целое число, начиная с 1, для каждой строки раздела. В отличие от RANK() и DENSE_RANK(), значение, возвращаемое ROW_NUMBER(), **не повторяется и не имеет пропусков** и **непрерывно увеличивается**.

Синтаксис:

```
ROW_NUMBER() OVER(partition_by_clause order_by_clause)
```

Примеры:

Ранжирование оценок по математике в группе. Этот пример использует данные из Примера таблицы `scores`.

```
select *, row_number() over(
    partition by subject
    order by score desc) as `rank`
from scores where subject in ('math');
```

```
+------+-------+---------+-------+------+
| id   | name  | subject | score | rank |
+------+-------+---------+-------+------+
|    3 | jack  | math    |    95 |    1 |
|    6 | amber | math    |    92 |    2 |
|    2 | tom   | math    |    80 |    3 |
|    4 | amy   | math    |    80 |    4 |
|    5 | mike  | math    |    70 |    5 |
|    1 | lily  | math    |  NULL |    6 |
+------+-------+---------+-------+------+
```

QUALIFY()

Предложение QUALIFY фильтрует результаты оконных функций. В операторе SELECT вы можете использовать предложение QUALIFY для применения условий к столбцу для фильтрации результатов. QUALIFY аналогично предложению HAVING в агрегатных функциях. Эта функция поддерживается начиная с v1.5.2.

QUALIFY упрощает написание операторов SELECT.

До использования QUALIFY оператор SELECT мог выглядеть так:

```
SELECT *
FROM (SELECT DATE,
             PROVINCE_CODE,
             TOTAL_SCORE,
             ROW_NUMBER() OVER(PARTITION BY PROVINCE_CODE ORDER BY
TOTAL_SCORE) AS SCORE_ROWNUMBER
      FROM example_table) T1
WHERE T1.SCORE_ROWNUMBER = 1;
```

После использования QUALIFY оператор сокращается до:

```
SELECT DATE, PROVINCE_CODE, TOTAL_SCORE
FROM example_table
QUALIFY ROW_NUMBER() OVER(PARTITION BY PROVINCE_CODE ORDER BY TOTAL_SCORE)
= 1;
```

QUALIFY поддерживает только следующие три оконные функции: ROW_NUMBER(), RANK() и DENSE_RANK().

Синтаксис:

```
SELECT <column_list>
FROM <data_source>
[GROUP BY ...]
[HAVING ...]
QUALIFY <window_function>
[ ... ]
```

Параметры:

`<column_list>`: столбцы, из которых вы хотите получить данные.

`<data_source>`: Источник данных обычно является таблицей.

`<window_function>`: За предложением `QUALIFY` может следовать только оконная функция, включая ROW_NUMBER(), RANK() и DENSE_RANK().

Примеры:

```
-- Создание таблицы.
CREATE TABLE sales_record (
   city_id INT,
   item STRING,
   sales INT
) DISTRIBUTED BY HASH(`city_id`);

-- Вставка данных в таблицу.
insert into sales_record values
(1,'fruit',95),
(2,'drinks',70),
(3,'fruit',87),
(4,'drinks',98);

-- Запрос данных из таблицы.
select * from sales_record order by city_id;
+---------+--------+-------+
| city_id | item   | sales |
+---------+--------+-------+
|       1 | fruit  |    95 |
|       2 | drinks |    70 |
|       3 | fruit  |    87 |
|       4 | drinks |    98 |
+---------+--------+-------+
```

Пример 1: Получение записей, номер строки которых больше 1, из таблицы.

```
SELECT city_id, item, sales
FROM sales_record
QUALIFY row_number() OVER (ORDER BY city_id) > 1;
+---------+--------+-------+
| city_id | item   | sales |
+---------+--------+-------+
|       2 | drinks |    70 |
|       3 | fruit  |    87 |
|       4 | drinks |    98 |
+---------+--------+-------+
```

Пример 2: Получение записей, номер строки которых равен 1, из каждого раздела таблицы. Таблица разделена на два раздела по `item`, и возвращается первая строка в каждом разделе.

```
SELECT city_id, item, sales
FROM sales_record
QUALIFY ROW_NUMBER() OVER (PARTITION BY item ORDER BY city_id) = 1
ORDER BY city_id;
+---------+--------+-------+
| city_id | item   | sales |
+---------+--------+-------+
|       1 | fruit  |    95 |
|       2 | drinks |    70 |
+---------+--------+-------+
2 rows in set (0.01 sec)
```

Пример 3: Получение записей с рангом продаж №1 из каждого раздела таблицы. Таблица разделена на два раздела по `item`, и возвращается строка с наивысшими продажами в каждом разделе.

```
SELECT city_id, item, sales
FROM sales_record
QUALIFY RANK() OVER (PARTITION BY item ORDER BY sales DESC) = 1
ORDER BY city_id;
+---------+--------+-------+
| city_id | item   | sales |
+---------+--------+-------+
|       1 | fruit  |    95 |
|       4 | drinks |    98 |
+---------+--------+-------+
```

Примечания по использованию:

- QUALIFY supports only the following three window functions: ROW_NUMBER(), RANK(), and DENSE_RANK().
- The execution order of clauses in a query with QUALIFY is evaluated in the following order:

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. Window
6. QUALIFY

7. DISTINCT
8. ORDER BY
9. LIMIT

SUM()

Calculates the sum of specified rows.

Syntax:

```
SUM([DISTINCT] expr) [OVER (analytic_clause)]
```

`DISTINCT` is supported from Selena v2.0.0. When specified, SUM() sums only distinct values in the window.

> **ПРИМЕЧАНИЕ**
>
> **Window frame limitation:** When using SUM(DISTINCT) as a window function, only RANGE frames are supported. ROWS frames are NOT supported.

Examples:

Group data by `subject` and calculate the sum of scores of all rows within the group. This example uses the data in the Sample table `scores`.

```
select *,
    sum(score)
        over (
            partition by subject
            order by score
            rows between unbounded preceding and unbounded following
        ) as 'sum'
from scores;
```

```
+------+-------+---------+-------+------+
| id   | name  | subject | score | sum  |
+------+-------+---------+-------+------+
|    6 | amber | NULL    |    90 |   90 |
|    1 | lily  | math    |  NULL |  417 |
|    5 | mike  | math    |    70 |  417 |
|    2 | tom   | math    |    80 |  417 |
|    4 | amy   | math    |    80 |  417 |
|    6 | amber | math    |    92 |  417 |
|    3 | jack  | math    |    95 |  417 |
|    3 | jack  | english |  NULL |  375 |
|    5 | mike  | english |    85 |  375 |
|    4 | amy   | english |    92 |  375 |
|    2 | tom   | english |    98 |  375 |
|    1 | lily  | english |   100 |  375 |
|    2 | tom   | physics |  NULL |  443 |
|    1 | lily  | physics |    60 |  443 |
|    5 | mike  | physics |    85 |  443 |
|    3 | jack  | physics |    99 |  443 |
|    4 | amy   | physics |    99 |  443 |
|    6 | amber | physics |   100 |  443 |
+------+-------+---------+-------+------+
```

Example 2: Using SUM(DISTINCT) over overall window

Sum distinct scores across all rows:

```
SELECT id, subject, score,
    SUM(DISTINCT score) OVER () AS distinct_sum
FROM test_scores;
```

Вывод:

```
+----+---------+-------+-------------+
| id | subject | score | distinct_sum|
+----+---------+-------+-------------+
|  1 | math    |    80 |          255|
|  2 | math    |    85 |          255|
|  3 | math    |    80 |          255|
|  4 | english |    90 |          255|
|  5 | english |    85 |          255|
|  6 | english |    90 |          255|
+----+---------+-------+-------------+
```

The distinct sum is 255 (80 + 85 + 90).

Example 3: Using SUM(DISTINCT) over framed window with RANGE frame

Sum distinct scores within each subject partition using a RANGE frame:

```
SELECT id, subject, score,
    SUM(DISTINCT score) OVER (
        PARTITION BY subject
        ORDER BY score
        RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    ) AS distinct_sum
FROM test_scores;
```

Вывод:

```
+----+---------+-------+-------------+
| id | subject | score | distinct_sum|
+----+---------+-------+-------------+
|  1 | math    |    80 |           80|
|  3 | math    |    80 |           80|
|  2 | math    |    85 |          165|
|  5 | english |    85 |           85|
|  4 | english |    90 |          175|
|  6 | english |    90 |          175|
+----+---------+-------+-------------+
```

For each row, the function sums distinct scores from the beginning of the partition up to and including the current row's score value.

VARIANCE, VAR_POP, VARIANCE_POP

Returns the population variance of an expression. VAR_POP and VARIANCE_POP are aliases of VARIANCE. These functions can be used as window functions since v1.5.2.

Syntax:

```
VARIANCE(expr) OVER([partition_by_clause] [order_by_clause]
[order_by_clause window_clause])
```

> **ПОДСКАЗКА**
>
> From 2.5.13, 3.0.7, 3.1.4 onwards, this window function supports the ORDER BY and Window clauses.

Parameters:

If `expr` is a table column, it must evaluate to TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, or DECIMAL.

Examples:

This example uses the data in the Sample table `scores`.

```
select *,
    variance(score)
        over (
            partition by subject
            order by score
        ) as 'variance'
from scores where subject in ('math');
+------+-------+---------+-------+--------------------+
| id   | name  | subject | score | variance           |
+------+-------+---------+-------+--------------------+
|    1 | lily  | math    |  NULL |               NULL |
|    5 | mike  | math    |    70 |                  0 |
|    2 | tom   | math    |    80 | 22.222222222222225 |
|    4 | amy   | math    |    80 | 22.222222222222225 |
|    6 | amber | math    |    92 |  60.74999999999997 |
|    3 | jack  | math    |    95 |  82.23999999999998 |
+------+-------+---------+-------+--------------------+
```

VAR_SAMP, VARIANCE_SAMP

Returns the sample variance of an expression. These functions can be used as window functions since v1.5.2.

Syntax:

```
VAR_SAMP(expr) OVER([partition_by_clause] [order_by_clause]
[order_by_clause window_clause])
```

> **ПОДСКАЗКА**
>
> From 2.5.13, 3.0.7, 3.1.4 onwards, this window function supports the ORDER BY and Window clauses.

Parameters:

If `expr` is a table column, it must evaluate to TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, or DECIMAL.

Examples:

This example uses the data in the Sample table `scores`.

```
select *,
    VAR_SAMP(score)
       over (partition by subject
            order by score) as VAR_SAMP
from scores where subject in ('math');
+------+-------+---------+-------+--------------------+
| id   | name  | subject | score | VAR_SAMP           |
+------+-------+---------+-------+--------------------+
|    1 | lily  | math    |  NULL |               NULL |
|    5 | mike  | math    |    70 |                  0 |
|    2 | tom   | math    |    80 | 33.333333333333336 |
|    4 | amy   | math    |    80 | 33.333333333333336 |
|    6 | amber | math    |    92 |  80.99999999999996 |
|    3 | jack  | math    |    95 | 102.79999999999997 |
+------+-------+---------+-------+--------------------+
```

STD, STDDEV, STDDEV_POP

Returns the standard deviation of an expression. These functions can be used as window functions since v1.5.2.

Syntax:

```
STD(expr) OVER([partition_by_clause] [order_by_clause] [order_by_clause
window_clause])
```

> **ПОДСКАЗКА**
>
> From 2.5.13, 3.0.7, 3.1.4 onwards, this window function supports the ORDER BY and Window clauses.

Parameters:

If `expr` is a table column, it must evaluate to TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, or DECIMAL.

Examples:

This example uses the data in the Sample table `scores`.

```
select *, STD(score)
    over (
        partition by subject
        order by score) as std
from scores where subject in ('math');
+------+-------+---------+-------+-------------------+
| id   | name  | subject | score | std               |
+------+-------+---------+-------+-------------------+
|    1 | lily  | math    |  NULL |              NULL |
|    5 | mike  | math    |    70 |                 0 |
|    4 | amy   | math    |    80 | 4.714045207910317 |
|    2 | tom   | math    |    80 | 4.714045207910317 |
|    6 | amber | math    |    92 | 7.794228634059946 |
|    3 | jack  | math    |    95 | 9.068627239003707 |
+------+-------+---------+-------+-------------------+
```

STDDEV_SAMP

Returns the sample standard deviation of an expression. This function can be used as a window function since v1.5.2.

Syntax:

```
STDDEV_SAMP(expr) OVER([partition_by_clause] [order_by_clause]
[order_by_clause window_clause])
```

> **ПОДСКАЗКА**
>
> From 2.5.13, 3.0.7, 3.1.4 onwards, this window function supports the ORDER BY and Window clauses.

Parameters:

If `expr` is a table column, it must evaluate to TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, or DECIMAL.

Examples:

This example uses the data in the Sample table `scores`.

```
select *, STDDEV_SAMP(score)
    over (
        partition by subject
        order by score
        ) as STDDEV_SAMP
from scores where subject in ('math');
+------+-------+---------+-------+--------------------+
| id   | name  | subject | score | STDDEV_SAMP        |
+------+-------+---------+-------+--------------------+
|    1 | lily  | math    |  NULL |               NULL |
|    5 | mike  | math    |    70 |                  0 |
|    2 | tom   | math    |    80 |  5.773502691896258 |
|    4 | amy   | math    |    80 |  5.773502691896258 |
|    6 | amber | math    |    92 |  8.999999999999998 |
|    3 | jack  | math    |    95 | 10.139033484509259 |
+------+-------+---------+-------+--------------------+

select *, STDDEV_SAMP(score)
    over (
        partition by subject
        order by score
        rows between unbounded preceding and 1 following) as STDDEV_SAMP
from scores where subject in ('math');
+------+-------+---------+-------+--------------------+
| id   | name  | subject | score | STDDEV_SAMP        |
+------+-------+---------+-------+--------------------+
|    1 | lily  | math    |  NULL |                  0 |
|    5 | mike  | math    |    70 | 7.0710678118654755 |
|    2 | tom   | math    |    80 |  5.773502691896258 |
|    4 | amy   | math    |    80 |  8.999999999999998 |
|    6 | amber | math    |    92 | 10.139033484509259 |
|    3 | jack  | math    |    95 | 10.139033484509259 |
+------+-------+---------+-------+--------------------+
```

COVAR_SAMP

Returns the sample covariance of two expressions. This function is supported from v1.5.2. It is also an aggregate function.

Syntax:

```
COVAR_SAMP(expr1,expr2) OVER([partition_by_clause] [order_by_clause]
[order_by_clause window_clause])
```

ПОДСКАЗКА

From 2.5.13, 3.0.7, 3.1.4 onwards, this window function supports the ORDER BY and Window clauses.

Parameters:

If `expr` is a table column, it must evaluate to TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, or DECIMAL.

Examples:

This example uses the data in the Sample table `scores`.

```
select *, COVAR_SAMP(id, score)
    over (
        partition by subject
        order by score) as covar_samp
from scores where subject in ('math');
+------+-------+---------+-------+----------------------+
| id   | name  | subject | score | covar_samp           |
+------+-------+---------+-------+----------------------+
|    1 | lily  | math    |  NULL |                 NULL |
|    5 | mike  | math    |    70 |                    0 |
|    2 | tom   | math    |    80 |   -6.666666666666668 |
|    4 | amy   | math    |    80 |   -6.666666666666668 |
|    6 | amber | math    |    92 |                  4.5 |
|    3 | jack  | math    |    95 | -0.24999999999999822 |
+------+-------+---------+-------+----------------------+

select *, COVAR_SAMP(id,score)
    over (
        partition by subject
        order by score
        rows between unbounded preceding and 1 following) as COVAR_SAMP
from scores where subject in ('math');
+------+-------+---------+-------+----------------------+
| id   | name  | subject | score | COVAR_SAMP           |
+------+-------+---------+-------+----------------------+
|    1 | lily  | math    |  NULL |                    0 |
|    5 | mike  | math    |    70 |                   -5 |
|    4 | amy   | math    |    80 |   -6.666666666666661 |
|    2 | tom   | math    |    80 |    4.500000000000004 |
|    6 | amber | math    |    92 | -0.24999999999999467 |
|    3 | jack  | math    |    95 | -0.24999999999999467 |
```

COVAR_POP

Returns the population covariance of two expressions. This function is supported from v1.5.2. It is also an aggregate function.

Syntax:

```
COVAR_POP(expr1, expr2) OVER([partition_by_clause] [order_by_clause]
[order_by_clause window_clause])
```

> **ПОДСКАЗКА**
>
> From 2.5.13, 3.0.7, 3.1.4 onwards, this window function supports the ORDER BY and Window clauses.

Parameters:

If `expr` is a table column, it must evaluate to TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, or DECIMAL.

Examples:

This example uses the data in the Sample table `scores`.

```
select *, COVAR_POP(id, score)
    over (
        partition by subject
        order by score) as covar_pop
from scores where subject in ('math');
+------+-------+---------+-------+----------------------+
| id   | name  | subject | score | covar_pop            |
+------+-------+---------+-------+----------------------+
|    1 | lily  | math    |  NULL |                 NULL |
|    5 | mike  | math    |    70 |                    0 |
|    2 | tom   | math    |    80 |  -4.4444444444444455 |
|    4 | amy   | math    |    80 |  -4.4444444444444455 |
|    6 | amber | math    |    92 |                3.375 |
|    3 | jack  | math    |    95 | -0.19999999999999857 |
+------+-------+---------+-------+----------------------+
```

CORR

Returns the Pearson correlation coefficient between two expressions. This function is supported from v1.5.2. It is also an aggregate function.

Syntax:

```
CORR(expr1, expr2) OVER([partition_by_clause] [order_by_clause]
[order_by_clause window_clause])
```

ПОДСКАЗКА

From 2.5.13, 3.0.7, 3.1.4 onwards, this window function supports the ORDER BY and Window clauses.

Parameters:

If `expr` is a table column, it must evaluate to TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, or DECIMAL.

Examples:

This example uses the data in the Sample table `scores`.

```
select *, CORR(id, score)
    over (
        partition by subject
        order by score) as corr
from scores where subject in ('math');
+------+-------+---------+-------+-----------------------+
| id   | name  | subject | score | corr                  |
+------+-------+---------+-------+-----------------------+
|    5 | mike  | math    |    70 | -0.015594571538795355 |
|    1 | lily  | math    |  NULL | -0.015594571538795355 |
|    2 | tom   | math    |    80 | -0.015594571538795355 |
|    4 | amy   | math    |    80 | -0.015594571538795355 |
|    3 | jack  | math    |    95 | -0.015594571538795355 |
|    6 | amber | math    |    92 | -0.015594571538795355 |
+------+-------+---------+-------+-----------------------+

select *, CORR(id,score)
    over (
        partition by subject
        order by score
        rows between unbounded preceding and 1 following) as corr
from scores where subject in ('math');
+------+-------+---------+-------+-------------------------+
| id   | name  | subject | score | corr                    |
+------+-------+---------+-------+-------------------------+
|    1 | lily  | math    |  NULL | 1.7976931348623157e+308 |
|    5 | mike  | math    |    70 |                      -1 |
|    2 | tom   | math    |    80 |     -0.7559289460184546 |
|    4 | amy   | math    |    80 |     0.29277002188455997 |
|    6 | amber | math    |    92 |   -0.015594571538795024 |
|    3 | jack  | math    |    95 |   -0.015594571538795024 |
+------+-------+---------+-------+-------------------------+
```

2.4 CAST

Преобразует входные данные в указанный тип. Например, `cast (input as BIGINT)` преобразует входные данные в значение типа BIGINT.

С версии v1.5.2 Selena поддерживает преобразование в тип ARRAY.

Синтаксис

```
cast (input as type)
```

Параметры

`input`: данные, которые вы хотите преобразовать. `type`: целевой тип данных.

Возвращаемое значение

Возвращает значение, тип данных которого совпадает с `type`.

Примеры

Пример 1: обычные преобразования данных

```
select cast('9.5' as DECIMAL(10,2));
+--------------------------------+
| CAST('9.5' AS DECIMAL64(10,2)) |
+--------------------------------+
|                           9.50 |
+--------------------------------+

select cast(NULL AS INT);
+-------------------+
| CAST(NULL AS INT) |
+-------------------+
|              NULL |
+-------------------+

select cast(true AS BOOLEAN);
+-----------------------+
| CAST(TRUE AS BOOLEAN) |
+-----------------------+
|                     1 |
+-----------------------+

select cast (1 as BIGINT);
+-------------------+
| CAST(1 AS BIGINT) |
+-------------------+
|                 1 |
+-------------------+
```

Пример 2: Преобразование входных данных в ARRAY.

```
-- Преобразование строки в ARRAY<ANY>.

select cast('[1,2,3]' as array<int>);
+-------------------------------+
| CAST('[1,2,3]' AS ARRAY<INT>) |
+-------------------------------+
| [1,2,3]                       |
+-------------------------------+

select cast('[1,2,3]' as array<bigint>);
+----------------------------------+
| CAST('[1,2,3]' AS ARRAY<BIGINT>) |
+----------------------------------+
| [1,2,3]                          |
+----------------------------------+

select cast('[1,2,3]' as array<string>);
+-------------------------------------------+
| CAST('[1,2,3]' AS ARRAY<VARCHAR(65533)>) |
+-------------------------------------------+
| ["1","2","3"]                             |
+-------------------------------------------+

select cast('["a", "b", "c"]' as array<string>);
+----------------------------------------------------+
| CAST('["a", "b", "c"]' AS ARRAY<VARCHAR(65533)>) |
+----------------------------------------------------+
| ["a","b","c"]                                      |
+----------------------------------------------------+

-- Преобразование JSON-массива в ARRAY<ANY>.

select cast(parse_json('[{"a":1}, {"a": 2}]')  as array<json>);
+-------------------------------------------------------------+
| CAST((parse_json('[{"a":1}, {"a": 2}]')) AS ARRAY<JSON>) |
+-------------------------------------------------------------+
| ['{"a": 1}','{"a": 2}']                                     |
+-------------------------------------------------------------+

select cast(parse_json('[1, 2, 3]')  as array<int>);
+-----------------------------------------------+
| CAST((parse_json('[1, 2, 3]')) AS ARRAY<INT>) |
+-----------------------------------------------+
| [1,2,3]                                       |
+-----------------------------------------------+
```

```
select cast(parse_json('["1","2","3"]') as array<string>);
+-----------------------------------------------------------------+
| CAST((parse_json('["1","2","3"]')) AS ARRAY<VARCHAR(65533)>) |
+-----------------------------------------------------------------+
| ["1","2","3"]                                                   |
+-----------------------------------------------------------------+
```

Пример 3: Преобразование данных во время загрузки.

```
curl --location-trusted -u <username>:<password> -T ~/user_data/bigint \
    -H "columns: tmp_k1, k1=cast(tmp_k1 as BIGINT)" \
    http://host:port/api/test/bigint/_stream_load
```

> **Примечание**
>
> Если исходное значение является значением с плавающей точкой (например, 12.0), оно будет преобразовано в NULL. Если вы хотите принудительно преобразовать этот тип в BIGINT, см. следующий пример:

```
curl --location-trusted -u <username>:<password> -T ~/user_data/bigint \
    -H "columns: tmp_k1, k1=cast(cast(tmp_k1 as DOUBLE) as BIGINT)" \
    http://host:port/api/test/bigint/_stream_load
```

```
MySQL > select cast(cast ("11.2" as double) as bigint);
+----------------------------------------+
| CAST(CAST('11.2' AS DOUBLE) AS BIGINT) |
+----------------------------------------+
|                                     11 |
+----------------------------------------+
```

2.5 Hive Bitmap UDF

Hive Bitmap UDF предоставляет UDF-функции, которые можно напрямую использовать в Hive. Они могут использоваться для генерации данных Bitmap и выполнения вычислений, связанных с Bitmap.

Формат Bitmap, определённый Hive Bitmap UDF, соответствует формату в Selena и может напрямую использоваться для загрузки данных Bitmap в Selena и выгрузки данных Bitmap из Selena в Hive.

Применимые сценарии:

- Объём исходных данных велик, и прямая загрузка этих данных в Selena для вычислений создаст огромную нагрузку на cluster Selena. Желаемое решение - генерировать данные Bitmap в Hive, а затем загружать Bitmap в Selena.
- Экспорт данных Bitmap, сгенерированных в Selena, в Hive для использования другими системами.

Поддерживаемые типы исходных и целевых данных:

- Версия 3.1 и более поздние поддерживают загрузку и выгрузку данных этих типов: String, Base64 и Binary.
- Версии 2.5 и 3.0 поддерживают только загрузку и выгрузку данных String и Base64.

UDF-функции Hive Bitmap, которые можно генерировать

- com.selena.hive.udf.UDAFBitmapAgg

  Объединяет несколько строк ненулевых значений в столбце в одну строку значений Bitmap, что эквивалентно встроенной агрегатной функции Selena bitmap_agg.

- com.selena.hive.udf.UDAFBitmapUnion

  Вычисляет объединение набора bitmap, что эквивалентно встроенной агрегатной функции Selena bitmap_union.

- com.selena.hive.udf.UDFBase64ToBitmap

  Преобразует строку в кодировке base64 в bitmap, что эквивалентно встроенной функции Selena base64_to_bitmap.

- com.selena.hive.udf.UDFBitmapAnd

  Вычисляет пересечение двух bitmap, что эквивалентно встроенной функции Selena bitmap_and.

- com.selena.hive.udf.UDFBitmapCount

  Подсчитывает количество значений в bitmap, что эквивалентно встроенной функции Selena bitmap_count.

- com.selena.hive.udf.UDFBitmapFromString

  Преобразует строку, разделённую запятыми, в bitmap, что эквивалентно встроенной функции Selena bitmap_from_string.

- com.selena.hive.udf.UDFBitmapOr

  Вычисляет объединение двух bitmap, что эквивалентно встроенной функции Selena bitmap_or.

- com.selena.hive.udf.UDFBitmapToBase64

  Преобразует Bitmap в строку Base64, что эквивалентно встроенной функции Selena bitmap_to_base64.

- com.selena.hive.udf.UDFBitmapToString

  Преобразует bitmap в строку, разделённую запятыми, что эквивалентно встроенной функции Selena bitmap_to_string.

- com.selena.hive.udf.UDFBitmapXor

  Вычисляет набор уникальных элементов в двух Bitmap, что эквивалентно встроенной функции Selena bitmap_xor.

Как использовать

1. Скомпилируйте и сгенерируйте Hive UDF на FE.

   ```
   ./build.sh --hive-udf
   ```

   JAR-пакет `hive-udf-1.0.0.jar` будет сгенерирован в директории `fe/hive-udf/`.

2. Загрузите JAR-пакет в HDFS.

```
hadoop  fs -put -f ./hive-udf-1.0.0.jar hdfs://<hdfs_ip>:
<hdfs_port>/hive-udf-1.0.0.jar
```

3. Загрузите JAR-пакет в Hive.

```
hive> add jar hdfs://<hdfs_ip>:<hdfs_port>/hive-udf-1.0.0.jar;
```

4. Загрузите UDF-функции.

```
hive> create temporary function bitmap_agg as
'com.selena.hive.udf.UDAFBitmapAgg';
hive> create temporary function bitmap_union as
'com.selena.hive.udf.UDAFBitmapUnion';
hive> create temporary function base64_to_bitmap as
'com.selena.hive.udf.UDFBase64ToBitmap';
hive> create temporary function bitmap_and as
'com.selena.hive.udf.UDFBitmapAnd';
hive> create temporary function bitmap_count as
'com.selena.hive.udf.UDFBitmapCount';
hive> create temporary function bitmap_from_string as
'com.selena.hive.udf.UDFBitmapFromString';
hive> create temporary function bitmap_or as
'com.selena.hive.udf.UDFBitmapOr';
hive> create temporary function bitmap_to_base64 as
'com.selena.hive.udf.UDFBitmapToBase64';
hive> create temporary function bitmap_to_string as
'com.selena.hive.udf.UDFBitmapToString';
hive> create temporary function bitmap_xor as
'com.selena.hive.udf.UDFBitmapXor';
```

Примеры использования

Генерация Bitmap в Hive и загрузка в Selena в формате Binary

1. Создайте исходную таблицу Hive.

```
hive> create table t_src(c1 bigint, c2 bigint) stored as parquet;

hive> insert into t_src values (1,1), (1,2), (1,3), (2,4), (2,5);

hive> select * from t_src;
1       1
1       2
1       3
2       4
2       5
```

2. Создайте таблицу bitmap в Hive.

```
hive> create table t_bitmap(c1 bigint, c2 binary) stored as parquet;
```

Hive генерирует bitmap через UDFBitmapAgg и записывает его в таблицу bitmap.

```
hive> insert into t_bitmap select c1, bitmap_agg(c2) from t_src group
by c1;
```

3. Создайте таблицу Bitmap в Selena.

```
mysql> create table t1(c1 int, c2 bitmap bitmap_union) aggregate
key(c1)  distributed by hash(c1);
```

4. Загрузите данные Bitmap в Selena разными способами.

   - Загрузка данных через функцию files.

```
mysql> insert into t1 select c1, bitmap_from_binary(c2) from files (
    "path" = "hdfs://<hdfs_ip>:<hdfs_port>/<hdfs_db>/t_bitmap/*",
    "format"="parquet",
    "compression" = "uncompressed"
    );
```

   - Загрузка данных через Hive Catalog.

```
mysql> insert into t1 select c1, bitmap_from_binary(c2) from
hive_catalog_hms.xxx_db.t_bitmap;
```

5. Просмотр результатов.

```
mysql> select c1, bitmap_to_string(c2) from t1;
+------+----------------------+
| c1   | bitmap_to_string(c2) |
+------+----------------------+
|    1 | 1,2,3                |
|    2 | 4,5                  |
+------+----------------------+
```

Экспорт Bitmap из Selena в Hive

1. Создайте таблицу Bitmap в Selena и запишите в неё данные.

```
mysql> create table t1(c1 int, c2 bitmap bitmap_union) aggregate
key(c1) buckets 3 distributed by hash(c1);

mysql> select c1, bitmap_to_string(c2) from t1;
+------+----------------------+
| c1   | bitmap_to_string(c2) |
+------+----------------------+
|    1 | 1,2,3                |
|    2 | 4,5                  |
+------+----------------------+
```

2. Создайте таблицу Bitmap в Hive.

```
hive> create table t_bitmap(c1 bigint, c2 binary) stored as parquet;
```

3. Экспортируйте данные разными способами.

   - Экспорт данных через INSERT INTO FILES (формат Binary).

```
mysql> insert into files (
    "path" = "hdfs://<hdfs_ip>:<hdfs_port>/<hdfs_db>/t_bitmap/",
    "format"="parquet",
    "compression" = "uncompressed"
) select c1, bitmap_to_binary(c2) as c2 from t1;
```

- Экспорт данных через Hive Catalog (формат Binary).

```
mysql> insert into hive_catalog_hms.<hdfs_db>.t_bitmap select c1,
bitmap_to_binary(c2) from t1;
```

4. Просмотр результатов в Hive.

```
hive> select c1, bitmap_to_string(c2) from t_bitmap;
1       1,2,3
2       4,5
```

2.6 Python UDF

> **EXPERIMENTAL FEATURE**
>
> Рекомендации по использованию экспериментальных функций

В этом разделе описывается, как разрабатывать пользовательские функции (UDF) с использованием Python.

Начиная с версии 3.4.0, Selena поддерживает создание Python UDF.

В настоящее время Selena поддерживает только скалярные UDF на Python.

Предварительные требования

Перед началом работы убедитесь, что выполнены следующие требования:

- Установлен Python 3.8 или более поздняя версия.
- UDF включены в Selena путём установки параметра `enable_udf` в значение `true` в файле конфигурации FE **fe/conf/fe.conf**, после чего необходимо перезапустить узлы FE для применения конфигурации. Подробнее см. Конфигурация FE - enable_udf.
- Для Python UDF требуется пакет pyarrow.

- Установите местоположение среды интерпретатора Python в экземпляре BE с помощью переменной окружения. Добавьте переменную `python_envs` и установите её значение в путь установки интерпретатора Python, например, `/opt/Python-3.8/`.

Разработка и использование Python UDF

Синтаксис:

```
CREATE [GLOBAL] FUNCTION function_name(arg_type [, ...])
RETURNS return_type
{PROPERTIES ("key" = "value" [, ...]) | key="value" [...] }
[AS $$ $$]
```

Параметр	Обязательный	Описание
GLOBAL	Нет	Создать ли глобальную UDF.
function_name	Да	Имя создаваемой функции. Вы можете включить имя базы данных в этот параметр, например, `db1.my_func`. Если `function_name` содержит имя базы данных, UDF создаётся в этой базе данных. В противном случае UDF создаётся в текущей базе данных. Имя новой функции и её параметры не должны совпадать с существующим именем в целевой базе данных. В противном случае функция не может быть создана. Создание выполняется успешно, если имя функции совпадает, но параметры отличаются.
arg_type	Да	Тип аргумента функции. Добавленный аргумент может быть представлен как `, ...`. Поддерживаемые типы данных см. в Соответствие между типами данных SQL и Python.
return_type	Да	Тип возвращаемого значения функции. Поддерживаемые типы данных см. в Соответствие между типами данных SQL и Python.
PROPERTIES	Да	Свойства функции, которые зависят от типа создаваемой UDF.
AS $$ $$	Нет	Укажите встроенный код UDF между метками `$$`.

Свойства включают:

Свойство	Обязательное	Описание
type	Да	Тип UDF. Значение `Python` указывает на создание UDF на основе Python.
symbol	Да	Имя класса для проекта Python, к которому принадлежит UDF. Значение этого параметра имеет формат `<package_name>.<class_name>`.
input	Нет	Тип входных данных. Допустимые значения: `scalar` (по умолчанию) и `arrow` (векторизованный ввод).
file	Нет	HTTP-URL, по которому можно загрузить файл пакета Python, содержащий код для UDF. Значение этого параметра имеет формат `http://<http_server_ip>:<http_server_port>/<python_package_name>`. Обратите внимание, что имя пакета должно иметь суффикс `.py.zip`. Значение по умолчанию: `inline`, что означает создание встроенной UDF.

Создание встроенной скалярной UDF с использованием Python

Следующий пример создаёт встроенную функцию `echo` со скалярным вводом с использованием Python.

```
CREATE FUNCTION python_echo(INT)
RETURNS INT
type = 'Python'
symbol = 'echo'
file = 'inline'
AS
$$
def echo(x):
    return x
$$
;
```

Создание встроенной векторизованной UDF с использованием Python

Поддерживается векторизованный ввод для повышения производительности обработки UDF.

Следующий пример создаёт встроенную функцию `add` с векторизованным вводом с использованием Python.

```
CREATE FUNCTION python_add(INT)
RETURNS INT
type = 'Python'
symbol = 'add'
input = "arrow"
AS
$$
import pyarrow.compute as pc
def add(x):
    return pc.add(x, 1)
$$
;
```

Создание пакетной UDF с использованием Python

При создании пакета Python вы должны упаковать модуль в файлы `.py.zip`, которые должны соответствовать формату zipimport.

```
> tree .
.
├── main.py
└── yaml
    ├── composer.py
    ├── constructor.py
    ├── cyaml.py
    ├── dumper.py
    ├── emitter.py
    ├── error.py
    ├── events.py
    ├── __init__.py
    ├── loader.py
    ├── nodes.py
    ├── parser.py
```

```
> cat main.py
import numpy
import yaml

def echo(a):
    return yaml.__version__
```

```
CREATE FUNCTION py_pack(string)
RETURNS  string
symbol = "add"
type = "Python"
file = "http://HTTP_IP:HTTP_PORT/m1.py.zip"
symbol = "main.echo"
;
```

Приложение

Соответствие между типами данных SQL и Python

Тип SQL	Тип Python 3
SCALAR	
TINYINT/SMALLINT/INT/BIGINT/LARGEINT	INT
STRING	string
DOUBLE	FLOAT
BOOLEAN	BOOL
DATETIME	DATETIME.DATETIME
FLOAT	FLOAT
CHAR	STRING
VARCHAR	STRING
DATE	DATETIME.DATE
DECIMAL	DECIMAL.DECIMAL
ARRAY	List
MAP	Dict
STRUCT	COLLECTIONS.NAMEDTUPLE
JSON	dict
VECTORIZED	
TYPE_BOOLEAN	pyarrow.lib.BoolArray
TYPE_TINYINT	pyarrow.lib.Int8Array
TYPE_SMALLINT	pyarrow.lib.Int15Array
TYPE_INT	pyarrow.lib.Int32Array
TYPE_BIGINT	pyarrow.lib.Int64Array
TYPE_FLOAT	pyarrow.FloatArray
TYPE_DOUBLE	pyarrow.DoubleArray
VARCHAR	pyarrow.StringArray

Тип SQL	Тип Python 3
DECIMAL	pyarrow.Decimal128Array
DATE	pyarrow.Date32Array
TYPE_TIME	pyarrow.TimeArray
ARRAY	pyarrow.ListArray

Компиляция Python

Выполните следующие шаги для компиляции Python:

1. Получите пакет OpenSSL.

```
wget 'https://github.com/openssl/openssl/archive/OpenSSL_1_1_1m.tar.gz'
```

2. Распакуйте пакет.

```
tar -zxf OpenSSL_1_1_1m.tar.gz
```

3. Перейдите в распакованную папку.

```
cd openssl-OpenSSL_1_1_1m
```

4. Установите переменную окружения `OPENSSL_DIR`.

```
export OPENSSL_DIR=`pwd`/install
```

5. Подготовьте исходный код для компиляции.

```
./Configure --prefix=`pwd`/install
./config --prefix=`pwd`/install
```

6. Скомпилируйте OpenSSL.

```
make -j 16 && make install
```

7. Установите переменную окружения `LD_LIBRARY_PATH`.

```
LD_LIBRARY_PATH=$OPENSSL_DIR/lib:$LD_LIBRARY_PATH
```

8. Вернитесь в рабочий каталог и получите пакет Python.

```
wget 'https://www.python.org/ftp/python/3.12.9/Python-3.12.9.tgz'
```

9. Распакуйте пакет.

```
tar -zxf ./Python-3.12.9.tgz
```

10. Перейдите в распакованную папку.

```
cd Python-3.12.9
```

11. Создайте и перейдите в каталог `build`.

```
mkdir build && cd build
```

12. Подготовьте исходный код для компиляции.

```
../configure --prefix=`pwd`/install --with-openssl=$OPENSSL_DIR
```

13. Скомпилируйте Python.

```
make -j 16 && make install
```

14. Установите PyArrow и grpcio.

```
./install/bin/pip3 install pyarrow grpcio
```

15. Сожмите файлы в пакет.

```
tar -zcf ./Python-3.12.9.tar.gz install
```

16. Распространите пакет на целевой сервер BE и распакуйте пакет.

```
tar -zxf ./Python-3.12.9.tar.gz
```

17. Измените файл конфигурации BE **be.conf**, добавив следующий элемент конфигурации.

```
python_envs=/home/disk1/sr/install
```

2.7 Aggregate

Any_value

Получает произвольную строку из каждой агрегированной группы. Вы можете использовать эту функцию для оптимизации запроса, который содержит предложение `GROUP BY`.

Синтаксис

```
ANY_VALUE(expr)
```

Параметры

`expr`: выражение, которое агрегируется. Начиная с версии 3.2, `expr` может вычисляться до ARRAY, MAP и STRUCT.

Возвращаемое значение

Возвращает произвольную строку из каждой агрегированной группы. Возвращаемое значение недетерминировано.

Примеры

Создайте таблицу `t0` и вставьте данные в эту таблицу.

```
CREATE TABLE t0(
  a INT,
  b BIGINT,
  c SMALLINT,
  d ARRAY<INT>,
  e JSON,
  f MAP<INT, INT>,
  g STRUCT<a STRING, b INT>
)
DUPLICATE KEY(a)
DISTRIBUTED BY HASH(a);

INSERT INTO t0 VALUES
(1, 1, 1, [2,3,4],parse_json('{"a":1, "b":true}'), map{1:1,3:4}, row(1,
2)),
(1, 2, 1, [2,3,5],parse_json('{"a":2, "b":true}'), map{1:2,3:3},row(2, 2)),
(2, 1, 1, [2,3,6],parse_json('{"a":3, "b":true}'), map{2:1,3:2},row(3, 2)),
(2, 2, 2, [2,4,5],parse_json('{"a":4, "b":false}'),map{1:3,3:1},row(4, 2)),
(3, 1, 1, [3,3,5],parse_json('{"a":5, "b":false}'),map{2:1,3:3},row(1, 2));
```

```
mysql> select * from t0 order by a;
+------+------+------+---------+----------------------+-----------+--------
---------+
| a    | b    | c    | d       | e                    | f         | g
|
+------+------+------+---------+----------------------+-----------+--------
---------+
|    1 |    1 |    1 | [2,3,4] | {"a": 1, "b": true}  | {1:1,3:4} |
{"a":"1","b":2} |
|    1 |    2 |    1 | [2,3,5] | {"a": 2, "b": true}  | {1:2,3:3} |
{"a":"2","b":2} |
|    2 |    1 |    1 | [2,3,6] | {"a": 3, "b": true}  | {2:1,3:2} |
{"a":"3","b":2} |
|    2 |    2 |    2 | [2,4,5] | {"a": 4, "b": false} | {1:3,3:1} |
{"a":"4","b":2} |
|    3 |    1 |    1 | [3,3,5] | {"a": 5, "b": false} | {2:1,3:3} |
{"a":"1","b":2} |
+------+------+------+---------+----------------------+-----------+--------
---------+
5 rows in set (0.01 sec)
```

Используйте `any_value` для запроса данных. Для `a = 1` и `a = 2` возвращается произвольная строка для `b`.

```
mysql> select a,any_value(b),sum(c) from t0 group by a;
+------+----------------+----------+
| a    | any_value(`b`) | sum(`c`) |
+------+----------------+----------+
|    1 |              1 |        2 |
|    2 |              1 |        3 |
|    3 |              1 |        1 |
+------+----------------+----------+
3 rows in set (0.01 sec)

mysql> select a,any_value(d),sum(b) from t0 group by a;
+------+--------------+--------+
| a    | any_value(d) | sum(b) |
+------+--------------+--------+
|    3 | [3,3,5]      |      1 |
|    1 | [2,3,4]      |      3 |
|    2 | [2,3,6]      |      3 |
+------+--------------+--------+
2 rows in set (0.01 sec)
```

Approx_count_distinct

Возвращает приблизительное значение агрегатной функции, аналогичное результату COUNT(DISTINCT col).

Работает быстрее, чем комбинация COUNT и DISTINCT, и использует память фиксированного размера, поэтому для столбцов с высокой кардинальностью используется меньше памяти.

Синтаксис

```
APPROX_COUNT_DISTINCT(expr)
```

Примеры

```
MySQL > select approx_count_distinct(query_id) from log_statis group by
datetime;
+-----------------------------------+
| approx_count_distinct(`query_id`) |
+-----------------------------------+
| 17721                             |
+-----------------------------------+
```

Ключевое слово

APPROX_COUNT_DISTINCT

Approx_top_k

Возвращает топ `k` наиболее часто встречающихся значений элементов в выражении `expr` вместе с их приблизительным количеством.

Эта функция поддерживается начиная с версии 3.0.

Синтаксис

```
APPROX_TOP_K(<expr> [ , <k> [ , <counter_num> ] ] )
```

Аргументы

- `expr`: Выражение типа STRING, BOOLEAN, DATE, DATETIME или числового типа.
- `k`: Необязательный INTEGER литерал, больший 0. Если `k` не указан, по умолчанию используется `5`. Максимальное значение — `100000`.
- `counter_num`: Необязательный INTEGER литерал, больший или равный `k`. Чем больше `counter_num`, тем точнее результат. Однако это также связано с увеличением затрат на CPU и память.
    - Максимальное значение — `100000`.
    - Если `counter_num` не указан, по умолчанию используется `max(min(2 * k, 100), 100000)`.

Возвращаемое значение

Результаты возвращаются как ARRAY типа STRUCT, где каждый STRUCT содержит поле `item` для значения (с его исходным входным типом) и поле `count` (типа BIGINT) с приблизительным количеством вхождений. Массив отсортирован по `count` в порядке убывания.

Агрегатная функция возвращает топ `k` наиболее часто встречающихся значений элементов в выражении expr вместе с их приблизительным количеством. Погрешность в каждом подсчёте может составлять до `2.0 * numRows / counter_num`, где `numRows` — общее количество строк. Более высокие значения `counter_num` обеспечивают лучшую точность за счёт увеличенного использования памяти. Выражения, которые имеют менее `counter_num` различных элементов, дадут точные подсчёты элементов. Результаты включают значения `NULL` как отдельный элемент в результатах.

Примеры

Используйте данные из таблицы scores в качестве примера.

```
-- Вычислить распределение оценок по каждому предмету.
MySQL > SELECT subject, APPROX_TOP_K(score)  AS top_k FROM scores GROUP BY
subject;
+---------+-------------------------------------------------------------
-----------------------------------------------------+
| subject | top_k
|
+---------+-------------------------------------------------------------
-----------------------------------------------------+
| physics | [{"item":99,"count":2},{"item":null,"count":1},
{"item":100,"count":1},{"item":85,"count":1},{"item":60,"count":1}] |
| english | [{"item":null,"count":1},{"item":92,"count":1},
{"item":98,"count":1},{"item":100,"count":1},{"item":85,"count":1}] |
| NULL    | [{"item":90,"count":1}]
|
| math    | [{"item":80,"count":2},{"item":null,"count":1},
{"item":92,"count":1},{"item":95,"count":1},{"item":70,"count":1}]  |
+---------+-------------------------------------------------------------
-----------------------------------------------------+

-- Вычислить распределение оценок по предмету математика.
MySQL > SELECT subject, APPROX_TOP_K(score)  AS top_k FROM scores WHERE
subject IN  ('math') GROUP BY subject;
+---------+-------------------------------------------------------------
-----------------------------------------------------+
| subject | top_k
|
+---------+-------------------------------------------------------------
-----------------------------------------------------+
| math    | [{"item":80,"count":2},{"item":null,"count":1},
{"item":95,"count":1},{"item":92,"count":1},{"item":70,"count":1}] |
+---------+-------------------------------------------------------------
-----------------------------------------------------+
```

Ключевое слово

APPROX_TOP_K

Avg

Возвращает среднее значение выбранных полей.

Необязательный параметр DISTINCT может использоваться для возврата взвешенного среднего.

Синтаксис

```
AVG([DISTINCT] expr)
```

Примеры

```
MySQL > SELECT datetime, AVG(cost_time)
FROM log_statis
group by datetime;
+---------------------+--------------------+
| datetime            | avg(`cost_time`)   |
+---------------------+--------------------+
| 2019-07-03 21:01:20 | 25.827794561933533 |
+---------------------+--------------------+

MySQL > SELECT datetime, AVG(distinct cost_time)
FROM log_statis
group by datetime;
+---------------------+---------------------------+
| datetime            | avg(DISTINCT `cost_time`) |
+---------------------+---------------------------+
| 2019-07-04 02:23:24 |        20.666666666666668 |
+---------------------+---------------------------+
```

Ключевое слово

AVG

Bitmap

Вот простой пример, иллюстрирующий использование нескольких агрегатных функций в Bitmap. Подробные определения функций или дополнительные функции Bitmap см. в разделе bitmap-functions.

Создание таблицы

При создании таблицы необходима модель агрегации. Тип данных — bitmap, а функция агрегации — bitmap_union.

```
CREATE TABLE `pv_bitmap` (
  `dt` int(11) NULL COMMENT "",
  `page` varchar(10) NULL COMMENT "",
  `user_id` bitmap BITMAP_UNION NULL COMMENT ""
) ENGINE=OLAP
AGGREGATE KEY(`dt`, `page`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`dt`);
```

Примечание: При большом объёме данных лучше создать таблицу rollup, соответствующую частым bitmap_union.

```
ALTER TABLE pv_bitmap ADD ROLLUP pv (page, user_id);
```

Загрузка данных

`TO_BITMAP (expr)`: Преобразует беззнаковое bigint от 0 до 18446744073709551615 в bitmap

`BITMAP_EMPTY ()`: Генерирует пустые столбцы bitmap, используется для значения по умолчанию, заполняемого при вставке или вводе

`BITMAP_HASH (expr)`: Преобразует столбцы любого типа в bitmap с помощью 32-битного хеширования

`BITMAP_HASH64 (expr)`: Преобразует столбцы любого типа в bitmap с помощью 64-битного хеширования

Stream Load

При вводе данных с помощью Stream Load вы можете преобразовать данные в поле Bitmap следующим образом:

```
cat data | curl --location-trusted -u user:passwd -T - \
    -H "columns: dt,page,user_id, user_id=to_bitmap(user_id)" \
    http://host:8410/api/test/testDb/_stream_load
```

```
cat data | curl --location-trusted -u user:passwd -T - \
    -H "columns: dt,page,user_id, user_id=bitmap_hash(user_id)" \
    http://host:8410/api/test/testDb/_stream_load
```

```
cat data | curl --location-trusted -u user:passwd -T - \
    -H "columns: dt,page,user_id, user_id=bitmap_hash64(user_id)" \
    http://host:8410/api/test/testDb/_stream_load
```

```
cat data | curl --location-trusted -u user:passwd -T - \
    -H "columns: dt,page,user_id, user_id=bitmap_empty()" \
    http://host:8410/api/test/testDb/_stream_load
```

Insert Into

При вводе данных с помощью Insert Into вам нужно выбрать соответствующий режим в зависимости от типа столбцов в исходной таблице.

- Тип столбца id2 в исходной таблице — bitmap

```
insert into bitmap_table1
select id, id2 from bitmap_table2;
```

- Тип столбца id2 в целевой таблице — bitmap

```
insert into bitmap_table1 (id, id2)
values (1001, to_bitmap(1000))
, (1001, to_bitmap(2000));
```

- Тип столбца id2 в исходной таблице — bitmap, и это результат агрегации с использованием bit_map_union().

```
insert into bitmap_table1
select id, bitmap_union(id2) from bitmap_table2 group by id;
```

- Тип столбца id2 в исходной таблице — INT, и тип bitmap генерируется с помощью to_bitmap().

```
insert into bitmap_table1
select id, to_bitmap(id2) from table;
```

- Тип столбца id2 в исходной таблице — STRING, и тип bitmap генерируется с помощью bitmap_hash().

```
insert into bitmap_table1
select id, bitmap_hash(id2) from table;
```

- Тип столбца id2 в исходной таблице — STRING, и тип bitmap генерируется с помощью bitmap_hash64().

```
insert into bitmap_table1
select id, bitmap_hash64(id2) from table;
```

Запрос данных

Синтаксис

`BITMAP_UNION (expr)`: Вычисляет объединение входных Bitmap и возвращает новый Bitmap.

`BITMAP_UNION_COUNT (expr)`: Вычисляет объединение входных Bitmap и возвращает его кардинальность, эквивалентно BITMAP_COUNT (BITMAP_UNION (expr)). Рекомендуется использовать функцию BITMAP_UNION_COUNT в первую очередь, так как её производительность лучше, чем у BITMAP_COUNT (BITMAP_UNION (expr)).

`BITMAP_UNION_INT (expr)`: Вычисляет количество различных значений в столбцах типа TINYINT, SMALLINT и INT, возвращает значение, аналогичное COUNT (DISTINCT expr).

`INTERSECT_COUNT (bitmap_column_to_count, filter_column, filter_values ...)`: Вычисляет кардинальность пересечения нескольких bitmap, которые удовлетворяют условию filter_column. bitmap_column_to_count — это столбец типа bitmap, filter_column — это столбец различных измерений, а filter_values — это список значений измерений.

`BITMAP_INTERSECT(expr)`: Вычисляет пересечение этой группы значений bitmap и возвращает новый bitmap.

Пример

Следующий SQL использует таблицу `pv_bitmap` выше в качестве примера:

Вычислить дедуплицированное значение для `user_id`:

```
select bitmap_union_count(user_id)
from pv_bitmap;

select bitmap_count(bitmap_union(user_id))
from pv_bitmap;
```

Вычислить дедуплицированное значение `id`:

```
select bitmap_union_int(id)
from pv_bitmap;
```

Вычислить удержание `user_id`:

```
select intersect_count(user_id, page, 'game') as game_uv,
    intersect_count(user_id, page, 'shopping') as shopping_uv,
    intersect_count(user_id, page, 'game', 'shopping') as retention --
Количество пользователей, которые посещают страницы 'game' и 'shopping'
from pv_bitmap
where page in ('game', 'shopping');
```

Ключевое слово

BITMAP,BITMAP_COUNT,BITMAP_EMPTY,BITMAP_UNION,BITMAP_UNION_INT,TO_BITMAP,BITMAP_UNION_COUNT,INTERSECT_COUNT,BITMAP_INTERSECT

Bool_or

Возвращает true, если хотя бы одна строка для `expr` имеет значение true. В противном случае возвращает false. `boolor_agg` является псевдонимом для этой функции.

Синтаксис

```
bool_or(expr)
```

Параметры

`expr`: Выражение должно вычисляться до булева значения, то есть true или false.

Возвращаемое значение

Возвращает булево значение. Ошибка возвращается, если `expr` не существует.

Примечания по использованию

Эта функция игнорирует значения NULL.

Примеры

1. Создайте таблицу с именем `employees`.

```
CREATE TABLE IF NOT EXISTS employees (
    region_num    TINYINT,
    id            BIGINT,
    is_manager    BOOLEAN
    )
    DISTRIBUTED BY HASH(region_num);
```

2. Вставьте данные в `employees`.

```
INSERT INTO employees VALUES
(3,432175, TRUE),
(4,567832, FALSE),
(3,777326, FALSE),
(5,342611, TRUE),
(2,403882, FALSE);
```

3. Запросите данные из `employees`.

```
MySQL > select * from employees;
+------------+--------+------------+
| region_num | id     | is_manager |
+------------+--------+------------+
|          3 | 432175 |          1 |
|          4 | 567832 |          0 |
|          3 | 777326 |          0 |
|          5 | 342611 |          1 |
|          2 | 403882 |          0 |
+------------+--------+------------+
5 rows in set (0.01 sec)
```

4. Используйте эту функцию для проверки наличия менеджеров в каждом регионе.

   Пример 1: Проверить, есть ли хотя бы один менеджер в каждом регионе.

```
MySQL > SELECT region_num, bool_or(is_manager) from employees
group by region_num;

+------------+---------------------+
| region_num | bool_or(is_manager) |
+------------+---------------------+
|          3 |                   1 |
|          4 |                   0 |
|          5 |                   1 |
|          2 |                   0 |
+------------+---------------------+
4 rows in set (0.01 sec)
```

Ключевое слово

BOOL_OR, bool_or, boolor_agg

Corr

Возвращает коэффициент корреляции Пирсона между двумя выражениями. Эта функция поддерживается начиная с версии 2.5.10. Она также может использоваться как оконная функция.

Синтаксис

```
CORR(expr1, expr2)
```

Параметры

`expr1` и `expr2` должны вычисляться до TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE или DECIMAL.

Если `expr1` и `expr2` являются столбцами таблицы, эта функция вычисляет коэффициент корреляции этих двух столбцов.

Возвращаемое значение

Возвращает значение DOUBLE. Формула следующая, где `n` представляет количество строк таблицы:

$$\frac{\sum_{i=1}^{n}\left((x_i - \bar{x})(y_i - \bar{y})\right)}{\sqrt{\sum_{i=1}^{n}\left((x_i - \bar{x})^2\right) \cdot \sum_{i=1}^{n}\left((y_i - \bar{y})^2\right)}}$$

Примечания по использованию

- Строка данных учитывается только тогда, когда оба столбца в этой строке являются ненулевыми значениями. В противном случае эта строка данных исключается из результата.
- Возвращается 0, если `n` равно 1.
- Возвращается NULL, если любой вход — NULL.

Примеры

Предположим, таблица `agg` имеет следующие данные:

```
mysql> select * from agg;
+------+-------+-------+
| no   | k     | v     |
+------+-------+-------+
|    1 | 10.00 |  NULL |
|    2 | 10.00 | 11.00 |
|    2 | 20.00 | 22.00 |
|    2 | 25.00 |  NULL |
|    2 | 30.00 | 35.00 |
+------+-------+-------+
```

Вычислить коэффициент корреляции столбцов `k` и `v`:

```
mysql> select no,CORR(k,v) from agg group by no;
+------+--------------------+
| no   | corr(k, v)         |
+------+--------------------+
|    1 |               NULL |
|    2 | 0.9988445981121532 |
+------+--------------------+
```

Count

Возвращает общее количество строк, указанных выражением.

Эта функция имеет три варианта:

- `COUNT(*)` подсчитывает все строки в таблице, независимо от того, содержат ли они значения NULL.
- `COUNT(expr)` подсчитывает количество строк с ненулевыми значениями в определённом столбце.
- `COUNT(DISTINCT expr)` подсчитывает количество различных ненулевых значений в столбце.

`COUNT(DISTINCT expr)` используется для точного подсчёта уникальных значений. Если вам требуется более высокая производительность подсчёта уникальных значений, см. Использование bitmap для точного подсчёта уникальных значений.

Начиная с Selena 2.4, вы можете использовать несколько COUNT(DISTINCT) в одном операторе.

Синтаксис

```
COUNT(expr)
COUNT(DISTINCT expr [,expr,...])`
```

Параметры

`expr`: столбец или выражение, на основе которого выполняется `count()`. Если `expr` — имя столбца, столбец может быть любого типа данных.

Возвращаемое значение

Возвращает числовое значение. Если строки не найдены, возвращается 0. Эта функция игнорирует значения NULL.

Примеры

Предположим, есть таблица с именем `test`. Запросите страну, категорию и поставщика каждого заказа по `id`.

```
select * from test order by id;
+------+----------+----------+------------+
| id   | country  | category | supplier   |
+------+----------+----------+------------+
| 1001 | US       | A        | supplier_1 |
| 1002 | Thailand | A        | supplier_2 |
| 1003 | Turkey   | B        | supplier_3 |
| 1004 | US       | A        | supplier_2 |
| 1005 | China    | C        | supplier_4 |
| 1006 | Japan    | D        | supplier_3 |
| 1007 | Japan    | NULL     | supplier_5 |
+------+----------+----------+------------+
```

Пример 1: Подсчитать количество строк в таблице `test`.

```
select count(*) from test;
+----------+
| count(*) |
+----------+
|        7 |
+----------+
```

Пример 2: Подсчитать количество значений в столбце `id`.

```
select count(id) from test;
+-----------+
| count(id) |
+-----------+
|         7 |
+-----------+
```

Пример 3: Подсчитать количество значений в столбце `category`, игнорируя значения NULL.

```
select count(category) from test;
  +-----------------+
  | count(category) |
  +-----------------+
  |         6       |
  +-----------------+
```

Пример 4: Подсчитать количество различных значений в столбце `category`.

```
select count(distinct category) from test;
+-------------------------+
| count(DISTINCT category) |
+-------------------------+
|                       4 |
+-------------------------+
```

Пример 5: Подсчитать количество комбинаций, которые можно составить из `category` и `supplier`.

```
select count(distinct category, supplier) from test;
+-----------------------------------+
| count(DISTINCT category, supplier) |
+-----------------------------------+
|                                 5 |
+-----------------------------------+
```

В выводе комбинация с `id` 1004 дублирует комбинацию с `id` 1002. Они учитываются только один раз. Комбинация с `id` 1007 имеет значение NULL и не учитывается.

Пример 6: Использование нескольких COUNT(DISTINCT) в одном операторе.

```
select count(distinct country, category), count(distinct country,supplier)
from test;
+-----------------------------------+-----------------------------------+
| count(DISTINCT country, category) | count(DISTINCT country, supplier) |
+-----------------------------------+-----------------------------------+
|                                 6 |                                 7 |
+-----------------------------------+-----------------------------------+
```

Count_if

Возвращает количество записей, удовлетворяющих указанному условию, или `0`, если ни одна запись не удовлетворяет условию.

Эта функция не поддерживает `DISTINCT`, например `count_if(DISTINCT x)`.

Эта функция внутренне преобразуется в `COUNT` + `IF`:

- До: `COUNT_IF(x)`
- После: `COUNT(IF(x, 1, NULL))`

Синтаксис

```
COUNT_IF(condition)
```

Параметры

`condition`: условие, которое должно вычисляться до BOOLEAN значения. Вы можете указать несколько условий count_if в одном операторе SELECT.

Возвращаемое значение

Возвращает числовое значение. Если записи не найдены, возвращается `0`. Эта функция игнорирует значения NULL.

Примеры

Предположим, есть таблица с именем `test_count_if`.

```
CREATE TABLE `test_count_if` (
  `v1` varchar(65533) NULL COMMENT "",
  `v2` varchar(65533) NULL COMMENT "",
  `v3` datetime NULL COMMENT "",
  `v4` int null
) ENGINE=OLAP
DUPLICATE KEY(v1, v2, v3)
PARTITION BY RANGE(`v3`)
(PARTITION p20220418 VALUES [("2022-04-18 00:00:00"), ("2022-04-19
00:00:00")),
PARTITION p20220419 VALUES [("2022-04-19 00:00:00"), ("2022-04-20
00:00:00")),
PARTITION p20220420 VALUES [("2022-04-20 00:00:00"), ("2022-04-21
00:00:00")),
PARTITION p20220421 VALUES [("2022-04-21 00:00:00"), ("2022-04-22
00:00:00")))
DISTRIBUTED BY HASH(`v1`) BUCKETS 4;

INSERT INTO test_count_if VALUES
('a','a', '2022-04-18 01:01:00', 1),
('a','b', '2022-04-18 02:01:00', NULL),
('a',NULL, '2022-04-18 02:05:00', 1),
('a','b', '2022-04-18 02:15:00', 3),
('a','b', '2022-04-18 03:15:00', 7),
('c',NULL, '2022-04-18 03:45:00', NULL),
('c',NULL, '2022-04-18 03:25:00', 2),
('c','a', '2022-04-18 03:27:00', 3);

SELECT * FROM test_count_if;
+------+------+---------------------+------+
| v1   | v2   | v3                  | v4   |
+------+------+---------------------+------+
| a    | NULL | 2022-04-18 02:05:00 |    1 |
| a    | a    | 2022-04-18 01:01:00 |    1 |
| a    | b    | 2022-04-18 02:01:00 | NULL |
| a    | b    | 2022-04-18 02:15:00 |    3 |
| a    | b    | 2022-04-18 03:15:00 |    7 |
| c    | NULL | 2022-04-18 03:25:00 |    2 |
| c    | NULL | 2022-04-18 03:45:00 | NULL |
| c    | a    | 2022-04-18 03:27:00 |    3 |
+------+------+---------------------+------+
```

Пример 1: Подсчитать количество строк в таблице `test_count_if`, где `v2` равен null.

```
select count_if(v2 is null) from test_count_if;
+----------------------+
| count_if(v2 IS NULL) |
+----------------------+
|                    3 |
+----------------------+
```

Пример 2: Подсчитать количество строк, где `v1 >= v2 or v4 = 1`

```
select count_if(v1 >= v2 or v4 = 1)from test_count_if;
+----------------------------------+
| count_if((v1 >= v2) OR (v4 = 1)) |
+----------------------------------+
|                                3 |
+----------------------------------+
```

Пример 3: Указать несколько условий.

```
select count_if(v1 >= v2), count_if(v1 >= v2 or v4 = 1), count_if(v1 >= v2
and v4 = 1) from test_count_if;
+--------------------+----------------------------------+------------------
-----------------+
| count_if(v1 >= v2) | count_if((v1 >= v2) OR (v4 = 1)) | count_if((v1 >=
v2) AND (v4 = 1)) |
+--------------------+----------------------------------+------------------
-----------------+
|                  2 |                                3 |
1 |
```

Covar_pop

Возвращает ковариацию генеральной совокупности двух выражений. Эта функция поддерживается начиная с версии 2.5.10. Она также может использоваться как оконная функция.

Синтаксис

```
COVAR_POP(expr1, expr2)
```

Параметры

`expr1` и `expr2` должны вычисляться до TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE или DECIMAL.

Если `expr1` и `expr2` являются столбцами таблицы, эта функция вычисляет ковариацию генеральной совокупности этих двух столбцов.

Возвращаемое значение

Возвращает значение DOUBLE. Формула следующая, где `n` представляет количество строк таблицы:

$$\frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n}$$

Примечания по использованию

- Строка данных учитывается только тогда, когда оба столбца в этой строке являются ненулевыми значениями. В противном случае эта строка данных исключается из результата.
- Возвращается NULL, если любой вход — NULL.

Примеры

Предположим, таблица `agg` имеет следующие данные:

```
mysql> select * from agg;
+------+-------+-------+
| no   | k     | v     |
+------+-------+-------+
|    1 | 10.00 |  NULL |
|    2 | 10.00 | 11.00 |
|    2 | 20.00 | 22.00 |
|    2 | 25.00 |  NULL |
|    2 | 30.00 | 35.00 |
+------+-------+-------+
```

Вычислить ковариацию генеральной совокупности столбцов `k` и `v`:

```
mysql> select no,COVAR_POP(k,v) from agg group by no;
+------+-------------------+
| no   | covar_pop(k, v)   |
+------+-------------------+
|    1 |              NULL |
|    2 | 79.99999999999999 |
+------+-------------------+
```

Covar_samp

Возвращает выборочную ковариацию двух выражений. Эта функция поддерживается начиная с версии 2.5.10. Она также может использоваться как оконная функция.

Синтаксис

```
COVAR_SAMP(expr1, expr2)
```

Параметры

`expr1` и `expr2` должны вычисляться до TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE или DECIMAL.

Если `expr1` и `expr2` являются столбцами таблицы, эта функция вычисляет выборочную ковариацию этих двух столбцов.

Возвращаемое значение

Возвращает значение DOUBLE. Формула следующая, где `n` представляет количество строк таблицы:

$$\frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

Примечания по использованию

- Строка данных учитывается только тогда, когда оба столбца в этой строке являются ненулевыми значениями. В противном случае эта строка данных исключается из результата.
- Возвращается 0, если `n` равно 1.
- Возвращается NULL, если любой вход — NULL.

Примеры

Предположим, таблица `agg` имеет следующие данные:

```
mysql> select * from agg;
+------+-------+-------+
| no   | k     | v     |
+------+-------+-------+
|    1 | 10.00 |  NULL |
|    2 | 10.00 | 11.00 |
|    2 | 20.00 | 22.00 |
|    2 | 25.00 |  NULL |
|    2 | 30.00 | 35.00 |
+------+-------+-------+
```

Вычислить выборочную ковариацию столбцов `k` и `v`:

```
mysql> select no,COVAR_SAMP(k,v) from agg group by no;
+------+--------------------+
| no   | covar_samp(k, v)   |
+------+--------------------+
|    1 |               NULL |
|    2 | 119.99999999999999 |
+------+--------------------+
```

Ds_hll_accumulate

Накапливает значения в скетч HyperLogLog и возвращает сериализованный скетч как VARBINARY. Эта функция является частью семейства функций DataSketches HLL для приблизительного подсчёта уникальных значений.

`ds_hll_accumulate` создаёт сериализованный скетч HyperLogLog, который можно объединять с другими скетчами с помощью `ds_hll_combine` и оценивать с помощью `ds_hll_estimate`.

Основана на Apache DataSketches и обеспечивает высокую точность для приблизительного подсчёта уникальных значений. Для получения дополнительной информации см. HyperLogLog Sketches.

Синтаксис

```
sketch ds_hll_accumulate(expr)
sketch ds_hll_accumulate(expr, log_k)
sketch ds_hll_accumulate(expr, log_k, tgt_type)
```

Параметры

- `expr`: Выражение для накопления в скетче. Может быть любого типа данных.
- `log_k`: Целое число. Диапазон [4, 21]. По умолчанию: 17. Контролирует точность и использование памяти скетча.
- `tgt_type`: Допустимые значения: `HLL_4`, `HLL_6` (по умолчанию) и `HLL_8`. Контролирует целевой тип скетча HyperLogLog.

Возвращаемый тип

Возвращает VARBINARY, содержащий сериализованный скетч HyperLogLog.

Примеры

```
-- Создание тестовой таблицы
CREATE TABLE t1 (
  id BIGINT,
  province VARCHAR(64),
  age SMALLINT,
  dt VARCHAR(10)
)
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 3;

-- Вставка тестовых данных
INSERT INTO t1 SELECT generate_series, generate_series, generate_series %
100, "2024-07-24"
FROM table(generate_series(1, 1000));

-- Базовое использование с одним аргументом
SELECT ds_hll_accumulate(id) FROM t1;

-- С пользовательским параметром log_k
SELECT ds_hll_accumulate(province, 20) FROM t1;

-- С параметрами log_k и tgt_type
SELECT ds_hll_accumulate(age, 12, "HLL_6") FROM t1;

-- Использование с group by
SELECT dt,
       ds_hll_accumulate(id),
       ds_hll_accumulate(province, 20),
       ds_hll_accumulate(age, 12, "HLL_6"),
       ds_hll_accumulate(dt)
FROM t1
GROUP BY dt
ORDER BY 1
LIMIT 3;
```

Связанные функции

- `ds_hll_combine`: Объединяет несколько сериализованных скетчей в один скетч

- `ds_hll_estimate`: Оценивает количество уникальных значений из сериализованного скетча
- `ds_hll_count_distinct`: Функция прямого приблизительного подсчёта уникальных значений

Ключевые слова

ds_HLL_ACCUMULATE, HLL, HYPERLOGLOG, APPROXIMATE, DISTINCT, COUNT

Ds_hll_combine

Объединяет несколько сериализованных скетчей HyperLogLog в один сериализованный скетч. Эта функция является частью семейства функций DataSketches HLL для приблизительного подсчёта уникальных значений.

`ds_hll_combine` принимает несколько VARBINARY сериализованных скетчей, созданных `ds_hll_accumulate`, и объединяет их в один скетч, который представляет объединение всех уникальных значений.

Основана на Apache DataSketches и обеспечивает высокую точность для приблизительного подсчёта уникальных значений. Для получения дополнительной информации см. HyperLogLog Sketches.

Синтаксис

```
sketch ds_hll_combine(sketch)
```

Параметры

- `sketch`: Столбец VARBINARY, содержащий сериализованные скетчи HyperLogLog, созданные `ds_hll_accumulate`.

Возвращаемый тип

Возвращает VARBINARY, содержащий объединённый сериализованный скетч HyperLogLog.

Примеры

```
-- Создание тестовой таблицы
CREATE TABLE t1 (
  id BIGINT,
  province VARCHAR(64),
  age SMALLINT,
  dt VARCHAR(10)
)
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 3;

-- Вставка тестовых данных
INSERT INTO t1 SELECT generate_series, generate_series, generate_series %
100, "2024-07-24"
FROM table(generate_series(1, 1000));

-- Создание таблицы для хранения скетчей
CREATE TABLE t2 (
  `id` bigint,
  `dt` varchar(10),
  `ds_id` binary,
  `ds_province` binary,
  `ds_age` binary,
  `ds_dt` binary
) ENGINE=OLAP
DISTRIBUTED BY HASH(id) BUCKETS 3;

-- Сохранение скетчей, созданных ds_hll_accumulate
INSERT INTO t2 SELECT id, dt,
  ds_hll_accumulate(id),
  ds_hll_accumulate(province, 10),
  ds_hll_accumulate(age, 20, "HLL_6"),
  ds_hll_accumulate(dt, 10, "HLL_8")
FROM t1;

-- Объединение скетчей с группировкой по дате
SELECT dt,
       ds_hll_combine(ds_id),
       ds_hll_combine(ds_province),
       ds_hll_combine(ds_age),
       ds_hll_combine(ds_dt)
FROM t2
GROUP BY dt
ORDER BY 1
```

```
LIMIT 3;
```

Связанные функции

- `ds_hll_accumulate`: Создаёт сериализованные скетчи HyperLogLog из данных
- `ds_hll_estimate`: Оценивает количество уникальных значений из сериализованного скетча
- `ds_hll_count_distinct`: Функция прямого приблизительного подсчёта уникальных значений

Ключевые слова

ds_HLL_COMBINE, HLL, HYPERLOGLOG, APPROXIMATE, DISTINCT, COUNT, MERGE, UNION

Ds_hll_count_distinct

Возвращает приблизительное значение агрегатной функции, аналогичное результату `COUNT(DISTINCT col)`. APPROX_COUNT_DISTINCT(expr) — похожая функция.

ds_hll_count_distinct быстрее, чем комбинация COUNT и DISTINCT, и использует память фиксированного размера, поэтому для столбцов с высокой кардинальностью требуется меньше памяти.

Работает медленнее, чем APPROX_COUNT_DISTINCT(expr), но с более высокой точностью, поскольку использует Apache Datasketches. Для получения дополнительной информации см. HyperLogLog Sketches.

Синтаксис

```
ds_hll_count_distinct(expr, [log_k], [tgt_type])
```

- `log_k`: Целое число. Диапазон [4, 21]. По умолчанию: 17.
- `tgt_type`: Допустимые значения: `HLL_4`, `HLL_6` (по умолчанию) и `HLL_8`.

Примеры

```
mysql> CREATE TABLE t1 (
    ->   id BIGINT NOT NULL,
    ->   province VARCHAR(64),
    ->   age SMALLINT,
    ->   dt VARCHAR(10) NOT NULL
    -> )
    -> DUPLICATE KEY(id)
    -> DISTRIBUTED BY HASH(id) BUCKETS 4;
Query OK, 0 rows affected (0.02 sec)

mysql> insert into t1 select generate_series, generate_series,
generate_series % 100, "2024-07-24" from table(generate_series(1, 100000));

Query OK, 100000 rows affected (0.29 sec)

mysql> select ds_hll_count_distinct(id), ds_hll_count_distinct(province),
ds_hll_count_distinct(age), ds_hll_count_distinct(dt) from t1 order by 1,
2;
+---------------------------+---------------------------------+------------
----------------+---------------------------+
| ds_hll_count_distinct(id) | ds_hll_count_distinct(province) |
ds_hll_count_distinct(age) | ds_hll_count_distinct(dt) |
+---------------------------+---------------------------------+------------
----------------+---------------------------+
|                    100090 |                          100140 |
100 |                         1 |
+---------------------------+---------------------------------+------------
----------------+---------------------------+
1 row in set (0.07 sec)

mysql> select ds_hll_count_distinct(id, 21),
ds_hll_count_distinct(province, 21), ds_hll_count_distinct(age, 21),
ds_hll_count_distinct(dt, 21) from t1 order by 1, 2;
+------------------------------+-------------------------------------+----
----------------------------+------------------------------+
| ds_hll_count_distinct(id, 21) | ds_hll_count_distinct(province, 21) |
ds_hll_count_distinct(age, 21) | ds_hll_count_distinct(dt, 21) |
+------------------------------+-------------------------------------+----
----------------------------+------------------------------+
|                        99995 |                              100001 |
100 |                            1 |
+------------------------------+-------------------------------------+----
----------------------------+------------------------------+
```

```
1 row in set (0.07 sec)


mysql> select ds_hll_count_distinct(id, 10, "HLL_8"),
ds_hll_count_distinct(province, 10, "HLL_8"), ds_hll_count_distinct(age,
10, "HLL_8"), ds_hll_count_distinct(dt, 10, "HLL_8") from t1 order by 1, 2;
+----------------------------------------+---------------------------------
-------------+-----------------------------------------+-------------------
---------------------+
| ds_hll_count_distinct(id, 10, 'HLL_8') | ds_hll_count_distinct(province,
10, 'HLL_8') | ds_hll_count_distinct(age, 10, 'HLL_8') |
ds_hll_count_distinct(dt, 10, 'HLL_8') |
+----------------------------------------+---------------------------------
-------------+-----------------------------------------+-------------------
---------------------+
|                                  99844 |
101905 |                                     96 |
1 |
+----------------------------------------+---------------------------------
-------------+-----------------------------------------+-------------------
---------------------+
1 row in set (0.09 sec)
```

Ключевые слова

DS_HLL_COUNT_DISTINCT,APPROX_COUNT_DISTINCT

Ds_hll_estimate

Оценивает приблизительное количество уникальных значений из сериализованного скетча HyperLogLog. Эта функция является частью семейства функций DataSketches HLL для приблизительного подсчёта уникальных значений.

`ds_hll_estimate` принимает VARBINARY сериализованный скетч, созданный `ds_hll_accumulate` или `ds_hll_combine`, и возвращает оценочное количество уникальных значений.

Основана на Apache DataSketches и обеспечивает высокую точность для приблизительного подсчёта уникальных значений. Для получения дополнительной информации см. HyperLogLog Sketches.

Синтаксис

```
bigint ds_hll_estimate(sketch)
```

Параметры

- `sketch`: Столбец VARBINARY, содержащий сериализованные скетчи HyperLogLog, созданные `ds_hll_accumulate` или `ds_hll_combine`.

Возвращаемый тип

Возвращает BIGINT, представляющий оценочное количество уникальных значений.

Примеры

```
-- Создание тестовой таблицы
CREATE TABLE t1 (
  id BIGINT,
  province VARCHAR(64),
  age SMALLINT,
  dt VARCHAR(10)
)
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 3;

-- Вставка тестовых данных
INSERT INTO t1 SELECT generate_series, generate_series, generate_series %
100, "2024-07-24"
FROM table(generate_series(1, 1000));

-- Создание таблицы для хранения скетчей
CREATE TABLE t2 (
  `id` bigint,
  `dt` varchar(10),
  `ds_id` binary,
  `ds_province` binary,
  `ds_age` binary,
  `ds_dt` binary
) ENGINE=OLAP
DISTRIBUTED BY HASH(id) BUCKETS 3;

-- Сохранение скетчей, созданных ds_hll_accumulate
INSERT INTO t2 SELECT id, dt,
  ds_hll_accumulate(id),
  ds_hll_accumulate(province, 10),
  ds_hll_accumulate(age, 20, "HLL_6"),
  ds_hll_accumulate(dt, 10, "HLL_8")
FROM t1;

-- Оценка количества уникальных значений с группировкой по дате
SELECT dt,
       ds_hll_estimate(ds_id),
       ds_hll_estimate(ds_province),
       ds_hll_estimate(ds_age),
       ds_hll_estimate(ds_dt)
FROM t2
GROUP BY dt
```

```
ORDER BY 1
LIMIT 3;
```

Связанные функции

- `ds_hll_accumulate`: Создаёт сериализованные скетчи HyperLogLog из данных
- `ds_hll_combine`: Объединяет несколько сериализованных скетчей в один скетч
- `ds_hll_count_distinct`: Функция прямого приблизительного подсчёта уникальных значений

Ключевые слова

ds_HLL_ESTIMATE, HLL, HYPERLOGLOG, APPROXIMATE, DISTINCT, COUNT, ESTIMATE

Ds_theta_count_distinct

Возвращает приблизительное значение агрегатной функции, аналогичное результату `COUNT(DISTINCT col)`. `ds_theta_count_distinct` быстрее, чем `COUNT(DISTINCT col)`, и использует меньше памяти для столбцов с высокой кардинальностью.

`ds_theta_count_distinct` похожа на `APPROX_COUNT_DISTINCT(expr)` и `DS_HLL_COUNT_DISTINCT(expr)`, но с другой точностью, поскольку использует Apache DataSketches. Для получения дополнительной информации см. Theta Sketches.

Относительная погрешность составляет 3.125% (95% доверительный интервал). Для получения дополнительной информации см. таблицу относительных погрешностей.

Синтаксис

```
BIGINT ds_theta_count_distinct(expr)
```

- `expr`: Столбец, в котором вычисляется приблизительное количество уникальных значений.

Примеры

```
mysql> CREATE TABLE t1 (
    ->   id BIGINT NOT NULL,
    ->   province VARCHAR(64),
    ->   age SMALLINT,
    ->   dt VARCHAR(10) NOT NULL
    -> )
    -> DUPLICATE KEY(id)
    -> DISTRIBUTED BY HASH(id) BUCKETS 4;
Query OK, 0 rows affected (0.02 sec)
mysql> insert into t1 select generate_series, generate_series,
generate_series % 100, "2024-07-24" from table(generate_series(1, 100000));
Query OK, 100000 rows affected (0.29 sec)
mysql> select ds_theta_count_distinct(id),
ds_theta_count_distinct(province), ds_theta_count_distinct(age),
ds_theta_count_distinct(dt) from t1 order by 1, 2;
+-----------------------------+-----------------------------------+--------
----------------------+-----------------------------+
| ds_theta_count_distinct(id) | ds_theta_count_distinct(province) |
ds_theta_count_distinct(age) | ds_theta_count_distinct(dt) |
+-----------------------------+-----------------------------------+--------
----------------------+-----------------------------+
|                      100215 |                            100846 |
100 |                           1 |
+-----------------------------+-----------------------------------+--------
----------------------+-----------------------------+
1 row in set (0.62 sec)
```

Ключевые слова

DS_THETA_COUNT_DISTINCT,DS_HLL_COUNT_DISTINCT,APPROX_COUNT_DISTINCT

Group_concat

Объединяет ненулевые значения из группы в одну строку с аргументом `sep`, который по умолчанию равен `,`, если не указан. Эта функция может использоваться для объединения значений из нескольких строк столбца в одну строку.

group_concat поддерживает DISTINCT и ORDER BY в версиях 3.0 новее 3.0.6 и версиях 3.1 новее 3.1.3.

Синтаксис

```
VARCHAR GROUP_CONCAT([DISTINCT] expr [,expr ...]
             [ORDER BY {unsigned_integer | col_name | expr}
                 [ASC | DESC] [,col_name ...]]
             [SEPARATOR sep])
```

Параметры

- `expr`: значения для объединения, нулевые значения игнорируются. Должен иметь тип VARCHAR. Вы можете опционально указать `DISTINCT` для удаления дублирующихся значений из выходной строки. Если вы хотите объединить несколько `expr` напрямую, используйте concat или concat_ws для указания форматов.
- Элементы в ORDER BY могут быть беззнаковыми целыми числами (начиная с 1), именами столбцов или обычными выражениями. Результаты сортируются по возрастанию по умолчанию. Вы также можете явно указать ключевое слово ASC. Если вы хотите отсортировать результаты по убыванию, добавьте ключевое слово DESC к имени столбца, по которому выполняется сортировка.
- `sep`: необязательный разделитель, используемый для объединения ненулевых значений из разных строк. Если он не указан, по умолчанию используется `,` (запятая). Чтобы удалить разделитель, укажите пустую строку `''`.

> **ПРИМЕЧАНИЕ**
>
> Начиная с v1.5.2, произошло изменение поведения при указании разделителя. Вы должны использовать `SEPARATOR` для объявления разделителя, например, `select group_concat(name SEPARATOR '-') as res from ss;`.

Возвращаемое значение

Возвращает строковое значение для каждой группы и возвращает NULL, если нет ненулевых значений.

Вы можете ограничить длину строки, возвращаемой group_concat, установив переменную сессии `group_concat_max_len`, которая по умолчанию равна 1024. Минимальное значение: 4. Единица измерения: символы.

Пример:

```
SET [GLOBAL | SESSION] group_concat_max_len = <value>;
```

Примеры

1. Создайте таблицу `ss`, которая содержит оценки по предметам.

```
CREATE TABLE `ss` (
  `id` int(11) NULL COMMENT "",
  `name` varchar(255) NULL COMMENT "",
  `subject` varchar(255) NULL COMMENT "",
  `score` int(11) NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 4
PROPERTIES (
"replication_num" = "1"
);

insert into ss values (1,"Tom","English",90);
insert into ss values (1,"Tom","Math",80);
insert into ss values (2,"Tom","English",NULL);
insert into ss values (2,"Tom",NULL,NULL);
insert into ss values (3,"May",NULL,NULL);
insert into ss values (3,"Ti","English",98);
insert into ss values (4,NULL,NULL,NULL);
insert into ss values (NULL,"Ti","Phy",98);

select * from ss order by id;
+------+------+---------+-------+
| id   | name | subject | score |
+------+------+---------+-------+
| NULL | Ti   | Phy     |    98 |
|    1 | Tom  | English |    90 |
|    1 | Tom  | Math    |    80 |
|    2 | Tom  | English |  NULL |
|    2 | Tom  | NULL    |  NULL |
|    3 | May  | NULL    |  NULL |
|    3 | Ti   | English |    98 |
|    4 | NULL | NULL    |  NULL |
+------+------+---------+-------+
```

2. Используйте group_concat.

Пример 1: Объедините имена в строку с разделителем по умолчанию, игнорируя нулевые значения. Дублирующиеся имена сохраняются.

```
select group_concat(name) as res from ss;
+---------------------------+
| res                       |
+---------------------------+
| Tom,Tom,Ti,Tom,Tom,May,Ti |
+---------------------------+
```

Пример 2: Объедините имена в строку, соединённые разделителем `-`, игнорируя нулевые значения. Дублирующиеся имена сохраняются.

```
select group_concat(name SEPARATOR '-') as res from ss;
+---------------------------+
| res                       |
+---------------------------+
| Ti-May-Ti-Tom-Tom-Tom-Tom |
+---------------------------+
```

Пример 3: Объедините уникальные имена в строку с разделителем по умолчанию, игнорируя нулевые значения. Дублирующиеся имена удаляются.

```
select group_concat(distinct name) as res from ss;
+---------------------------+
| res                       |
+---------------------------+
| Ti,May,Tom                |
+---------------------------+
```

Пример 4: Объедините строки имя-предмет одного и того же ID в порядке возрастания `score`. Например, `TomMath` и `TomEnglish` имеют общий ID 1, и они объединяются запятой в порядке возрастания `score`.

```
 select id, group_concat(distinct name,subject order by score) as res from
ss group by id order by id;
 +------+--------------------+
 | id   | res                |
 +------+--------------------+
 | NULL | TiPhy              |
 |    1 | TomMath,TomEnglish |
 |    2 | TomEnglish         |
 |    3 | TiEnglish          |
 |    4 | NULL               |
 +------+--------------------+
```

Пример 5: group_concat вложен с concat(), который используется для объединения `name`, `-` и `subject` в строку. Строки в одной и той же строке сортируются в порядке возрастания `score`.

```
 select id, group_concat(distinct concat(name, '-',subject) order by score)
as res from ss group by id order by id;
 +------+----------------------+
 | id   | res                  |
 +------+----------------------+
 | NULL | Ti-Phy               |
 |    1 | Tom-Math,Tom-English |
 |    2 | Tom-English          |
 |    3 | Ti-English           |
 |    4 | NULL                 |
 +------+----------------------+
```

Пример 6: Совпадающий результат не найден, и возвращается NULL.

```
select group_concat(distinct name) as res from ss where id < 0;
 +------+
 | res  |
 +------+
 | NULL |
 +------+
```

Пример 7: Ограничьте длину возвращаемой строки шестью символами.

```
 set group_concat_max_len = 6;

 select id, group_concat(distinct name,subject order by score) as res from
ss group by id order by id;
 +------+--------+
 | id   | res    |
 +------+--------+
 | NULL | TiPhy  |
 |    1 | TomMat |
 |    2 | NULL   |
 |    3 | TiEngl |
 |    4 | NULL   |
 +------+--------+
```

Ключевое слово

GROUP_CONCAT,CONCAT,ARRAY_AGG

Grouping

Указывает, является ли столбец агрегатным столбцом. Если это агрегатный столбец, возвращается 0. В противном случае возвращается 1.

Синтаксис

```
GROUPING(col_expr)
```

Параметры

`col_expr`: Выражение для столбца измерения, указанного в списке выражений расширения ROLLUP, CUBE или GROUPING SETS предложения GROUP BY.

Примеры

```
MySQL > select * from t;
+------+------+
| k1   | k2   |
+------+------+
| NULL | B    |
| NULL | b    |
| A    | NULL |
| A    | B    |
| A    | b    |
| a    | NULL |
| a    | B    |
| a    | b    |
+------+------+
8 rows in set (0.12 sec)

MySQL > SELECT k1, k2, GROUPING(k1), GROUPING(k2), COUNT(*) FROM t GROUP BY
ROLLUP(k1, k2);
+------+------+--------------+--------------+----------+
| k1   | k2   | grouping(k1) | grouping(k2) | count(*) |
+------+------+--------------+--------------+----------+
| NULL | NULL |            1 |            1 |        8 |
| NULL | NULL |            0 |            1 |        2 |
| NULL | B    |            0 |            0 |        1 |
| NULL | b    |            0 |            0 |        1 |
| A    | NULL |            0 |            1 |        3 |
| a    | NULL |            0 |            1 |        3 |
| A    | NULL |            0 |            0 |        1 |
| A    | B    |            0 |            0 |        1 |
| A    | b    |            0 |            0 |        1 |
| a    | NULL |            0 |            0 |        1 |
| a    | B    |            0 |            0 |        1 |
| a    | b    |            0 |            0 |        1 |
+------+------+--------------+--------------+----------+
12 rows in set (0.12 sec)
```

Grouping_id

grouping_id используется для различения результатов статистики группировки одного и того же стандарта группировки.

Синтаксис

```
GROUPING_ID(expr)
```

Примеры

```
MySQL > SELECT COL1,GROUPING_ID(COL2) AS 'GroupingID' FROM tbl GROUP BY
ROLLUP (COL1, COL2);
+------+------------+
| COL1 | GroupingID |
+------+------------+
| NULL |          1 |
| 2.20 |          1 |
| 2.20 |          0 |
| 1.10 |          1 |
| 1.10 |          0 |
+------+------------+
```

Hll_raw_agg

Эта функция является агрегатной функцией, которая используется для агрегации полей HLL. Она возвращает значение HLL.

Синтаксис

```
hll_raw_agg(hll)
```

Параметры

`hll`: столбец HLL, который генерируется другими столбцами или на основе загруженных данных.

Возвращаемое значение

Возвращает значение типа HLL.

Примеры

```
mysql> select k1, hll_cardinality(hll_raw_agg(v1)) from tbl group by k1;
+------+----------------------------------+
| k1   | hll_cardinality(hll_raw_agg(`v1`)) |
+------+----------------------------------+
|    2 |                                4 |
|    1 |                                3 |
+------+----------------------------------+
```

Hll_union

Возвращает объединение набора значений HLL.

Синтаксис

```
hll_union(hll)
```

Примеры

```
mysql> select k1, hll_cardinality(hll_union(v1)) from tbl group by k1;
+------+----------------------------------+
| k1   | hll_cardinality(hll_union(`v1`)) |
+------+----------------------------------+
|    2 |                                4 |
|    1 |                                3 |
+------+----------------------------------+
```

Hll_union_agg

HLL — это инженерная реализация на основе алгоритма HyperLogLog, которая используется для сохранения промежуточных результатов процесса вычисления HyperLogLog.

Может использоваться только как столбец значений таблицы и уменьшает объём данных посредством агрегации для достижения цели ускорения запросов.

Приблизительный результат с погрешностью около 1% на основе HLL. Столбец HLL генерируется другими столбцами или на основе данных, загруженных в таблицу.

При загрузке функция hll_hash используется для указания, какой столбец используется для генерации столбца HLL. Часто используется для замены Count Distinct и быстрого расчёта UV в бизнесе в сочетании с rollup.

Синтаксис

```
HLL_UNION_AGG(hll)
```

Примеры

```
MySQL > select HLL_UNION_AGG(uv_set) from test_uv;
+-------------------------+
| HLL_UNION_AGG(`uv_set`) |
+-------------------------+
| 17721                   |
+-------------------------+
```

Ключевое слово

HLL_UNION_AGG,HLL,UNION,AGG

Mann_whitney_u_test

`mann_whitney_u_test` выполняет ранговый критерий Манна-Уитни для выборок, полученных из двух генеральных совокупностей. Критерий Манна-Уитни U является непараметрическим критерием, который можно использовать для определения того, были ли две генеральные совокупности выбраны из одного и того же распределения.

Синтаксис

```
MANN_WHITNEY_U_TEST (sample_data, sample_treatment[, alternative[,
continuity_correction]])
```

Параметры

- `sample_data`: значение данных выборки. Должно быть числового типа данных.
- `sample_treatment`: индекс данных выборки, где каждый элемент указывает на группу обработки, к которой принадлежит соответствующая выборка. Значения должны быть

типа boolean, где `false` представляет первую группу, а `true` представляет вторую группу.

- `alternative` (опционально): строковая константа, указывающая альтернативную гипотезу. Может принимать одно из следующих значений:

-
  - 'two-sided': значение по умолчанию. Проверяет, различаются ли средние значения двух генеральных совокупностей.
-
  - 'less': проверяет, меньше ли среднее значение первой генеральной совокупности, чем среднее значение второй генеральной совокупности.
-
  - 'greater': проверяет, больше ли среднее значение первой генеральной совокупности, чем среднее значение второй генеральной совокупности.
- `continuity_correction` (опционально): булева константа, указывающая, следует ли применять поправку на непрерывность. Поправка на непрерывность корректирует статистику U на `0.5` в направлении среднего значения распределения U, что может повысить точность теста для малых размеров выборки. Значение по умолчанию — `true`.

Возвращаемое значение

Функция возвращает json-массив, содержащий следующие два элемента:

Статистику U Манна-Уитни и p-значение, связанное с тестом.

Примечания по использованию

Эта функция игнорирует NULL.

Примеры

Предположим, существует таблица с именем `testing_data` со следующими данными.

```
create table testing_data (
    id int,
    score int,
    treatment boolean
)
properties(
    "replication_num" = "1"
);

insert into testing_data values
    (1, 80, false),
    (2, 100, false),
    (3, NULL, false),
    (4, 60, true),
    (5, 70, true),
    (6, 85, true);
```

```
select * from testing_data;
+------+-------+-----------+
| id   | score | treatment |
+------+-------+-----------+
|    1 |    80 |         0 |
|    2 |   100 |         0 |
|    3 |  NULL |         0 |
|    4 |    60 |         1 |
|    5 |    70 |         1 |
|    6 |    85 |         1 |
+------+-------+-----------+
```

Запрос:

```
SELECT MANN_WHITNEY_U_TEST(score, treatment) FROM testing_data;
```

Результат:

```
+----------------------------------------+
| mann_whitney_u_test(score, treatment) |
+----------------------------------------+
| [5, 0.38647623077123283]               |
+----------------------------------------+
```

Запрос:

```
SELECT MANN_WHITNEY_U_TEST(score, treatment, 'less') FROM testing_data;
```

Результат:

```
+-----------------------------------------------+
| mann_whitney_u_test(score, treatment, 'less') |
+-----------------------------------------------+
| [5, 0.9255426634106172]                       |
+-----------------------------------------------+
```

Запрос:

```
SELECT MANN_WHITNEY_U_TEST(score, treatment, 'two-sided', 0) FROM
testing_data;
```

Результат:

```
+-------------------------------------------------------+
| mann_whitney_u_test(score, treatment, 'two-sided', 0) |
+-------------------------------------------------------+
| [5, 0.2482130789899235]                               |
+-------------------------------------------------------+
```

Max

Возвращает максимальное значение выражения expr.

Синтаксис

```
MAX(expr)
```

Примеры

```
MySQL > select max(scan_rows)
from log_statis
group by datetime;
+------------------+
| max(`scan_rows`) |
+------------------+
|          4671587 |
+------------------+
```

Ключевое слово

MAX

Max_by

Возвращает значение `x`, связанное с максимальным значением `y`.

Например, `SELECT max_by(subject, exam_result) FROM exam;` возвращает предмет с наивысшим баллом на экзамене.

Эта функция поддерживается начиная с версии 2.5.

Синтаксис

```
max_by(x,y)
```

Параметры

- `x`: выражение любого типа.
- `y`: выражение типа, который можно упорядочить.

Возвращаемое значение

Возвращает значение того же типа, что и `x`.

Примечания по использованию

- `y` должен быть сортируемым типом. Если вы используете несортируемый тип `y`, такой как `bitmap` или `hll`, возвращается ошибка.

- Если `y` содержит нулевое значение, строка, соответствующая нулевому значению, игнорируется.
- Если несколько значений `x` имеют одинаковое максимальное значение `y`, эта функция возвращает первое встретившееся значение `x`.

Примеры

1. Создайте таблицу `exam`.

```
CREATE TABLE exam (
    subject_id INT,
    subject STRING,
    exam_result INT
) DISTRIBUTED BY HASH(`subject_id`);
```

2. Вставьте значения в эту таблицу и запросите данные из неё.

```
insert into exam values
(1,'math',90),
(2,'english',70),
(3,'physics',95),
(4,'chemistry',85),
(5,'music',95),
(6,'biology',null);

select * from exam order by subject_id;
+------------+-----------+-------------+
| subject_id | subject   | exam_result |
+------------+-----------+-------------+
|          1 | math      |          90 |
|          2 | english   |          70 |
|          3 | physics   |          95 |
|          4 | chemistry |          85 |
|          5 | music     |          95 |
|          6 | biology   |        null |
+------------+-----------+-------------+
6 rows in set (0.03 sec)
```

3. Получите предмет с наивысшим баллом. Два предмета `physics` и `music` имеют одинаковый наивысший балл `95`, и возвращается первый встретившийся предмет (`physics`).

```
SELECT max_by(subject, exam_result) FROM exam;
+------------------------------+
| max_by(subject, exam_result) |
+------------------------------+
| physics                      |
+------------------------------+
1 row in set (0.01 sec)
```

Min

Возвращает минимальное значение выражения expr.

Синтаксис

```
MIN(expr)
```

Примеры

```
MySQL > select min(scan_rows)
from log_statis
group by datetime;
+------------------+
| min(`scan_rows`) |
+------------------+
|                0 |
+------------------+
```

Ключевое слово

MIN

Min_by

Возвращает значение `x`, связанное с минимальным значением `y`.

Например, `SELECT min_by(subject, exam_result) FROM exam;` возвращает предмет с наименьшим баллом на экзамене.

Эта функция поддерживается начиная с версии 2.5.

Синтаксис

```
min_by(x,y)
```

Параметры

- `x`: выражение любого типа.
- `y`: выражение типа, который можно упорядочить.

Возвращаемое значение

Возвращает значение того же типа, что и `x`.

Примечания по использованию

- `y` должен быть сортируемым типом. Если вы используете несортируемый тип `y`, такой как `bitmap` или `hll`, возвращается ошибка.
- Если `y` содержит нулевое значение, строка, соответствующая нулевому значению, игнорируется.
- Если несколько значений `x` имеют одинаковое минимальное значение `y`, эта функция возвращает первое встретившееся значение `x`.

Примеры

1. Создайте таблицу `exam`.

```sql
CREATE TABLE exam (
    subject_id INT,
    subject STRING,
    exam_result INT
) DISTRIBUTED BY HASH(`subject_id`);
```

2. Вставьте значения в эту таблицу и запросите данные из неё.

```
insert into exam values
(1,'math',90),
(2,'english',70),
(3,'physics',95),
(4,'chemistry',85),
(5,'music',95),
(6,'biology',null);

select * from exam order by subject_id;
+------------+-----------+-------------+
| subject_id | subject   | exam_result |
+------------+-----------+-------------+
|          1 | math      |          90 |
|          2 | english   |          70 |
|          3 | physics   |          95 |
|          4 | chemistry |          85 |
|          5 | music     |          95 |
|          6 | biology   |        null |
+------------+-----------+-------------+
6 rows in set (0.03 sec)
```

3. Получите предмет с наименьшим баллом. Возвращается предмет `english` с наименьшим баллом `70`.

```
SELECT min_by(subject, exam_result) FROM exam;
+------------------------------+
| min_by(subject, exam_result) |
+------------------------------+
| english                      |
+------------------------------+
1 row in set (0.01 sec)
```

Multi_distinct_count

Возвращает общее количество строк для `expr`, эквивалентно count(distinct expr).

Синтаксис

```
multi_distinct_count(expr)
```

Параметры

`expr`: столбец или выражение, на основе которого выполняется `multi_distinct_count()`. Если `expr` — имя столбца, столбец может быть любого типа данных.

Возвращаемое значение

Возвращает числовое значение. Если строки не найдены, возвращается 0. Эта функция игнорирует значения NULL.

Примеры

Предположим, есть таблица с именем `test`. Запросите категорию и поставщика каждого заказа по `id`.

```
select * from test order by id;
+------+----------+----------+------------+
| id   | country  | category | supplier   |
+------+----------+----------+------------+
| 1001 | US       | A        | supplier_1 |
| 1002 | Thailand | A        | supplier_2 |
| 1003 | Turkey   | B        | supplier_3 |
| 1004 | US       | A        | supplier_2 |
| 1005 | China    | C        | supplier_4 |
| 1006 | Japan    | D        | supplier_3 |
| 1007 | Japan    | NULL     | supplier_5 |
+------+----------+----------+------------+
```

Пример 1: Подсчитайте количество уникальных значений в столбце `category`.

```
select multi_distinct_count(category) from test;
+--------------------------------+
| multi_distinct_count(category) |
+--------------------------------+
|                              4 |
+--------------------------------+
```

Пример 2: Подсчитайте количество уникальных значений в столбце `supplier`.

```
select multi_distinct_count(supplier) from test;
+--------------------------------+
| multi_distinct_count(supplier) |
+--------------------------------+
|                              5 |
+--------------------------------+
```

Multi_distinct_sum

Возвращает сумму уникальных значений в `expr`, эквивалентно sum(distinct expr).

Синтаксис

```
multi_distinct_sum(expr)
```

Параметры

- `expr`: столбцы, участвующие в вычислении. Значения столбцов могут быть следующих типов: TINYINT, SMALLINT, INT, LARGEINT, FLOAT, DOUBLE или DECIMAL.

Возвращаемое значение

Соответствие между типами значений столбцов и типами возвращаемых значений:

- BOOLEAN -> BIGINT
- TINYINT -> BIGINT
- SMALLINT -> BIGINT
- INT -> BIGINT
- BIGINT -> BIGINT
- FLOAT -> DOUBLE
- DOUBLE -> DOUBLE
- LARGEINT -> LARGEINT
- DECIMALV2 -> DECIMALV2
- DECIMAL32 -> DECIMAL128
- DECIMAL64 -> DECIMAL128
- DECIMAL128 -> DECIMAL128

Примеры

1. Создайте таблицу с полем `k0` типа INT.

```
CREATE TABLE tabl
(k0 BIGINT NOT NULL) ENGINE=OLAP
DUPLICATE KEY(`k0`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`k0`)
PROPERTIES(
    "replication_num" = "3",
    "storage_format" = "DEFAULT"
);
```

2. Вставьте значения в таблицу.

```
--
INSERT INTO tabl VALUES ('0'), ('1'), ('1'), ('1'), ('2'), ('2');
```

3. Используйте multi_distinct_sum() для вычисления суммы уникальных значений в столбце `k0`.

```
MySQL > select multi_distinct_sum(k0) from tabl;
+------------------------+
| multi_distinct_sum(k0) |
+------------------------+
|                      3 |
+------------------------+
1 row in set (0.03 sec)
```

Уникальные значения `k0` — 0, 1, 2, и их сумма равна 3.

Percentile_approx

Возвращает приближённое значение p-го процентиля, где значение p находится между 0 и 1.

Параметр сжатия (compression) является необязательным и имеет диапазон настройки [2048, 10000]. Чем больше значение, тем выше точность, больше потребление памяти и

дольше время вычисления. Если параметр не указан или выходит за пределы диапазона [2048, 10000], функция выполняется с параметром сжатия по умолчанию 10000.

Эта функция использует фиксированный объём памяти, поэтому для столбцов с высокой кардинальностью можно использовать меньше памяти. Может использоваться для расчёта статистики, такой как tp99.

Синтаксис

```
DOUBLE PERCENTILE_APPROX(expr, DOUBLE p[, DOUBLE compression])
```

Примеры

```
MySQL > select `table`, percentile_approx(cost_time,0.99)
from log_statis
group by `table`;
+----------+-------------------------------------+
| table    | percentile_approx(`cost_time`, 0.99) |
+----------+-------------------------------------+
| test     |                                54.22 |
+----------+-------------------------------------+

MySQL > select `table`, percentile_approx(cost_time,0.99, 4096)
from log_statis
group by `table`;
+----------+-----------------------------------------------+
| table    | percentile_approx(`cost_time`, 0.99, 4096.0) |
+----------+-----------------------------------------------+
| test     |                                         54.21 |
+----------+-----------------------------------------------+
```

Ключевое слово

PERCENTILE_APPROX,PERCENTILE,APPROX

Percentile_approx_weight

Возвращает приближенное значение p-го процентиля с учётом веса. `percentile_approx_weight` — это взвешенная версия `PERCENTILE_APPROX`, которая позволяет пользователям указывать вес (константное значение или числовой столбец) для каждого входного значения.

Эта функция использует фиксированный объём памяти, поэтому для столбцов с высокой кардинальностью можно использовать меньше памяти. Она также может использоваться для вычисления таких статистик, как tp99.

Синтаксис

```
DOUBLE PERCENTILE_APPROX_WEIGHT(expr, BIGINT weight, DOUBLE p[, DOUBLE
compression])
```

- `expr`: столбец, для которого рассчитывается процентиль.
- `pth`: значение процентиля. Диапазон: [0, 1]. Например, `0.99` означает 99-й процентиль.
- `weight`: столбец веса. Должен быть положительным константным числом или столбцом.
- `compression`: (опционально) коэффициент сжатия. Диапазон: [2048, 10000]. Чем больше значение, тем выше точность, больше потребление памяти и больше время вычисления. Если этот параметр не указан или значение выходит за пределы диапазона [2048, 10000], используется значение по умолчанию `10000`.

Примеры

```
CREATE TABLE t1 (
    c1 int,
    c2 double,
    c3 tinyint,
    c4 int,
    c5 bigint,
    c6 largeint,
    c7 string,
    c8 double,
    c9 date,
    c10 datetime,
    c11 array<int>,
    c12 map<double, double>,
    c13 struct<a bigint, b double>
    )
DUPLICATE KEY(c1)
DISTRIBUTED BY HASH(c1)
BUCKETS 1
PROPERTIES ("replication_num" = "1");
insert into t1
    select generate_series, generate_series,  11, 111, 1111, 11111,
"111111", 1.1, "2024-09-01", "2024-09-01 18:00:00", [1, 2, 3], map(1, 5.5),
row(100, 100)
    from table(generate_series(1, 50000, 3));
-- использование константного значения в качестве веса
mysql> select percentile_approx_weighted(c1, 1, 0.9) from t1;
+----------------------------------------+
| percentile_approx_weighted(c1, 1, 0.9) |
+----------------------------------------+
|                         45000.39453125 |
+----------------------------------------+
1 row in set (0.07 sec)
-- использование числового столбца в качестве веса
mysql> select percentile_approx_weighted(c2, c1, 0.5) from t1;
+----------------------------------------+
| percentile_approx_weighted(c2, c1, 0.5) |
+----------------------------------------+
|                               35355.9375 |
+----------------------------------------+
1 row in set (0.07 sec)
-- использование веса и сжатия для вычисления процентиля
mysql> select percentile_approx_weighted(c2, c1, 0.5, 10000) from t1;
+-------------------------------------------------+
```

```
| percentile_approx_weighted(c2, c1, 0.5, 10000) |
+------------------------------------------------+
|                                     35355.9375 |
+------------------------------------------------+
1 row in set (0.09 sec)
```

Keywords

PERCENTILE_APPROX_WEIGHT,PERCENTILE_APPROX,PERCENTILE,APPROX

Percentile_cont

Вычисляет процентильное значение `expr` с линейной интерполяцией.

Синтаксис

```
PERCENTILE_CONT (expr, percentile)
```

Параметры

- `expr`: выражение, по которому упорядочиваются значения. Должно быть числовым типом данных, DATE или DATETIME. Например, если вы хотите найти медианный балл по физике, укажите столбец, содержащий баллы по физике.

- `percentile`: процентиль искомого значения. Это константа с плавающей точкой от 0 до 1. Например, если вы хотите найти медианное значение, установите этот параметр равным `0.5`.

Возвращаемое значение

Возвращает значение, которое находится на указанном процентиле. Если ни одно входное значение не находится точно на желаемом процентиле, результат вычисляется с использованием линейной интерполяции двух ближайших входных значений.

Тип данных совпадает с `expr`.

Примечания по использованию

Эта функция игнорирует значения NULL.

Примеры

Предположим, есть таблица с именем `exam` со следующими данными.

```
select * from exam order by Subject;
+-----------+-------+
| Subject   | Score |
+-----------+-------+
| chemistry |    80 |
| chemistry |   100 |
| chemistry |  NULL |
| math      |    60 |
| math      |    70 |
| math      |    85 |
| physics   |    75 |
| physics   |    80 |
| physics   |    85 |
| physics   |    99 |
+-----------+-------+
```

Рассчитайте медианный балл по каждому предмету, игнорируя значения NULL.

Запрос:

```
SELECT Subject, PERCENTILE_CONT (Score, 0.5)  FROM exam group by Subject;
```

Результат:

```
+-----------+-----------------------------+
| Subject   | percentile_cont(Score, 0.5) |
+-----------+-----------------------------+
| chemistry |                          90 |
| math      |                          70 |
| physics   |                        82.5 |
+-----------+-----------------------------+
```

Percentile_disc

Возвращает процентильное значение на основе дискретного распределения входного столбца `expr`. Если точное процентильное значение не может быть найдено, эта функция

возвращает большее значение из двух ближайших значений.

Эта функция поддерживается начиная с версии 2.5.

Синтаксис

```
PERCENTILE_DISC (expr, percentile)
```

Параметры

- `expr`: столбец, для которого вы хотите рассчитать процентильное значение. Столбец может быть любого типа данных, который поддерживает сортировку.
- `percentile`: процентиль искомого значения. Должно быть константой с плавающей точкой от 0 до 1. Например, если вы хотите найти медианное значение, установите этот параметр равным `0.5`. Если вы хотите найти значение на 70-м процентиле, укажите 0.7.

Возвращаемое значение

Тип данных возвращаемого значения совпадает с `expr`.

Примечания по использованию

Значения NULL игнорируются при расчёте.

Примеры

Создайте таблицу `exam` и вставьте данные в эту таблицу.

```
CREATE TABLE exam (
    subject STRING,
    score INT
)
DISTRIBUTED BY HASH(`subject`);

INSERT INTO exam VALUES
('chemistry',80),
('chemistry',100),
('chemistry',null),
('math',60),
('math',70),
('math',85),
('physics',75),
('physics',80),
('physics',85),
('physics',99);
```

```
select * from exam order by subject;
+-----------+-------+
| subject   | score |
+-----------+-------+
| chemistry |    80 |
| chemistry |   100 |
| chemistry |  NULL |
| math      |    60 |
| math      |    70 |
| math      |    85 |
| physics   |    75 |
| physics   |    80 |
| physics   |    85 |
| physics   |    99 |
+-----------+-------+
```

Рассчитайте медиану по каждому предмету.

```
select subject, percentile_disc(score, 0.5)
from exam group by subject;
```

Вывод

```
+-----------+----------------------------+
| subject   | percentile_disc(score, 0.5) |
+-----------+----------------------------+
| chemistry |                        100 |
| math      |                         70 |
| physics   |                         85 |
+-----------+----------------------------+
```

Percentile_disc_lc

Возвращает процентильное значение на основе дискретного распределения входного столбца `expr`. Поведение аналогично percentile_disc. Однако алгоритм реализации отличается. percentile_disc требует получения всех входных данных, и память, потребляемая при сортировке слиянием для получения процентильных значений, равна памяти всех входных данных. С другой стороны, percentile_disc_lc строит хеш-таблицу key->count, поэтому при низкой кардинальности входных данных не происходит заметного увеличения памяти, даже если размер входных данных велик.

Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
PERCENTILE_DISC_LC (expr, percentile)
```

Параметры

- `expr`: столбец, для которого вы хотите рассчитать процентильное значение. Столбец может быть любого типа данных, который поддерживает сортировку.
- `percentile`: процентиль искомого значения. Должно быть константой с плавающей точкой от 0 до 1. Например, если вы хотите найти медианное значение, установите этот параметр равным `0.5`. Если вы хотите найти значение на 70-м процентиле, укажите 0.7.

Возвращаемое значение

Тип данных возвращаемого значения совпадает с `expr`.

Примечания по использованию

Значения NULL игнорируются при расчёте.

Примеры

Создайте таблицу `exam` и вставьте данные в эту таблицу.

```
CREATE TABLE exam (
    subject STRING,
    score INT
)
DISTRIBUTED BY HASH(`subject`);

INSERT INTO exam VALUES
('chemistry',80),
('chemistry',100),
('chemistry',null),
('math',60),
('math',70),
('math',85),
('physics',75),
('physics',80),
('physics',85),
('physics',99);
```

```
select * from exam order by subject;
+-----------+-------+
| subject   | score |
+-----------+-------+
| chemistry |    80 |
| chemistry |   100 |
| chemistry |  NULL |
| math      |    60 |
| math      |    70 |
| math      |    85 |
| physics   |    75 |
| physics   |    80 |
| physics   |    85 |
| physics   |    99 |
+-----------+-------+
```

Рассчитайте медиану по каждому предмету.

```
select subject, percentile_disc_lc (score, 0.5)
from exam group by subject;
```

Вывод

```
+-----------+--------------------------------+
| subject   | percentile_disc_lc(score, 0.5) |
+-----------+--------------------------------+
| physics   |                             85 |
| chemistry |                            100 |
| math      |                             70 |
+-----------+--------------------------------+
```

Retention

Вычисляет коэффициент удержания пользователей за указанный период времени. Эта функция принимает от 1 до 31 условия и проверяет, истинно ли каждое условие. Если условие истинно, возвращается 1. В противном случае возвращается 0. В итоге возвращается массив из 0 и 1. Вы можете рассчитать коэффициент удержания пользователей на основе этих данных.

Синтаксис

```
ARRAY retention(ARRAY input)
```

Параметры

`input`: массив условий. Можно передать максимум 31 условие. Разделяйте несколько условий запятыми.

Возвращаемое значение

Возвращает массив из 0 и 1. Количество элементов 0 и 1 равно количеству входных условий.

Вычисление начинается с первого условия.

- Если условие истинно, возвращается 1. В противном случае возвращается 0.

- Если первое условие не истинно, текущая позиция и все последующие позиции устанавливаются в 0.

Примеры

1. Создайте таблицу с именем `test` и вставьте данные.

```
CREATE TABLE test(
    id TINYINT,
    action STRING,
    time DATETIME
)
ENGINE=olap
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id);

INSERT INTO test VALUES
(1,'pv','2022-01-01 08:00:05'),
(2,'pv','2022-01-01 10:20:08'),
(1,'buy','2022-01-02 15:30:10'),
(2,'pv','2022-01-02 17:30:05'),
(3,'buy','2022-01-01 05:30:09'),
(3,'buy','22022-01-02 08:10:15'),
(4,'pv','2022-01-02 21:09:15'),
(5,'pv','2022-01-01 22:10:53'),
(5,'pv','2022-01-02 19:10:52'),
(5,'buy','2022-01-02 20:00:50');
```

2. Запросите данные из `test`.

```
MySQL > select * from test order by id;
+------+--------+---------------------+
| id   | action | time                |
+------+--------+---------------------+
|    1 | pv     | 2022-01-01 08:00:05 |
|    1 | buy    | 2022-01-02 15:30:10 |
|    2 | pv     | 2022-01-01 10:20:08 |
|    2 | pv     | 2022-01-02 17:30:05 |
|    3 | buy    | 2022-01-01 05:30:09 |
|    3 | buy    | 2022-01-02 08:10:15 |
|    4 | pv     | 2022-01-02 21:09:15 |
|    5 | pv     | 2022-01-01 22:10:53 |
|    5 | pv     | 2022-01-02 19:10:52 |
|    5 | buy    | 2022-01-02 20:00:50 |
+------+--------+---------------------+
10 rows in set (0.01 sec)
```

3. Используйте `retention` для вычисления коэффициента удержания пользователей.

   Пример 1: Проверьте поведение пользователей по следующим условиям: просмотр страницы товара 2022-01-01 (action='pv') и оформление заказа 2022-01-02 (action='buy').

```
MySQL > select id, retention([action='pv' and to_date(time)='2022-01-
01',
                              action='buy' and to_date(time)='2022-01-
02']) as retention
from test
group by id
order by id;

+------+-----------+
| id   | retention |
+------+-----------+
|    1 | [1,1]     |
|    2 | [1,0]     |
|    3 | [0,0]     |
|    4 | [0,0]     |
|    5 | [1,1]     |
+------+-----------+
5 rows in set (0.01 sec)
```

В результате:

- Пользователи 1 и 5 выполнили оба условия, возвращается [1,1].
- Пользователь 2 не выполнил второе условие, возвращается [1,0].
- Пользователь 3 выполнил второе условие, но не выполнил первое. Возвращается [0,0].
- Пользователь 4 не выполнил ни одного условия, возвращается [0,0].

Пример 2: Рассчитайте процент пользователей, которые просмотрели страницу товара 2022-01-01 (action='pv') и оформили заказ 2022-01-02 (action='buy').

```
MySQL > select sum(r[1]),sum(r[2])/sum(r[1])
from (select id, retention([action='pv' and to_date(time)='2022-01-01',
                            action='buy' and to_date(time)='2022-01-
02']) as r
from test
group by id
order by id) t;

+-----------+---------------------------+
| sum(r[1]) | (sum(r[2])) / (sum(r[1])) |
+-----------+---------------------------+
|         3 |        0.6666666666666666 |
+-----------+---------------------------+
1 row in set (0.02 sec)
```

Возвращаемое значение — коэффициент удержания пользователей на 2022-01-02.

Ключевое слово

retention, retention rate, RETENTION

Std

Возвращает стандартное отклонение выражения. Начиная с версии 2.5.10, эта функция также может использоваться как оконная функция.

Синтаксис

```
STD(expr)
```

Параметры

`expr`: выражение. Если это столбец таблицы, он должен вычисляться в TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE или DECIMAL.

Возвращаемое значение

Возвращает значение типа DOUBLE.

Примеры

```
MySQL > select * from std_test;
+------+------+
| col0 | col1 |
+------+------+
|    0 |    0 |
|    1 |    2 |
|    2 |    4 |
|    3 |    6 |
|    4 |    8 |
+------+------+
```

Рассчитайте стандартное отклонение `col0` и `col1`.

```
MySQL > select std(col0) as std_of_col0, std(col1) as std_of_col1 from
std_test;
+--------------------+--------------------+
| std_of_col0        | std_of_col1        |
+--------------------+--------------------+
| 1.4142135623730951 | 2.8284271247461903 |
+--------------------+--------------------+
```

Ключевое слово

STD

Stddev,stddev_pop,std

Возвращает стандартное отклонение генеральной совокупности выражения expr. Начиная с версии 2.5.10, эта функция также может использоваться как оконная функция.

Синтаксис

```
STDDEV(expr)
```

Параметры

`expr`: выражение. Если это столбец таблицы, он должен вычисляться в TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE или DECIMAL.

Возвращаемое значение

Возвращает значение типа DOUBLE. Формула следующая, где `n` представляет количество строк таблицы:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

Примеры

```
mysql> SELECT stddev(lo_quantity), stddev_pop(lo_quantity) from lineorder;
+----------------------+-------------------------+
| stddev(lo_quantity) | stddev_pop(lo_quantity) |
+----------------------+-------------------------+
|   14.43100708360797 |       14.43100708360797 |
+----------------------+-------------------------+
```

Смотрите также

stddev_samp

Ключевое слово

STDDEV,STDDEV_POP,POP

Stddev_samp

Возвращает выборочное стандартное отклонение выражения. Начиная с версии 2.5.10, эта функция также может использоваться как оконная функция.

Синтаксис

```
STDDEV_SAMP(expr)
```

Параметры

`expr`: выражение. Если это столбец таблицы, он должен вычисляться в TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE или DECIMAL.

Возвращаемое значение

Возвращает значение типа DOUBLE. Формула следующая, где `n` представляет количество строк таблицы:

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Примеры

```
MySQL > select stddev_samp(scan_rows)
from log_statis
group by datetime;
+--------------------------+
| stddev_samp(`scan_rows`) |
+--------------------------+
|        2.372044195280762 |
+--------------------------+
```

Смотрите также

stddev

Ключевое слово

STDDEV_SAMP,STDDEV,SAMP

Sum

Возвращает сумму ненулевых значений для `expr`. Вы можете использовать ключевое слово DISTINCT для вычисления суммы уникальных ненулевых значений.

Синтаксис

```
SUM([DISTINCT] expr)
```

Параметры

`expr`: выражение, которое вычисляется в числовое значение. Поддерживаемые типы данных: TINYINT, SMALLINT, INT, FLOAT, DOUBLE или DECIMAL.

Возвращаемое значение

Соответствие типов данных между входным значением и возвращаемым значением:

- TINYINT -> BIGINT
- SMALLINT -> BIGINT
- INT -> BIGINT
- FLOAT -> DOUBLE

- DOUBLE -> DOUBLE
- DECIMAL -> DECIMAL

Примечания по использованию

- Эта функция игнорирует нулевые значения.
- Возвращается ошибка, если `expr` не существует.
- Если передано выражение VARCHAR, эта функция неявно преобразует входные данные в значения DOUBLE. Если преобразование не удается, возвращается ошибка.

Примеры

1. Создайте таблицу с именем `employees`.

```
CREATE TABLE IF NOT EXISTS employees (
    region_num    TINYINT        COMMENT "range [-128, 127]",
    id            BIGINT         COMMENT "range [-2^63 + 1 ~ 2^63 -
1]",
    hobby         STRING         NOT NULL COMMENT "upper limit value
65533 bytes",
    income        DOUBLE         COMMENT "8 bytes",
    sales       DECIMAL(12,4)  COMMENT ""
    )
    DISTRIBUTED BY HASH(region_num);
```

2. Вставьте данные в `employees`.

```
INSERT INTO employees VALUES
(3,432175,'3',25600,1250.23),
(4,567832,'3',37932,2564.33),
(3,777326,'2',null,1932.99),
(5,342611,'6',43727,45235.1),
(2,403882,'4',36789,52872.4);
```

3. Запросите данные из `employees`.

```
MySQL > select * from employees;
+------------+--------+-------+--------+------------+
| region_num | id     | hobby | income | sales      |
+------------+--------+-------+--------+------------+
|          5 | 342611 | 6     |  43727 | 45235.1000 |
|          2 | 403882 | 4     |  36789 | 52872.4000 |
|          4 | 567832 | 3     |  37932 |  2564.3300 |
|          3 | 432175 | 3     |  25600 |  1250.2300 |
|          3 | 777326 | 2     |   NULL |  1932.9900 |
+------------+--------+-------+--------+------------+
5 rows in set (0.01 sec)
```

4. Используйте эту функцию для вычисления суммы.

   Пример 1: Рассчитайте общие продажи каждого региона.

```
MySQL > SELECT region_num, sum(sales) from employees
group by region_num;

+------------+------------+
| region_num | sum(sales) |
+------------+------------+
|          2 | 52872.4000 |
|          5 | 45235.1000 |
|          4 |  2564.3300 |
|          3 |  3183.2200 |
+------------+------------+
4 rows in set (0.01 sec)
```

   Пример 2: Рассчитайте общий доход сотрудников каждого региона. Эта функция игнорирует нулевые значения, и доход сотрудника с id `777326` не учитывается.

```
MySQL > select region_num, sum(income) from employees
group by region_num;

+------------+-------------+
| region_num | sum(income) |
+------------+-------------+
|          2 |       36789 |
|          5 |       43727 |
|          4 |       37932 |
|          3 |       25600 |
+------------+-------------+
4 rows in set (0.01 sec)
```

Пример 3: Рассчитайте общее количество хобби. Столбец `hobby` имеет тип STRING и будет неявно преобразован в DOUBLE во время вычисления.

```
MySQL > select sum(DISTINCT hobby) from employees;

+---------------------+
| sum(DISTINCT hobby) |
+---------------------+
|                  15 |
+---------------------+
1 row in set (0.01 sec)
```

Пример 4: Используйте `sum` с предложением WHERE для расчёта общего дохода сотрудников, чей ежемесячный доход превышает 30000.

```
MySQL > select sum(income) from employees
WHERE income > 30000;

+-------------+
| sum(income) |
+-------------+
|      118448 |
+-------------+
1 row in set (0.00 sec)
```

Ключевое слово

SUM, sum

Var_samp,variance_samp

Возвращает выборочную дисперсию выражения. Начиная с версии 2.5.10, эта функция также может использоваться как оконная функция.

Синтаксис

```
VAR_SAMP(expr)
```

Параметры

`expr`: выражение. Если это столбец таблицы, он должен вычисляться в TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE или DECIMAL.

Возвращаемое значение

Возвращает значение типа DOUBLE.

Примеры

```
MySQL > select var_samp(scan_rows)
from log_statis
group by datetime;
+------------------------+
| var_samp(`scan_rows`) |
+------------------------+
|    5.6227132145741789 |
+------------------------+
```

Ключевое слово

VAR_SAMP,VARIANCE_SAMP,VAR,SAMP,VARIANCE

Variance,var_pop,variance_pop

Возвращает дисперсию генеральной совокупности выражения. Начиная с версии 2.5.10, эта функция также может использоваться как оконная функция.

Синтаксис

```
VARIANCE(expr)
```

Параметры

`expr`: выражение. Если это столбец таблицы, он должен вычисляться в TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE или DECIMAL.

Возвращаемое значение

Возвращает значение типа DOUBLE.

Примеры

```
MySQL > select var_pop(i_current_price), i_rec_start_date from item group
by i_rec_start_date;
+--------------------------+------------------+
| var_pop(i_current_price) | i_rec_start_date |
+--------------------------+------------------+
|       314.96177792808226 | 1997-10-27       |
|       463.73633459357285 | NULL             |
|       302.02102643609123 | 1999-10-28       |
|        337.9318386924913 | 2000-10-27       |
|       333.80931439318346 | 2001-10-27       |
+--------------------------+------------------+

MySQL > select variance(i_current_price), i_rec_start_date from item group
by i_rec_start_date;
+---------------------------+------------------+
| variance(i_current_price) | i_rec_start_date |
+---------------------------+------------------+
|        314.96177792808226 | 1997-10-27       |
|         463.7363345935729 | NULL             |
|        302.02102643609123 | 1999-10-28       |
|         337.9318386924912 | 2000-10-27       |
|        333.80931439318346 | 2001-10-27       |
+---------------------------+------------------+
```

Ключевое слово

VARIANCE,VAR_POP,VARIANCE_POP

Window_funnel

Ищет цепочку событий в скользящем окне и вычисляет максимальное количество последовательных событий в цепочке событий. Эта функция обычно используется для анализа конверсии. Поддерживается начиная с версии 2.3.

Эта функция работает согласно следующим правилам:

- Она начинает подсчёт с первого события в цепочке событий. Если первое событие найдено, счётчик событий устанавливается в 1 и скользящее окно начинается. Если первое событие не найдено, возвращается 0.
- В скользящем окне счётчик увеличивается, если события в цепочке событий происходят последовательно. Если скользящее окно превышено, счётчик событий больше не увеличивается.
- Если несколько цепочек событий соответствуют указанному условию, возвращается самая длинная цепочка событий.

Синтаксис

```
BIGINT window_funnel(BIGINT window, DATE|DATETIME time, INT mode,
array[cond1, cond2, ..., condN])
```

Параметры

- `window`: Длина скользящего окна. Поддерживаемый тип данных — BIGINT. Единица измерения зависит от параметра `time`. Если тип данных `time` — DATE, единица измерения — дни. Если тип данных `time` — DATETIME, единица измерения — секунды.
- `time`: Столбец, содержащий временные метки. Поддерживаются типы DATE и DATETIME.
- `mode`: Режим фильтрации цепочки событий. Поддерживаемый тип данных — INT. Диапазон значений: 0, 1, 2.
    - `0` — значение по умолчанию, указывает на обычное вычисление воронки.

- `1` указывает на режим `DEDUPLICATION`, то есть отфильтрованная цепочка событий не может содержать повторяющиеся события. Предположим, параметр `array` равен `[event_type = 'A', event_type = 'B', event_type = 'C', event_type = 'D']`, а исходная цепочка событий — "A-B-C-B-D". Событие B повторяется, и отфильтрованная цепочка событий — "A-B-C".
- `2` указывает на режим `FIXED`, то есть отфильтрованная цепочка событий не может содержать события, нарушающие указанную последовательность. Предположим, используется предыдущий параметр `array`, а исходная цепочка событий — "A-B-D-C". Событие D нарушает последовательность, и отфильтрованная цепочка событий — "A-B".
- `4` указывает на режим `INCREASE`, что означает, что отфильтрованные события должны иметь строго возрастающие временные метки. Дублирующиеся временные метки нарушают цепочку событий. Этот режим поддерживается начиная с версии 2.5.

- `array`: Определённая цепочка событий. Должна быть массивом.

Возвращаемое значение

Возвращает значение типа BIGINT.

Примеры

Пример 1: Вычислить максимальное количество последовательных событий на основе `uid`. Скользящее окно — 1800 секунд, режим фильтрации — `0`.

В этом примере используется таблица `action`, в которой данные отсортированы по `uid`.

```
mysql> select * from action;
+------+------------+---------------------+
| uid  | event_type | time                |
+------+------------+---------------------+
| 1    | Browse     | 2020-01-02 11:00:00 |
| 1    | Click      | 2020-01-02 11:10:00 |
| 1    | Order      | 2020-01-02 11:20:00 |
| 1    | Pay        | 2020-01-02 11:30:00 |
| 1    | Browse     | 2020-01-02 11:00:00 |
| 2    | Order      | 2020-01-02 11:00:00 |
| 2    | Pay        | 2020-01-02 11:10:00 |
| 3    | Browse     | 2020-01-02 11:20:00 |
| 3    | Click      | 2020-01-02 12:00:00 |
| 4    | Browse     | 2020-01-02 11:50:00 |
| 4    | Click      | 2020-01-02 12:00:00 |
| 5    | Browse     | 2020-01-02 11:50:00 |
| 5    | Click      | 2020-01-02 12:00:00 |
| 5    | Order      | 2020-01-02 11:10:00 |
| 6    | Browse     | 2020-01-02 11:50:00 |
| 6    | Click      | 2020-01-02 12:00:00 |
| 6    | Order      | 2020-01-02 12:10:00 |
+------+------------+---------------------+
17 rows in set (0.01 sec)
```

Выполните следующий запрос:

```
select uid,
       window_funnel(1800,time,0,[event_type='Browse', event_type='Click',
        event_type='Order', event_type='Pay']) AS level
from action
group by uid
order by uid;
+------+-------+
| uid  | level |
+------+-------+
| 1    |     4 |
| 2    |     0 |
| 3    |     1 |
| 4    |     2 |
| 5    |     2 |
| 6    |     3 |
+------+-------+
```

Описание результатов:

- Соответствующая цепочка событий для `uid = 1` — "Browse-Click-Order-Pay", возвращается `4`. Время последнего события "Browse" (2020-01-02 11:00:00) не соответствует условию и не учитывается.
- Цепочка событий для `uid = 2` не начинается с первого события "Browse", возвращается `0`.
- Соответствующая цепочка событий для `uid = 3` — "Browse", возвращается `1`. Событие "Click" превышает временное окно в 1800 секунд и не учитывается.
- Соответствующая цепочка событий для `uid = 4` — "Browse-Click", возвращается `2`.
- Соответствующая цепочка событий для `uid = 5` — "Browse-Click", возвращается `2`. Событие "Order" (2020-01-02 11:10:00) не принадлежит цепочке событий и не учитывается.
- Соответствующая цепочка событий для `uid = 6` — "Browse-Click-Order", возвращается `3`.

Пример 2: Вычислить максимальное количество последовательных событий на основе `uid`. Скользящее окно — 1800 секунд, используются режимы фильтрации `0` и `1`.

В этом примере используется таблица `action1`, в которой данные отсортированы по `time`.

```
mysql> select * from action1 order by time;
+------+------------+---------------------+
| uid  | event_type | time                |
+------+------------+---------------------+
| 1    | Browse     | 2020-01-02 11:00:00 |
| 2    | Browse     | 2020-01-02 11:00:01 |
| 1    | Click      | 2020-01-02 11:10:00 |
| 1    | Order      | 2020-01-02 11:29:00 |
| 1    | Click      | 2020-01-02 11:29:50 |
| 1    | Pay        | 2020-01-02 11:30:00 |
| 1    | Click      | 2020-01-02 11:40:00 |
+------+------------+---------------------+
7 rows in set (0.03 sec)
```

Выполните следующий запрос:

```
select uid,
       window_funnel(1800,time,0,[event_type='Browse',
        event_type='Click', event_type='Order', event_type='Pay']) AS level
from action1
group by uid
order by uid;
+------+-------+
| uid  | level |
+------+-------+
| 1    |     4 |
| 2    |     1 |
+------+-------+
2 rows in set (0.02 sec)
```

Для `uid = 1` событие "Click" (2020-01-02 11:29:50) является повторяющимся событием, но оно всё равно учитывается, поскольку используется режим `0`. Поэтому возвращается `4`.

Измените `mode` на `1` и выполните запрос снова.

```
+------+-------+
| uid  | level |
+------+-------+
| 1    |     3 |
| 2    |     1 |
+------+-------+
2 rows in set (0.05 sec)
```

Самая длинная цепочка событий после дедупликации — "Browse-Click-Order", возвращается `3`.

Пример 3: Вычислить максимальное количество последовательных событий на основе `uid`. Скользящее окно — 1900 секунд, используются режимы фильтрации `0` и `2`.

В этом примере используется таблица `action2`, в которой данные отсортированы по `time`.

```
mysql> select * from action2 order by time;
+------+------------+---------------------+
| uid  | event_type | time                |
+------+------------+---------------------+
| 1    | Browse     | 2020-01-02 11:00:00 |
| 2    | Browse     | 2020-01-02 11:00:01 |
| 1    | Click      | 2020-01-02 11:10:00 |
| 1    | Pay        | 2020-01-02 11:30:00 |
| 1    | Order      | 2020-01-02 11:31:00 |
+------+------------+---------------------+
5 rows in set (0.01 sec)
```

Выполните следующий запрос:

```
select uid,
       window_funnel(1900,time,0,[event_type='Browse', event_type='Click',
        event_type='Order', event_type='Pay']) AS level
from action2
group by uid
order by uid;
+------+-------+
| uid  | level |
+------+-------+
| 1    |     3 |
| 2    |     1 |
+------+-------+
2 rows in set (0.02 sec)
```

Для `uid = 1` возвращается `3`, поскольку используется режим `0`, и событие "Pay" (2020-01-02 11:30:00) не нарушает цепочку событий.

Измените `mode` на `2` и выполните запрос снова.

```
select uid,
       window_funnel(1900,time,2,[event_type='Browse', event_type='Click',
        event_type='Order', event_type='Pay']) AS level
from action2
group by uid
order by uid;
+------+-------+
| uid  | level |
+------+-------+
| 1    |     2 |
| 2    |     1 |
+------+-------+
2 rows in set (0.06 sec)
```

Возвращается `2`, поскольку событие "Pay" нарушает цепочку событий, и счётчик событий останавливается. Отфильтрованная цепочка событий — "Browse-Click".

Пример 4: Вычислить максимальное количество последовательных событий на основе `uid`. Скользящее окно — 1900 секунд, используются режимы фильтрации `0` и `4`.

В этом примере используется таблица `action3`, в которой данные отсортированы по `time`.

```
select * from action3 order by time;
+------+------------+---------------------+
| uid  | event_type | time                |
+------+------------+---------------------+
| 1    | Browse     | 2020-01-02 11:00:00 |
| 1    | Click      | 2020-01-02 11:00:01 |
| 2    | Browse     | 2020-01-02 11:00:03 |
| 1    | Order      | 2020-01-02 11:00:31 |
| 2    | Click      | 2020-01-02 11:00:03 |
| 2    | Order      | 2020-01-02 11:01:03 |
+------+------------+---------------------+
3 rows in set (0.02 sec)
```

Выполните следующий запрос:

```
select uid,
       window_funnel(1900,time,0,[event_type='Browse', event_type='Click',
        event_type='Order']) AS level
from action3
group by uid
order by uid;
+------+-------+
| uid  | level |
+------+-------+
|    1 |     3 |
|    2 |     3 |
+------+-------+
```

Для `uid = 1` и `uid = 2` возвращается `3`.

Измените `mode` на `4` и выполните запрос снова.

```
select uid,
       window_funnel(1900,time,4,[event_type='Browse', event_type='Click',
        event_type='Order']) AS level
from action3
group by uid
order by uid;
+------+-------+
| uid  | level |
+------+-------+
|    1 |     3 |
|    2 |     1 |
+------+-------+
1 row in set (0.02 sec)
```

Для `uid = 2` возвращается `1`, поскольку используется режим `4` (строго возрастающий). "Click" происходит в ту же секунду, что и "BROWSE". Поэтому "Click" и "Order" не учитываются.

Ключевые слова

window funnel, funnel, window_funnel

2.8 Array

All_match

Возвращает, соответствуют ли все элементы массива заданному предикату.

- Возвращает `true` (1), если все элементы соответствуют предикату (особый случай — когда массив пуст).
- Возвращает `false` (0), если один или несколько элементов не соответствуют.
- Возвращает NULL, если предикат возвращает NULL для одного или нескольких элементов и `true` для всех остальных элементов.

Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
all_match(lambda_function, arr1, arr2...)
```

Возвращает, соответствуют ли все элементы `arr1` заданному предикату в lambda-функции.

Параметры

- `arr1`: массив для проверки соответствия.
- `arrN`: дополнительные массивы, используемые в lambda-функции.
- `lambda_function`: lambda-функция, используемая для проверки соответствия значений.

Возвращаемое значение

Возвращает значение типа BOOLEAN.

Примечания по использованию

- Lambda-функция следует примечаниям по использованию в array_map().
- Если входной массив равен null или lambda-функция возвращает null, возвращается null.

- Если `arr1` пуст, возвращается `true`.
- Чтобы применить эту функцию к MAP, перепишите `all_match((k,v)->k>v,map)` в `all_match(map_values(transform_values((k,v)->k>v, map)))`. Например, `select all_match(map_values(transform_values((k,v)->k>v, map{2:1})));` возвращает 1.

Примеры

Проверка, являются ли все элементы в `x` меньше элементов в `y`.

```
select all_match((x,y) -> x < y, [1,2,-8], [4,5,6]);
+-----------------------------------------------------+
| all_match((x, y) -> x < y, [1, 2, -8], [4, 5, 6]) |
+-----------------------------------------------------+
|                                                   1 |
+-----------------------------------------------------+

select all_match((x,y) -> x < y, [1,2,null], [4,5,6]);
+-------------------------------------------------------+
| all_match((x, y) -> x < y, [1, 2, NULL], [4, 5, 6]) |
+-------------------------------------------------------+
|                                                  NULL |
+-------------------------------------------------------+

select all_match((x,y) -> x < y, [1,2,8], [4,5,6]);
+-----------------------------------------------------+
| all_match((x, y) -> x < y, [1, 2, 8], [4, 5, 6]) |
+-----------------------------------------------------+
|                                                   0 |
+-----------------------------------------------------+

select all_match((x,y) -> x < y, [], []);
+-------------------------------------+
| all_match((x, y) -> x < y, [], []) |
+-------------------------------------+
|                                   1 |
+-------------------------------------+

select all_match((x,y) -> x < y, null, [4,5,6]);
+-----------------------------------------------+
| all_match((x, y) -> x < y, NULL, [4, 5, 6]) |
+-----------------------------------------------+
|                                          NULL |
+-----------------------------------------------+
```

Any_match

Возвращает, соответствует ли хотя бы один элемент массива заданному предикату.

- Возвращает `true` (1), если один или несколько элементов соответствуют предикату.
- Возвращает `false` (0), если ни один из элементов не соответствует (особый случай — когда массив пуст).
- Возвращает NULL, если предикат возвращает NULL для одного или нескольких элементов и `false` для всех остальных элементов.

Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
any_match(lambda_function, arr1, arr2...)
```

Возвращает, соответствует ли хотя бы один элемент `arr1` заданному предикату в lambda-функции.

Параметры

- `arr1`: массив для проверки соответствия.
- `arrN`: дополнительные массивы, используемые в lambda-функции.
- `lambda_function`: lambda-функция, используемая для проверки соответствия значений.

Возвращаемое значение

Возвращает значение типа BOOLEAN.

Примечания по использованию

- Lambda-функция следует примечаниям по использованию в array_map().
- Если входной массив равен null или lambda-функция возвращает null, возвращается null.
- Если `arr1` пуст, возвращается `false`.

- Чтобы применить эту функцию к MAP, перепишите `any_match((k,v)->k>v,map)` в `any_match(map_values(transform_values((k,v)->k>v, map)))`. Например, `select any_match(map_values(transform_values((k,v)->k>v, map{2:1})));` возвращает 1.

Примеры

Проверка, является ли хотя бы один элемент в `x` меньше элементов в `y`.

```
select any_match((x,y) -> x < y, [1,2,8], [4,5,6]);
+-------------------------------------------------+
| any_match((x, y) -> x < y, [1, 2, 8], [4, 5, 6]) |
+-------------------------------------------------+
|                                               1 |
+-------------------------------------------------+

select any_match((x,y) -> x < y, [11,12,8], [4,5,6]);
+----------------------------------------------------+
| any_match((x, y) -> x < y, [11, 12, 8], [4, 5, 6]) |
+----------------------------------------------------+
|                                                  0 |
+----------------------------------------------------+

select any_match((x,y) -> x < y, [11,12,null], [4,5,6]);
+-------------------------------------------------------+
| any_match((x, y) -> x < y, [11, 12, NULL], [4, 5, 6]) |
+-------------------------------------------------------+
|                                                  NULL |
+-------------------------------------------------------+

select any_match((x,y) -> x < y, [], []);
+-----------------------------------+
| any_match((x, y) -> x < y, [], []) |
+-----------------------------------+
|                                 0 |
+-----------------------------------+

select any_match((x,y) -> x < y, null, [4,5,6]);
+----------------------------------------------+
| any_match((x, y) -> x < y, NULL, [4, 5, 6]) |
+----------------------------------------------+
|                                         NULL |
+----------------------------------------------+
```

Array_agg

Агрегирует значения (включая `NULL`) в столбце в массив (несколько строк в одну строку) и опционально упорядочивает элементы по определенным столбцам. Начиная с версии v1.5.2, array_agg() поддерживает использование ORDER BY для сортировки элементов.

Синтаксис

```
ARRAY_AGG([distinct] col [order by col0 [desc | asc] [nulls first | nulls last] ...])
```

Параметры

- `col`: столбец, значения которого вы хотите агрегировать. Поддерживаемые типы данных: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, VARCHAR, CHAR, DATETIME, DATE, ARRAY (начиная с v1.5.2), MAP (начиная с v1.5.2) и STRUCT (начиная с v1.5.2).
- `col0`: столбец, который определяет порядок `col`. Может быть более одного столбца ORDER BY.
- `[desc | asc]`: указывает, следует ли сортировать элементы в порядке возрастания (по умолчанию) или убывания `col0`.
- `[nulls first | nulls last]`: указывает, размещаются ли значения null на первом или последнем месте.

Возвращаемое значение

Возвращает значение типа ARRAY, опционально отсортированное по `col0`.

Примечания по использованию

- Порядок элементов в массиве является случайным, что означает, что он может отличаться от порядка значений в столбце, если не указаны столбцы ORDER BY или если столбцы не отсортированы по порядку.
- Тип данных элементов в возвращаемом массиве совпадает с типом данных значений в столбце.
- Возвращает `NULL`, если входные данные пусты и нет столбцов group-by.
- ARRAY_AGG_DISTINCT() является псевдонимом ARRAY_AGG(DISTINCT).

Примеры

Рассмотрим следующую таблицу данных в качестве примера:

```
mysql> select * from t;
+------+------+------+
| a    | name | pv   |
+------+------+------+
|   11 |      |   33 |
|    2 | NULL |  334 |
|    1 | fzh  |    3 |
|    1 | fff  |    4 |
|    1 | fff  |    5 |
+------+------+------+
```

Пример 1: Группировка значений в столбце `a` и агрегация значений в столбце `pv` в массив с упорядочиванием `a` по `name`.

```
mysql> select a, array_agg(pv order by name nulls first) from t group by a;
+------+---------------------------------+
| a    | array_agg(pv ORDER BY name ASC) |
+------+---------------------------------+
|    2 | [334]                           |
|   11 | [33]                            |
|    1 | [4,5,3]                         |
+------+---------------------------------+

-- Агрегация значений без упорядочивания.
mysql> select a, array_agg(pv) from t group by a;
+------+---------------+
| a    | array_agg(pv) |
+------+---------------+
|   11 | [33]          |
|    2 | [334]         |
|    1 | [3,4,5]       |
+------+---------------+
3 rows in set (0.03 sec)
```

Пример 2: Агрегация значений в столбце `pv` в массив с упорядочиванием по `name`.

```
mysql> select array_agg(pv order by name desc nulls last) from t;
+-----------------------------------+
| array_agg(pv ORDER BY name DESC) |
+-----------------------------------+
| [3,4,5,33,334]                    |
+-----------------------------------+
1 row in set (0.02 sec)

-- Агрегация значений без упорядочивания.
mysql> select array_agg(pv) from t;
+----------------+
| array_agg(pv)  |
+----------------+
| [3,4,5,33,334] |
+----------------+
1 row in set (0.03 sec)
```

Пример 3: Агрегация значений в столбце `pv` с использованием предложения WHERE. Если ни одно значение в `pv` не соответствует условию фильтрации, возвращается значение `NULL`.

```
mysql> select array_agg(pv order by name desc nulls last) from t where a <
0;
+-----------------------------------+
| array_agg(pv ORDER BY name DESC) |
+-----------------------------------+
| NULL                              |
+-----------------------------------+
1 row in set (0.02 sec)

-- Агрегация значений без упорядочивания.
mysql> select array_agg(pv) from t where a < 0;
+---------------+
| array_agg(pv) |
+---------------+
| NULL          |
+---------------+
1 row in set (0.03 sec)
```

Ключевые слова

ARRAY_AGG, ARRAY

Array_append

Добавляет новый элемент в конец массива. Возвращает массив.

Синтаксис

```
array_append(any_array, any_element)
```

Примеры

```
mysql> select array_append([1, 2], 3);
+------------------------+
| array_append([1,2], 3) |
+------------------------+
| [1,2,3]                |
+------------------------+
1 row in set (0.00 sec)
```

Вы можете добавить NULL в массив.

```
mysql> select array_append([1, 2], NULL);
+---------------------------+
| array_append([1,2], NULL) |
+---------------------------+
| [1,2,NULL]                |
+---------------------------+
1 row in set (0.01 sec)
```

Ключевые слова

ARRAY_APPEND,ARRAY

Array_avg

Вычисляет среднее значение всех данных в ARRAY и возвращает этот результат.

Синтаксис

```
array_avg(array(type))
```

`array(type)` поддерживает следующие типы элементов: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, DECIMALV2.

Примеры

```
mysql> select array_avg([11, 11, 12]);
+-----------------------+
| array_avg([11,11,12]) |
+-----------------------+
| 11.333333333333334    |
+-----------------------+

mysql> select array_avg([11.33, 11.11, 12.324]);
+---------------------------------+
| array_avg([11.33,11.11,12.324]) |
+---------------------------------+
| 11.588                          |
+---------------------------------+
```

Keyword

ARRAY_AVG,ARRAY

Array_concat

Объединяет несколько массивов в один массив, который содержит все элементы из этих массивов.

Элементы в объединяемых массивах могут быть одинакового или разных типов. Однако мы рекомендуем, чтобы элементы были одного типа.

Nulls обрабатываются как обычные значения.

Синтаксис

```
array_concat(input0, input1, ...)
```

Параметры

`input`: один или несколько массивов, которые вы хотите объединить. Укажите массивы в формате `(input0, input1, ...)`. Эта функция поддерживает следующие типы элементов массива: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, VARCHAR, DECIMALV2, DATETIME, DATE и JSON. **JSON поддерживается начиная с версии 2.5.**

Возвращаемое значение

Возвращает массив, который содержит все элементы из массивов, указанных в параметре `input`. Элементы возвращаемого массива имеют тот же тип данных, что и элементы входных массивов. Кроме того, элементы возвращаемого массива следуют порядку входных массивов и их элементов.

Примеры

Пример 1: Объединение массивов, содержащих числовые элементы.

```
select array_concat([57.73,97.32,128.55,null,324.2], [3], [5]) as res;

+-----------------------------------+

| res                               |

+-----------------------------------+

| [57.73,97.32,128.55,null,324.2,3,5] |

+-----------------------------------+
```

Пример 2: Объединение массивов, содержащих строковые элементы.

```
select array_concat(["sql","storage","execute"], ["Query"], ["Vectorized",
"cbo"]);

+-----------------------------------------------------------------------------
--+

| array_concat(['sql','storage','execute'], ['Query'],
['Vectorized','cbo']) |

+-----------------------------------------------------------------------------
--+

| ["sql","storage","execute","Query","Vectorized","cbo"]
|

+-----------------------------------------------------------------------------
--+
```

Пример 3: Объединение двух массивов разных типов.

```
select array_concat([57,65], ["pear","apple"]);
+-------------------------------------------+
| array_concat([57, 65], ['pear', 'apple']) |
+-------------------------------------------+
| ["57","65","pear","apple"]                |
+-------------------------------------------+
```

Пример 4: Обработка nulls как обычных значений.

```
select array_concat(["sql",null], [null], ["Vectorized", null]);

+-----------------------------------------------------------+

| array_concat(['sql',NULL], [NULL], ['Vectorized',NULL]) |

+-----------------------------------------------------------+

| ["sql",null,null,"Vectorized",null]                      |

+-----------------------------------------------------------+
```

Array_contains

Проверяет, содержит ли массив определенный элемент. Если да, возвращается 1; в противном случае возвращается 0.

Синтаксис

```
array_contains(any_array, any_element)
```

Примеры

```
mysql> select array_contains(["apple","orange","pear"], "orange");
+-----------------------------------------------------+
| array_contains(['apple','orange','pear'], 'orange') |
+-----------------------------------------------------+
|                                                   1 |
+-----------------------------------------------------+
1 row in set (0.01 sec)
```

Вы также можете проверить, содержит ли массив NULL.

```
mysql> select array_contains([1, NULL], NULL);
+--------------------------------+
| array_contains([1,NULL], NULL) |
+--------------------------------+
|                              1 |
+--------------------------------+
1 row in set (0.00 sec)
```

Вы можете проверить, содержит ли многомерный массив определенный подмассив. В этом случае вам необходимо убедиться, что элементы подмассива полностью совпадают, включая порядок расположения элементов.

```
mysql> select array_contains([[1,2,3], [4,5,6]], [4,5,6]);
+---------------------------------------------+
| array_contains([[1,2,3],[4,5,6]], [4,5,6]) |
+---------------------------------------------+
|                                           1 |
+---------------------------------------------+
1 row in set (0.00 sec)

mysql> select array_contains([[1,2,3], [4,5,6]], [4,6,5]);
+---------------------------------------------+
| array_contains([[1,2,3],[4,5,6]], [4,6,5]) |
+---------------------------------------------+
|                                           0 |
+---------------------------------------------+
1 row in set (0.00 sec)
```

Keyword

ARRAY_CONTAINS,ARRAY

Array_contains_all

Проверяет, содержит ли `arr1` все элементы `arr2`, то есть является ли `arr2` подмножеством `arr1`. Если да, возвращается 1. Если нет, возвращается 0.

Синтаксис

```
BOOLEAN array_contains_all(arr1, arr2)
```

Параметры

`arr`: два массива для сравнения. Этот синтаксис проверяет, является ли `arr2` подмножеством `arr1`.

Типы данных элементов в обоих массивах должны быть одинаковыми. Информацию о типах данных элементов массива, поддерживаемых Selena, см. в разделе ARRAY.

Возвращаемое значение

Возвращает значение типа BOOLEAN.

Возвращается 1, если `arr2` является подмножеством `arr1`. В противном случае возвращается 0.

Если какой-либо из двух массивов равен NULL, возвращается NULL.

Примечания по использованию

- Если массив содержит элементы `null`, `null` обрабатывается как значение.
- Пустой массив является подмножеством любого массива.
- Элементы в двух массивах могут иметь разный порядок.

Примеры

1. Создайте таблицу с именем `t1` и вставьте в нее данные.

```
CREATE TABLE t1 (
    c0 INT,
    c1 ARRAY<INT>,
    c2 ARRAY<INT>
) ENGINE=OLAP
DUPLICATE KEY(c0)
DISTRIBUTED BY HASH(c0);

INSERT INTO t1 VALUES
    (1,[1,2,3],[1,2]),
    (2,[1,2,3],[1,4]),
    (3,NULL,[1]),
    (4,[1,2,null],NULL),
    (5,[1,2,null],[null]),
    (6,[2,3],[]);
```

2. Запросите данные из этой таблицы.

```
SELECT * FROM t1 ORDER BY c0;
+------+------------+----------+
| c0   | c1         | c2       |
+------+------------+----------+
|    1 | [1,2,3]    | [1,2]    |
|    2 | [1,2,3]    | [1,4]    |
|    3 | NULL       | [1]      |
|    4 | [1,2,null] | NULL     |
|    5 | [1,2,null] | [null]   |
|    6 | [2,3]      | []       |
+------+------------+----------+
```

3. Проверьте, является ли каждая строка `c2` подмножеством соответствующей строки `c1`.

```
SELECT c0, c1, c2, array_contains_all(c1, c2) FROM t1 ORDER BY c0;
+------+------------+----------+----------------------------+
| c0   | c1         | c2       | array_contains_all(c1, c2) |
+------+------------+----------+----------------------------+
|    1 | [1,2,3]    | [1,2]    |                          1 |
|    2 | [1,2,3]    | [1,4]    |                          0 |
|    3 | NULL       | [1]      |                       NULL |
|    4 | [1,2,null] | NULL     |                       NULL |
|    5 | [1,2,null] | [null]   |                          1 |
|    6 | [2,3]      | []       |                          1 |
+------+------------+----------+----------------------------+
```

В выводе:

Для строки 1, `c2` является подмножеством `c1` и возвращается 1.

Для строки 2, `c2` не является подмножеством `c1` и возвращается 0.

Для строки 3, `c1` равен NULL и возвращается NULL.

Для строки 4, `c2` равен NULL и возвращается NULL.

Для строки 5, оба массива содержат `null`, и `null` обрабатывается как обычное значение, возвращается `1`.

Для строки 6, `c2` является пустым массивом и считается подмножеством `c1`. Поэтому возвращается `1`.

Array_contains_seq

Проверяет, содержатся ли все элементы array2 в array1 в том же порядке. Функция возвращает 1 тогда и только тогда, когда `array1 = prefix + array2 + suffix`.

Например:

- `select array_contains_seq([1,2,3,4], [1,2,3]);` возвращает 1.
- `select array_contains_seq([1,2,3,4], [4,3]);` возвращает 0.

Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
BOOLEAN array_contains_seq(arr1, arr2)
```

Параметры

`arr`: два массива для сравнения. Этот синтаксис проверяет, является ли `arr2` подмножеством `arr1` и находятся ли элементы в том же порядке.

Типы данных элементов в двух массивах должны совпадать. Список типов данных элементов массива, поддерживаемых Selena, см. в разделе ARRAY.

Возвращаемое значение

Возвращает значение типа BOOLEAN.

- Возвращается `1`, если `arr2` является подмножеством `arr1` и элементы в `arr2` соблюдают тот же порядок, что и в `arr1`. В противном случае возвращается `0`.
- Пустой массив является подмножеством любого массива. Поэтому возвращается `1`, если `arr2` пуст, но `arr1` является допустимым массивом.
- Возвращается NULL, если любой из входных массивов является NULL.
- Значения NULL в массивах обрабатываются как обычные значения. Например, `SELECT array_contains_seq([1, 2, NULL, 3, 4], [2,3])` вернет 0. Однако `SELECT array_contains_seq([1, 2, NULL, 3, 4], [2,NULL,3])` вернет 1.

Примеры

```
MySQL > select array_contains_seq([1,2,3,4], [1,2,3]);
+----------------------------------------------+
| array_contains_seq([1, 2, 3, 4], [1, 2, 3]) |
+----------------------------------------------+
|                                            1 |
+----------------------------------------------+

MySQL > select array_contains_seq([1,2,3,4], [3,2]);
+-------------------------------------------+
| array_contains_seq([1, 2, 3, 4], [3, 2]) |
+-------------------------------------------+
|                                         0 |
+-------------------------------------------+

MySQL > select array_contains_seq([1, 2, NULL, 3, 4], ['a']);
+-------------------------------------------------+
| array_contains_all([1, 2, NULL, 3, 4], ['a']) |
+-------------------------------------------------+
|                                               0 |
+-------------------------------------------------+

MySQL > select array_contains_seq([1,2,3,4,null], null);
+-------------------------------------------+
| array_contains([1, 2, 3, 4, NULL], NULL) |
+-------------------------------------------+
|                                      NULL |
+-------------------------------------------+

MySQL > select array_contains_seq([1,2,3,4], []);
+--------------------------------------+
| array_contains_seq([1, 2, 3, 4], []) |
+--------------------------------------+
|                                    1 |
+--------------------------------------+
```

Array_cum_sum

Возвращает кумулятивные суммы элементов в массиве.

Синтаксис

```
array_cum_sum(array(bigint))
array_cum_sum(array(double))
```

Параметры

`array`: Элементы в массиве должны быть типа BIGINT (8-байтовое знаковое целое число) или DOUBLE (8-байтовое число с плавающей точкой).

Возвращаемые значения

Возвращает массив.

Возвращается NULL, если входной массив является NULL.

Примеры

Пример 1: Возврат кумулятивных сумм массива.

```
select array_cum_sum([11, 11, 12]);
+---------------------------+
| array_cum_sum([11,11,12]) |
+---------------------------+
| [11,22,34]                |
+---------------------------+
```

Пример 2: Использование этой функции с CAST.

```
select array_cum_sum([cast(11.33 as double),cast(11.11 as
double),cast(12.324 as double)]);
+-----------------------------------------------------------------------------
-------------+
| array_cum_sum([CAST(11.33 AS DOUBLE), CAST(11.11 AS DOUBLE), CAST(12.324
AS DOUBLE)]) |
+-----------------------------------------------------------------------------
-------------+
| [11.33,22.439999999999998,34.763999999999996]
|
+-----------------------------------------------------------------------------
-------------+
```

Пример 3: Входной массив содержит null.

```
select array_cum_sum([null,1,2]);
+---------------------------------+
| array_cum_sum([null,1,2])       |
+---------------------------------+
| [null,1,3]                      |
+---------------------------------+
```

Пример 4: Входной массив является null.

```
select array_cum_sum(null);
+---------------------------------+
| array_cum_sum(NULL)             |
+---------------------------------+
| NULL                            |
+---------------------------------+
```

Array_difference

Вычисляет разность между каждыми двумя соседними элементами массива путем вычитания каждого элемента из следующего за ним элемента и возвращает массив, состоящий из этих разностей.

Синтаксис

```
array_difference(input)
```

Параметры

`input`: массив, для которого вы хотите вычислить разности между каждыми двумя соседними элементами.

Возвращаемое значение

Возвращает массив того же типа данных и длины, что и массив, указанный в параметре `input`.

Примеры

Пример 1:

```
mysql> SELECT array_difference([342, 32423, 213, 23432]);
+-----------------------------------------+
| array_difference([342,32423,213,23432]) |
+-----------------------------------------+
| [0,32081,-32210,23219]                  |
+-----------------------------------------+
```

Пример 2:

```
mysql> SELECT array_difference([342, 32423, 213, null, 23432]);
+-----------------------------------------------+
| array_difference([342,32423,213,NULL,23432]) |
+-----------------------------------------------+
| [0,32081,-32210,null,null]                    |
+-----------------------------------------------+
```

Пример 3:

```
mysql> SELECT array_difference([1.2, 2.3, 3.2, 4324242.55]);
+---------------------------------------------+
| array_difference([1.2,2.3,3.2,4324242.55]) |
+---------------------------------------------+
| [0,1.1,0.9,4324239.35]                      |
+---------------------------------------------+
```

Пример 4:

```
mysql> SELECT array_difference([false, true, false]);
+----------------------------------------+
| array_difference([FALSE, TRUE, FALSE]) |
+----------------------------------------+
| [0,1,-1]                               |
+----------------------------------------+
```

Array_distinct

Удаляет дублирующиеся элементы из массива.

Синтаксис

```
array_distinct(array)
```

Параметры

`array`: массив, из которого вы хотите удалить дублирующиеся элементы. Поддерживается только тип данных ARRAY.

Возвращаемое значение

Возвращает массив.

Примечания по использованию

- Элементы возвращаемого массива могут быть отсортированы в порядке, отличном от порядка элементов указанного вами массива.
- Элементы возвращаемого массива имеют тот же тип данных, что и элементы указанного вами массива.

Примеры

В этом разделе в качестве примера используется следующая таблица:

```
mysql> select * from test;

+------+---------------+

| c1   | c2            |

+------+---------------+

|    1 | [1,1,2]       |

|    2 | [1,null,null] |

|    3 | NULL          |

|    4 | [null]        |

+------+---------------+
```

Удаление дублирующихся значений из столбца `c2`.

```
mysql> select c1, array_distinct(c2) from test;

+------+----------------------+

| c1   | array_distinct(`c2`) |

+------+----------------------+

|    1 | [2,1]                |

|    2 | [null,1]             |

|    3 | NULL                 |

|    4 | [null]               |

+------+----------------------+
```

Array_filter

Возвращает значения из массива, соответствующие заданному фильтру.

Эта функция имеет две следующие формы. Вариант с lambda-выражением обеспечивает более гибкую фильтрацию массивов. Для получения дополнительной информации о lambda-функциях см. Lambda expression. Эта функция поддерживается начиная с версии 2.5.

Синтаксис

```
array_filter(array, array<bool>)
array_filter(lambda_function, arr1,arr2...)
```

- `array_filter(array, array<bool>)`

  Возвращает значения из массива, соответствующие `array<bool>`.

- `array_filter(lambda_function, arr1,arr2...)`

  Возвращает значения из массива, соответствующие lambda-функции.

Параметры

`array`: массив, из которого нужно отфильтровать значения.

`array<bool>`: выражение, используемое для фильтрации значений.

`lambda_function`: lambda-функция, используемая для фильтрации значений.

Примечания по использованию

- Два входных параметра `array_filter(array, array<bool>)` должны быть типа ARRAY, а выражение фильтра должно возвращать `array<bool>`.
- Lambda-функция в `array_filter(lambda_function, arr1,arr2...)` следует примечаниям по использованию в array_map().
- Если входной массив равен null, возвращается null. Если массив фильтра равен null, возвращается пустой массив.

Примеры

- Примеры без использования lambda-функций

```
-- Все элементы массива соответствуют фильтру.
select array_filter([1,2,3],[1,1,1]);
+-----------------------------------+
| array_filter([1, 2, 3], [1, 1, 1]) |
+-----------------------------------+
| [1,2,3]                           |
+-----------------------------------+
1 row in set (0.01 sec)

-- Фильтр равен null, возвращается пустой массив.
select array_filter([1,2,3],null);
+-------------------------------+
| array_filter([1, 2, 3], NULL) |
+-------------------------------+
| []                            |
+-------------------------------+
1 row in set (0.01 sec)

-- Входной массив равен null, возвращается null.
select array_filter(null,[1]);
+-------------------------+
| array_filter(NULL, [1]) |
+-------------------------+
| NULL                    |
+-------------------------+

-- И входной массив, и фильтр равны null. Возвращается null.
select array_filter(null,null);
+--------------------------+
| array_filter(NULL, NULL) |
+--------------------------+
| NULL                     |
+--------------------------+
1 row in set (0.01 sec)

-- Фильтр содержит элемент null, возвращается пустой массив.
select array_filter([1,2,3],[null]);
+---------------------------------+
| array_filter([1, 2, 3], [NULL]) |
+---------------------------------+
| []                              |
+---------------------------------+
1 row in set (0.01 sec)

-- Фильтр содержит два элемента null, возвращается пустой массив.
```

```
select array_filter([1,2,3],[null,null]);
+---------------------------------------+
| array_filter([1, 2, 3], [NULL, NULL]) |
+---------------------------------------+
| []                                    |
+---------------------------------------+
1 row in set (0.00 sec)

-- Только один элемент соответствует фильтру.
select array_filter([1,2,3],[null,1,0]);
+---------------------------------------+
| array_filter([1, 2, 3], [NULL, 1, 0]) |
+---------------------------------------+
| [2]                                   |
+---------------------------------------+
1 row in set (0.00 sec)
```

- Примеры с использованием lambda-функций

```
-- Возвращает элементы x, которые меньше элементов y.
select array_filter((x,y) -> x < y, [1,2,null], [4,5,6]);
+-------------------------------------------------------+
| array_filter((x, y) -> x < y, [1, 2, NULL], [4, 5, 6]) |
+-------------------------------------------------------+
| [1,2]                                                 |
+-------------------------------------------------------+
```

```
-- Возвращает элементы x, которые больше или равны 3.
SELECT array_filter(x->x>=3, [2,3,4]);
+---------------------------------------+
| array_filter(x -> x >= 3, [2, 3, 4]) |
+---------------------------------------+
| [3,4]                                 |
+---------------------------------------+
```

Array_flatten

`array_flatten` выравнивает один уровень вложенных массивов.

Синтаксис

```
array_flatten(param)
```

Параметры

`param`: вложенный массив, который необходимо выровнять. Поддерживаются только вложенные массивы, причем массив может быть многоуровневым, элементы массива могут иметь любой тип данных, поддерживаемый Selena.

Возвращаемое значение

Тип данных возвращаемого значения — массив после выравнивания одного уровня.

Примеры

Пример 1: Выравнивание двухуровневого вложенного массива.

```
mysql> SELECT array_flatten([[1, 2], [1, 4]]) as res;
+-----------+
| res       |
+-----------+
| [1,2,1,4] |
+-----------+
```

Пример 2: Выравнивание трёхуровневого вложенного массива.

```
mysql> SELECT array_flatten([[[1],[2]], [[3],[4]]]) as res;
+-------------------+
| res               |
+-------------------+
| [[1],[2],[3],[4]] |
+-------------------+
```

Array_generate

Возвращает массив уникальных значений в диапазоне, заданном параметрами `start` и `end`, с шагом `step`.

Эта функция поддерживается начиная с версии 3.1.

Синтаксис

```
ARRAY array_generate([start,] end [, step])
```

Параметры

- `start`: необязательный, начальное значение. Должно быть константой или столбцом типа TINYINT, SMALLINT, INT, BIGINT или LARGEINT. Значение по умолчанию — 1.
- `end`: обязательный, конечное значение. Должно быть константой или столбцом типа TINYINT, SMALLINT, INT, BIGINT или LARGEINT.
- `step`: необязательный, шаг приращения. Должен быть константой или столбцом типа TINYINT, SMALLINT, INT, BIGINT или LARGEINT. Когда `start` меньше `end`, значение по умолчанию — 1. Когда `start` больше `end`, значение по умолчанию — -1.

Возвращаемое значение

Возвращает массив, элементы которого имеют тот же тип данных, что и входные параметры.

Примечания по использованию

- Если какой-либо входной параметр является столбцом, необходимо указать таблицу, которой принадлежит столбец.
- Если какой-либо входной параметр является столбцом, необходимо указать остальные параметры. Значения по умолчанию не поддерживаются.
- Если какой-либо входной параметр равен NULL, возвращается NULL.
- Если `step` равен 0, возвращается пустой массив.
- Если `start` равен `end`, возвращается это значение.

© ООО «Платформа Данных Селена», 2026

Примеры

Входные параметры являются константами

```
mysql> select array_generate(9);
+---------------------+
| array_generate(9)   |
+---------------------+
| [1,2,3,4,5,6,7,8,9] |
+---------------------+

select array_generate(9,12);
+-----------------------+
| array_generate(9, 12) |
+-----------------------+
| [9,10,11,12]          |
+-----------------------+

select array_generate(9,6);
+----------------------+
| array_generate(9, 6) |
+----------------------+
| [9,8,7,6]            |
+----------------------+

select array_generate(9,6,-1);
+--------------------------+
| array_generate(9, 6, -1) |
+--------------------------+
| [9,8,7,6]                |
+--------------------------+

select array_generate(3,3);
+----------------------+
| array_generate(3, 3) |
+----------------------+
| [3]                  |
+----------------------+
```

Один из входных параметров является столбцом

```
CREATE TABLE `array_generate`
(
  `c1` TINYINT,
  `c2` SMALLINT,
  `c3` INT
)
ENGINE = OLAP
DUPLICATE KEY(`c1`)
DISTRIBUTED BY HASH(`c1`);

INSERT INTO `array_generate` VALUES
(1, 6, 3),
(2, 9, 4);
```

```
mysql> select array_generate(1,c2,2) from `array_generate`;
+--------------------------+
| array_generate(1, c2, 2) |
+--------------------------+
| [1,3,5]                  |
| [1,3,5,7,9]              |
+--------------------------+
```

Array_intersect

Возвращает массив элементов, находящихся на пересечении одного или нескольких массивов.

Синтаксис

```
array_intersect(input0, input1, ...)
```

Параметры

`input`: один или несколько массивов, пересечение которых нужно получить. Укажите массивы в формате `(input0, input1, ...)` и убедитесь, что указанные массивы имеют одинаковый тип данных.

Возвращаемое значение

Возвращает массив того же типа данных, что и указанные массивы.

Примеры

Пример 1:

```
mysql> SELECT array_intersect(["SQL", "storage"], ["mysql", "query",
"SQL"], ["SQL"])
AS no_intersect ;
+--------------+
| no_intersect |
+--------------+
| ["SQL"]      |
+--------------+
```

Пример 2:

```
mysql> SELECT array_intersect(["SQL", "storage"], ["mysql", null], [null])
AS no_intersect ;
+--------------+
| no_intersect |
+--------------+
| []           |
+--------------+
```

Пример 3:

```
mysql> SELECT array_intersect(["SQL", null, "storage"], ["mysql", null],
[null]) AS no_intersect ;
+--------------+
| no_intersect |
+--------------+
| [null]       |
+--------------+
```

Array_join

Объединяет элементы массива в строку.

Синтаксис

```
array_join(array, sep[, null_replace_str])
```

Параметры

- `array`: массив, элементы которого вы хотите объединить. Поддерживается только тип данных ARRAY.
- `sep`: разделитель, который используется для разделения объединенных элементов массива. Поддерживается только тип данных VARCHAR.
- `null_replace_str`: строка, которая используется для замены значений `NULL`. Поддерживается только тип данных VARCHAR.

Возвращаемое значение

Возвращает значение типа VARCHAR.

Примечания по использованию

- Значение параметра `array` должно быть одномерным массивом.
- Параметр `array` не поддерживает значения DECIMAL.
- Если вы установите параметр `sep` в `NULL`, возвращаемое значение будет `NULL`.
- Если вы не укажете параметр `null_replace_str`, значения `NULL` будут отброшены.
- Если вы установите параметр `null_replace_str` в `NULL`, возвращаемое значение будет `NULL`.

Примеры

Пример 1: Объединение элементов массива. В этом примере значение `NULL` в массиве отбрасывается, а объединенные элементы массива разделяются подчеркиваниями (`_`).

```
mysql> select array_join([1, 3, 5, null], '_');

+-------------------------------+

| array_join([1,3,5,NULL], '_') |

+-------------------------------+

| 1_3_5                         |

+-------------------------------+
```

Пример 2: Объединение элементов массива. В этом примере значение `NULL` в массиве заменяется строкой `NULL`, а объединенные элементы массива разделяются подчеркиваниями (`_`).

```
mysql> select array_join([1, 3, 5, null], '_', 'NULL');

+---------------------------------------+

| array_join([1,3,5,NULL], '_', 'NULL') |

+---------------------------------------+

| 1_3_5_NULL                            |

+---------------------------------------+
```

Array_length

Возвращает количество элементов в массиве. Тип результата — INT. Если входной параметр NULL, результат также NULL. Null-элементы учитываются в длине.

Имеет псевдоним cardinality().

Синтаксис

```
INT array_length(any_array)
```

Параметры

`any_array`: значение ARRAY, из которого вы хотите получить количество элементов.

Возвращаемое значение

Возвращает значение типа INT.

Примеры

```
mysql> select array_length([1,2,3]);
+-----------------------+
| array_length([1,2,3]) |
+-----------------------+
|                     3 |
+-----------------------+
1 row in set (0.00 sec)

mysql> select array_length([1,2,3,null]);
+-------------------------------+
| array_length([1, 2, 3, NULL]) |
+-------------------------------+
|                             4 |
+-------------------------------+

mysql> select array_length([[1,2], [3,4]]);
+----------------------------+
| array_length([[1,2],[3,4]]) |
+----------------------------+
|                          2 |
+----------------------------+
1 row in set (0.01 sec)
```

Keywords

ARRAY_LENGTH, ARRAY, CARDINALITY

Array_map

array_map() — это функция SQL высшего порядка, которая может принимать лямбда-функции в качестве аргументов. Она применяет лямбда-функцию к входным массивам и возвращает новый массив.

array_map() может принимать несколько массивов и может быть вложена с лямбда-функциями для захвата переменной. Для получения дополнительной информации о лямбда-функциях см. Lambda expression. Эта функция поддерживается начиная с версии v1.5.2.

Псевдоним — transform().

Синтаксис

```
array_map(lambda_function, arr1, arr2...)
array_map(arr1, arr2..., lambda_function)
```

Примечания по использованию

- Лямбда-функции могут использоваться только как **первый** или **последний** параметр функции высшего порядка. Любая позиция не влияет на вычисление.
- Количество массивов должно соответствовать количеству входных параметров в лямбда-функции. В противном случае возвращается ошибка.
- Все массивы должны иметь одинаковое количество элементов. В противном случае возвращается ошибка.

Примеры

Пример 1: Увеличение каждого элемента массива на 100.

```
select array_map(x -> x+100,[1,2,3]);
+-------------------------------------+
| array_map(x -> x + 100, [1, 2, 3]) |
+-------------------------------------+
| [101,102,103]                       |
+-------------------------------------+
```

Пример 2: Поэлементное суммирование массива1 [1,2,3] и массива2 [11,12,13].

```
select array_map((x,y) -> x + y, [1,2,3], [11,12,13]);
+-----------------------------------------------------+
| array_map((x, y) -> x + y, [1, 2, 3], [11, 12, 13]) |
+-----------------------------------------------------+
| [12,14,16]                                          |
+-----------------------------------------------------+
```

Пример 3: Определение, больше ли элемент в `x` значения 1.5. Если да, для элемента возвращается log(x). Если нет, для соответствующего элемента возвращается (x+y).

```
select array_map((x,y) -> if(x>1.5,log(x),x+y), [1,2,3], [11,12,13]);
+-------------------------------------------------------------------------
+
| array_map((x, y) -> if(x > 1.5, log(x), x + y), [1, 2, 3], [11, 12, 13])
|
+-------------------------------------------------------------------------
+
| [12,0.6931471805599453,1.0986122886681098]
|
+-------------------------------------------------------------------------
+
```

Пример 4: Использование array_map() для захвата переменных. Возвращаемый элемент со значением 1 указывает, что условие выполнено.

```
-- В примере таблицы last_avg — это средний балл на последнем экзамене.
score представляет баллы по трем предметам на текущем экзамене.

+------+----------+------------+
| id   | last_avg | score      |
+------+----------+------------+
|    1 |       55 | [50,60,70] |
|    2 |       73 | [70,65,75] |
|    3 |       89 | [88,92,90] |
+------+----------+------------+

-- Найти баллы, которые выше среднего балла последнего экзамена.
select array_map(x -> x > last_avg, score) from test_tbl;
+--------------------------------------+
| array_map(x -> x > last_avg, score)  |
+--------------------------------------+
| [0,1,1]                              |
| [0,0,1]                              |
| [0,1,1]                              |
+--------------------------------------+
```

Пример 5: Использование вложенных лямбда-функций.

```
select array_map(x -> array_map(x->x+100, x),[[1,2.3],[4,3,2]]);
+----------------------------------------------------------------------+
| array_map(x -> array_map(x -> x + 100, x), [[1, 2.3], [4, 3, 2]]) |
+----------------------------------------------------------------------+
| [[101,102.3],[104,103,102]]                                          |
+----------------------------------------------------------------------+
```

Пример 6: Возвращается ошибка, потому что количество массивов и параметров лямбда-функции не совпадает. Лямбда-функция требует только один массив, но передано два массива.

```
select array_map(x -> x,[1],[2,4]);
ERROR 1064 (HY000): Lambda arguments should equal to lambda input arrays.
```

Array_max

Получает максимальное значение среди всех данных в ARRAY и возвращает этот результат.

Синтаксис

```
array_max(array(type))
```

Примеры

```
mysql> select array_max([113, 11, 12]);
+------------------------+
| array_max([113,11,12]) |
+------------------------+
| 113                    |
+------------------------+

mysql> select array_max([11.33, 11.11, 12.324]);
+---------------------------------+
| array_max([11.33,11.11,12.324]) |
+---------------------------------+
| 12.324                          |
+---------------------------------+

mysql> select array_max([cast('2020-02-25 11:35:17' as datetime),
cast('2019-08-25 17:07:10' as datetime), cast('2025-08-25 17:07:10' as
datetime)]);
+---------------------------------------------------------------------------
-------------------------------------------------------------+
| array_max([CAST('2020-02-25 11:35:17' AS DATETIME),CAST('2019-08-25
17:07:10' AS DATETIME),CAST('2025-08-25 17:07:10' AS DATETIME)]) |
+---------------------------------------------------------------------------
-------------------------------------------------------------+
| 2025-08-25 17:07:10
|
+---------------------------------------------------------------------------
-------------------------------------------------------------+
```

Keyword

ARRAY_MAX,ARRAY

Array_min

Получает минимальное значение из всех данных в ARRAY и возвращает этот результат.

Синтаксис

```
array_min(array(type))
```

Примеры

```
mysql> select array_min([113, 11, 12]);
+------------------------+
| array_min([113,11,12]) |
+------------------------+
| 11                     |
+------------------------+

mysql> select array_min([11.33, 11.11, 12.324]);
+---------------------------------+
| array_min([11.33,11.11,12.324]) |
+---------------------------------+
| 11.11                           |
+---------------------------------+

mysql> select array_min([cast('2020-02-25 11:35:17' as datetime),
cast('2019-08-25 17:07:10' as datetime), cast('2025-08-25 17:07:10' as
datetime)]);
+---------------------------------------------------------------------------
-------------------------------------------------------------+
| array_min([CAST('2020-02-25 11:35:17' AS DATETIME),CAST('2019-08-25
17:07:10' AS DATETIME),CAST('2025-08-25 17:07:10' AS DATETIME)]) |
+---------------------------------------------------------------------------
-------------------------------------------------------------+
| 2019-08-25 17:07:10
|
+---------------------------------------------------------------------------
-------------------------------------------------------------+
```

Keyword

ARRAY_MIN,ARRAY

Array_position

Получает позицию элемента в массиве.

Синтаксис

```
array_position(any_array, any_element)
```

Параметры

- `any_array`: массив для поиска.
- `any_element`: выражение, соответствующее элементу в массиве.

Возвращаемое значение

Возвращает позицию указанного элемента, если элемент найден в указанном массиве, и возвращает 0, если элемент не найден в массиве.

Примеры

- Пример 1: Получить позицию элемента в массиве.

```
mysql> select array_position(["apple","orange","pear"], "orange");

+-----------------------------------------------------+

| array_position(['apple','orange','pear'], 'orange') |

+-----------------------------------------------------+

|                                                   2 |

+-----------------------------------------------------+

1 row in set (0.01 sec)
```

- Пример 2: Получить позицию `NULL` в массиве.

```
mysql> select array_position([1, NULL], NULL);
```

```
+--------------------------------+

| array_position([1,NULL], NULL) |

+--------------------------------+

|                              2 |

+--------------------------------+

1 row in set (0.00 sec)
```

- Пример 3: Получить позицию подмассива в многомерном массиве. Позиция может быть возвращена только в том случае, если подмассив, состоящий из тех же элементов в том же порядке, что и указанный подмассив, найден в многомерном массиве.

```
mysql> select array_position([[1,2,3], [4,5,6]], [4,5,6]);

+-------------------------------------------+

| array_position([[1,2,3],[4,5,6]], [4,5,6]) |

+-------------------------------------------+

|                                         2 |

+-------------------------------------------+

1 row in set (0.00 sec)



mysql> select array_position([[1,2,3], [4,5,6]], [4,6,5]);

+-------------------------------------------+

| array_position([[1,2,3],[4,5,6]], [4,6,5]) |

+-------------------------------------------+

|                                         0 |

+-------------------------------------------+

1 row in set (0.00 sec)
```

Keywords

ARRAY_POSITION, ARRAY

Array_remove

Удаляет элемент из массива.

Синтаксис

```
array_remove(any_array, any_element)
```

Параметры

- `any_array`: массив для поиска.
- `any_element`: выражение, соответствующее элементу в массиве.

Возвращаемое значение

Возвращает массив, из которого удален указанный элемент.

Примеры

```
mysql> select array_remove([1,2,3,null,3], 3);

+---------------------------------+

| array_remove([1,2,3,NULL,3], 3) |

+---------------------------------+

| [1,2,null]                      |

+---------------------------------+

1 row in set (0.01 sec)
```

Keywords

ARRAY_REMOVE, ARRAY

Array_repeat

`array_repeat` возвращает массив, содержащий данный элемент, повторенный указанное количество раз.

Синтаксис

```
array_repeat(element, count)
```

Параметры

- `element`: Элемент для повторения может быть любого типа данных, поддерживаемого Selena.

- `count`: Количество повторений, типа INT.

Возвращаемое значение

Тип данных возвращаемого значения - это тип ARRAY элемента.

Примечания по использованию

- Когда count меньше 1, возвращается пустой массив.
- Когда параметр element равен NULL, результатом является массив, состоящий из count значений NULL.
- Когда параметр count равен NULL, результат равен NULL.

Примеры

Пример 1:

```
select array_repeat(1,5) as res;
```

```
+-------------+
| res         |
+-------------+
| [1,1,1,1,1] |
+-------------+
```

Пример 2:

```
select  array_repeat([1,2],3) as res;
```

```
+---------------------+
| res                 |
+---------------------+
| [[1,2],[1,2],[1,2]] |
+---------------------+
```

Пример 3:

```
select array_repeat(1,-1) as res;
```

```
+------+
| res  |
+------+
| []   |
+------+
```

Пример 4:

```
select  array_repeat(null,3) as res;
```

```
+------+
| res  |
+------+
| NULL |
+------+
```

Пример 5:

```
CREATE TABLE IF NOT EXISTS test (COLA INT, COLB INT) PROPERTIES
("replication_num"="1");
INSERT INTO test (COLA, COLB) VALUES (1, 3), (NULL, 3), (2, NULL);
select array_repeat(COLA,COLB) from test;
```

```
+--------------------------+
| array_repeat(COLA, COLB) |
+--------------------------+
| [1,1,1]                  |
| [null,null,null]         |
| NULL                     |
+--------------------------+
```

Array_slice

Возвращает срез массива. Эта функция извлекает `length` элементов из `input` начиная с позиции, указанной в `offset`.

Синтаксис

```
array_slice(input, offset, length)
```

Параметры

- `input`: массив, из которого вы хотите извлечь срез. Функция поддерживает следующие типы элементов массива: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, VARCHAR, DECIMALV2, DATETIME, DATE и JSON. **JSON поддерживается начиная с версии 2.5.**

- `offset`: позиция, с которой начинается извлечение элементов. Допустимые значения начинаются с `1`. Должно быть значением типа BIGINT.

- `length`: длина среза, который вы хотите извлечь. Должно быть значением типа BIGINT.

Возвращаемое значение

Возвращает массив того же типа данных, что и массив, указанный в параметре `input`.

Примечания по использованию

- Смещение начинается с 1.
- Если указанная длина превышает фактическое количество элементов, которые могут быть извлечены, возвращаются все соответствующие элементы. См. Пример 4.

Примеры

Пример 1: Извлечь 2 элемента, начиная с третьего элемента.

```
mysql> select array_slice([1,2,4,5,6], 3, 2) as res;
+-------+
| res   |
+-------+
| [4,5] |
+-------+
```

Пример 2: Извлечь 2 элемента, начиная с первого элемента.

```
mysql> select array_slice(["sql","storage","query","execute"], 1, 2) as
res;
+--------------------+
| res                |
+--------------------+
| ["sql","storage"] |
+--------------------+
```

Пример 3: Null-элементы обрабатываются как обычные значения.

```
mysql> select array_slice([57.73,97.32,128.55,null,324.2], 3, 3) as res;
+----------------------+
| res                  |
+----------------------+
| [128.55,null,324.2] |
+----------------------+
```

Пример 4: Извлечь 5 элементов, начиная с третьего элемента.

Функция пытается извлечь 5 элементов, но с третьего элемента доступно только 3 элемента. В результате возвращаются все 3 элемента.

```
mysql> select array_slice([57.73,97.32,128.55,null,324.2], 3, 5) as res;
+----------------------+
| res                  |
+----------------------+
| [128.55,null,324.2] |
+----------------------+
```

Array_sort

Сортирует элементы массива в порядке возрастания.

Синтаксис

```
array_sort(array)
```

```
array_sort(array, (x,y)->expr(x,y))
```

Параметры

`array`: массив, элементы которого вы хотите отсортировать. Поддерживается только тип данных ARRAY.

`lambda_comparator`: необязательная lambda-функция компаратор в формате `(x,y)->expr(x,y)`. Lambda-компаратор:

- Должен зависеть только от аргументов x и y
- Должен возвращать булев или числовой тип
- Результат не должен быть NULL
- Должен удовлетворять требованиям строгого слабого порядка

Возвращаемое значение

Возвращает массив.

Примечания по использованию

Базовая сортировка

- Эта функция сортирует элементы массива в порядке возрастания при использовании без lambda-компаратора.
- Значения `NULL` помещаются в начало возвращаемого массива.
- Элементы возвращаемого массива имеют тот же тип данных, что и элементы входного массива.

Сортировка с Lambda-компаратором

- При использовании lambda-компаратора порядок сортировки определяется результатом выражения:
  - Для булева возврата: `true` означает, что x должен быть перед y
  - Для числового возврата: отрицательное значение означает, что x идет перед y
- Lambda-компаратор должен удовлетворять требованиям строгого слабого порядка:
  - **Иррефлексивность**: Для всех `x`, `expr(x, x)` должно возвращать false (булев) или неотрицательное значение (числовой)
  - **Асимметрия**: Если `expr(x, y)` выполняется, то `expr(y, x)` не должно выполняться
  - **Транзитивность**: Если `expr(x, y)` и `expr(y, z)` выполняются, то `expr(x, z)` должно выполняться
  - **Связность**: Для всех `x` и `y`, либо `expr(x, y)`, либо `expr(y, x)`, либо `x = y` должно быть истинным
- Выражение lambda-компаратора должно:
  - Зависеть только от параметров `x` и `y`
  - Возвращать булев или числовой тип
  - Никогда не возвращать NULL
  - Всегда выдавать одинаковые результаты для одинаковых входных значений

Порядок убывания

- Если вы хотите отсортировать элементы массива в порядке убывания, используйте функцию reverse.

Примеры

В качестве примера используется следующая таблица:

```
mysql> select * from test;

+------+--------------+

| c1   | c2           |

+------+--------------+

|    1 | [4,3,null,1] |

|    2 | NULL         |

|    3 | [null]       |

|    4 | [8,5,1,4]    |

+------+--------------+
```

Сортировать значения столбца `c2` в порядке возрастания.

```
mysql> select c1, array_sort(c2) from test;

+------+------------------+

| c1   | array_sort(`c2`) |

+------+------------------+

|    1 | [null,1,3,4]     |

|    2 | NULL             |

|    3 | [null]           |

|    4 | [1,4,5,8]        |

+------+------------------+
```

Примеры с Lambda-компаратором

Отсортировать массив целых чисел в порядке убывания, используя lambda-компаратор:

```
mysql> select array_sort([3,1,4,2], (x,y) -> x > y);
+-----------------------------------------+
| array_sort([3, 1, 4, 2], (x, y) -> x > y) |
+-----------------------------------------+
| [4, 3, 2, 1]                            |
+-----------------------------------------+
```

Отсортировать массив целых чисел, используя числовые возвращаемые значения:

```
mysql> select array_sort([3,1,4,2], (x,y) -> x - y);
+-----------------------------------------+
| array_sort([3, 1, 4, 2], (x, y) -> x - y) |
+-----------------------------------------+
| [1, 2, 3, 4]                            |
+-----------------------------------------+
```

Отсортировать массив строк по длине:

```
mysql> select array_sort(['apple', 'banana', 'cherry', 'date'], (x,y) ->
length(x) - length(y));
+-----------------------------------------------------------------------------
--+
| array_sort(['apple', 'banana', 'cherry', 'date'], (x, y) -> length(x) -
length(y)) |
+-----------------------------------------------------------------------------
--+
| ['date', 'apple', 'banana', 'cherry']
|
+-----------------------------------------------------------------------------
--+
```

Отсортировать массив строк в алфавитном порядке, используя булев возврат:

```
mysql> select array_sort(['banana', 'apple', 'cherry'], (x,y) -> x > y);
+-------------------------------------------------------+
| array_sort(['banana', 'apple', 'cherry'], (x, y) -> x > y) |
+-------------------------------------------------------+
| ['apple', 'banana', 'cherry']                          |
+-------------------------------------------------------+
```

Отсортировать массив с пользовательской бизнес-логикой (сначала четные числа, затем нечетные):

```
mysql> select array_sort([5,2,8,1,9,4], (x,y) ->
    case when x % 2 = 0 and y % 2 = 0 then x - y
         when x % 2 = 0 and y % 2 = 1 then -1
         when x % 2 = 1 and y % 2 = 0 then 1
         else x - y end);
+-----------------------------------------------------------------------------
-----------------------------------------+
| array_sort([5, 2, 8, 1, 9, 4], (x, y) -> case when x % 2 = 0 and y % 2 =
0 then x - y when x % 2 = 0 and y % 2 = 1 then -1 when x % 2 = 1 and y % 2
= 0 then 1 else x - y end) |
+-----------------------------------------------------------------------------
-----------------------------------------+
| [2, 4, 8, 1, 5, 9]
|
+-----------------------------------------------------------------------------
-----------------------------------------+
```

Отсортировать массив десятичных дробей с пользовательским сравнением точности:

```
mysql> select array_sort([3.141, 2.718, 1.414, 2.236], (x,y) -> round(x, 2)
- round(y, 2));
+-----------------------------------------------------------------------------
--------------+
| array_sort([3.141, 2.718, 1.414, 2.236], (x, y) -> round(x, 2) - round(y,
2))         |
+-----------------------------------------------------------------------------
--------------+
| [1.414, 2.236, 2.718, 3.141]
|
+-----------------------------------------------------------------------------
--------------+
```

Array_sortby

Сортирует элементы массива в соответствии с порядком возрастания элементов в другом массиве или массиве, преобразованном из lambda-выражения. Для получения дополнительной информации см. Lambda-выражение. Эта функция поддерживается начиная с версии 2.5.

Элементы в двух массивах подобны парам ключ-значение. Например, b = [7,5,6] является ключом сортировки для a = [3,1,4]. В соответствии с отношением ключ-значение, элементы в двух массивах имеют следующее взаимно однозначное соответствие.

Массив	Элемент 1	Элемент 2	Элемент 3
a	3	1	4
b	7	5	6

После того, как массив `b` отсортирован в порядке возрастания, он становится [5,6,7]. Массив `a` соответствующим образом становится [1,4,3].

Массив	Элемент 1	Элемент 2	Элемент 3
a	1	4	3
b	5	6	7

Когда количество массивов, участвующих в сортировке, больше двух, цель этой функции - отсортировать array0 на основе значений в нескольких столбцах массивов (таких как array1, array2, array3 и т.д.). Правила сортировки следующие:

Сначала сравниваются соответствующие элементы в array1; Если они одинаковы, сравниваются соответствующие элементы в array2; И так далее, до последнего столбца массива.

Примечание:

Элементы в массивах, участвующих в сортировке, должны быть сортируемого типа или типа JSON. Размер массивов, участвующих в сортировке, должен соответствовать исходному массиву (за исключением значений NULL).

Объяснение примера

Даны следующие четыре массива:

```
array0 = [1, 2, 3, 4, 5]
array1 = [6, 5, 5, 5, 4]
array2 = ["d", "b", "a", "b", "4"]
array3 = ["2023-01-01", "2023-01-04", "2023-01-03", "2023-01-05", "2023-01-
02"]
```

Шаги сортировки:

1. Сравнить array1:

- Отсортированный порядок индексов array1 - [4, 2, 3, 1, 0], потому что 4 < 5 < 5 < 5 < 6.

2. Для элементов [5, 5, 5] в array1 сравнить array2:

- Отсортированный порядок [a, b, b] в array2 - [2, 3, 4], потому что a < b = b.

3. Для элементов [b, b] в array1 и array2 сравнить array3:

- Отсортированный порядок [2023-01-04, 2023-01-05] в array3 - [2, 4], потому что 2023-01-04 < 2023-01-05.

Окончательный результат сортировки для array0:

```
array0 = [5, 3, 2, 4, 1]
```

Синтаксис

```
array_sortby(array0, array1)
array_sortby(<lambda function>, array0 [, array1...])
array_sortby(array0, array1, [array2, array3...])
```

- `array_sortby(array0, array1)`

  Сортирует `array0` в соответствии с порядком возрастания `array1`.

- `array_sortby(<lambda function>, array0 [, array1...])`

  Сортирует `array0` в соответствии с массивом, возвращенным из lambda-функции.

- `array_sortby(array0, array1, [array2, array3...])`

  Сортирует array0 в порядке возрастания на основе значений нескольких столбцов массивов (таких как array1, array2, array3 и т.д.). Правило сортировки: сначала сравниваются соответствующие элементы array1; если они одинаковы, сравниваются соответствующие элементы array2; и так далее, до последнего столбца массива.

Параметры

- `array0`: массив, который вы хотите отсортировать. Это должен быть массив, выражение массива или `null`. Элементы в массиве должны быть сортируемыми.
- `array1`: массив сортировки, используемый для сортировки `array0`. Это должен быть массив, выражение массива или `null`.
- `lambda function`: lambda-выражение, используемое для генерации массива сортировки.
- `array1, [array2, array3...]`: массив сортировки, используемый для сортировки `array0`. Это должен быть массив, выражение массива или `null`.

Возвращаемое значение

Возвращает массив.

Примечания по использованию

- Эта функция может сортировать элементы массива только в порядке возрастания.
- Значения `NULL` помещаются в начало возвращаемого массива.
- Если вы хотите отсортировать элементы массива в порядке убывания, используйте функцию reverse.
- Если массив сортировки (`array1`) равен null, данные в `array0` остаются неизменными.
- Элементы возвращаемого массива имеют тот же тип данных, что и элементы `array0`. Атрибут null-значений также совпадает.
- Все массивы должны иметь одинаковое количество элементов. В противном случае возвращается ошибка.

Примеры

Следующая таблица используется для демонстрации использования этой функции.

```
CREATE TABLE `test_array` (
  `c1` int(11) NULL COMMENT "",
  `c2` ARRAY<int(11)> NULL COMMENT "",
  `c3` ARRAY<int(11)> NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`c1`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`c1`)
PROPERTIES (
"replication_num" = "3",
"storage_format" = "DEFAULT",
"enable_persistent_index" = "true",
"compression" = "LZ4"
);

insert into test_array values
(1,[4,3,5],[82,1,4]),
(2,null,[23]),
(3,[4,2],[6,5]),
(4,null,null),
(5,[],[]),
(6,NULL,[]),
(7,[],null),
(8,[null,null],[3,6]),
(9,[432,21,23],[5,4,null]);

select * from test_array order by c1;
+------+-------------+------------+
| c1   | c2          | c3         |
+------+-------------+------------+
|    1 | [4,3,5]     | [82,1,4]   |
|    2 | NULL        | [23]       |
|    3 | [4,2]       | [6,5]      |
|    4 | NULL        | NULL       |
|    5 | []          | []         |
|    6 | NULL        | []         |
|    7 | []          | NULL       |
|    8 | [null,null] | [3,6]      |
|    9 | [432,21,23] | [5,4,null] |
+------+-------------+------------+
9 rows in set (0.00 sec)
```

Пример 1: Отсортировать `c3` в соответствии с `c2`. Этот пример также предоставляет результат array_sort() для сравнения.

```
select c1, c3, c2, array_sort(c2), array_sortby(c3,c2)
from test_array order by c1;
+------+------------+-------------+----------------+----------------------+
| c1   | c3         | c2          | array_sort(c2) | array_sortby(c3, c2) |
+------+------------+-------------+----------------+----------------------+
|    1 | [82,1,4]   | [4,3,5]     | [3,4,5]        | [1,82,4]             |
|    2 | [23]       | NULL        | NULL           | [23]                 |
|    3 | [6,5]      | [4,2]       | [2,4]          | [5,6]                |
|    4 | NULL       | NULL        | NULL           | NULL                 |
|    5 | []         | []          | []             | []                   |
|    6 | []         | NULL        | NULL           | []                   |
|    7 | NULL       | []          | []             | NULL                 |
|    8 | [3,6]      | [null,null] | [null,null]    | [3,6]                |
|    9 | [5,4,null] | [432,21,23] | [21,23,432]    | [4,null,5]           |
+------+------------+-------------+----------------+----------------------+
```

Пример 2: Отсортировать массив `c3` на основе `c2`, сгенерированного из lambda-выражения. Этот пример эквивалентен Примеру 1. Он также предоставляет результат array_sort() для сравнения.

```
select
    c1,
    c3,
    c2,
    array_sort(c2) as sorted_c2_asc,
    array_sortby((x,y) -> y, c3, c2) as sorted_c3_by_c2
from test_array order by c1;
+------+------------+-------------+----------------+-----------------+
| c1   | c3         | c2          | sorted_c2_asc | sorted_c3_by_c2 |
+------+------------+-------------+----------------+-----------------+
|    1 | [82,1,4]   | [4,3,5]     | [3,4,5]        | [82,1,4]        |
|    2 | [23]       | NULL        | NULL           | [23]            |
|    3 | [6,5]      | [4,2]       | [2,4]          | [5,6]           |
|    4 | NULL       | NULL        | NULL           | NULL            |
|    5 | []         | []          | []             | []              |
|    6 | []         | NULL        | NULL           | []              |
|    7 | NULL       | []          | []             | NULL            |
|    8 | [3,6]      | [null,null] | [null,null]    | [3,6]           |
|    9 | [5,4,null] | [432,21,23] | [21,23,432]    | [4,null,5]      |
+------+------------+-------------+----------------+-----------------+
```

Пример 3: Отсортировать массив `c3` на основе порядка возрастания `c2+c3`.

```
select
    c3,
    c2,
    array_map((x,y)-> x+y,c3,c2) as sum,
    array_sort(array_map((x,y)-> x+y, c3, c2)) as sorted_sum,
    array_sortby((x,y) -> x+y, c3, c2) as sorted_c3_by_sum
from test_array where c1=1;
+----------+---------+----------+------------+------------------+
| c3       | c2      | sum      | sorted_sum | sorted_c3_by_sum |
+----------+---------+----------+------------+------------------+
| [82,1,4] | [4,3,5] | [86,4,9] | [4,9,86]   | [1,4,82]         |
+----------+---------+----------+------------+------------------+
```

```
CREATE TABLE test_array_sortby_muliti (
    id INT(11) not null,
    array_col1 ARRAY<INT>,
    array_col2 ARRAY<DOUBLE>,
    array_col3 ARRAY<VARCHAR(20)>,
    array_col4 ARRAY<DATE>
) ENGINE=OLAP
DUPLICATE KEY(id)
COMMENT "OLAP"
DISTRIBUTED BY HASH(id)
PROPERTIES (
    "replication_num" = "1",
    "storage_format" = "DEFAULT",
    "enable_persistent_index" = "true",
    "compression" = "LZ4"
);

INSERT INTO test_array_sortby_multi VALUES
(1, [4, 3, 5], [1.1, 2.2, 2.2], ['a', 'b', 'c'], ['2023-01-01', '2023-01-
02', '2023-01-03']),
(2, [6, 7, 8], [6.6, 5.5, 6.6], ['d', 'e', 'd'], ['2023-01-04', '2023-01-
05', '2023-01-06']),
(3, NULL, [7.7, 8.8, 8.8], ['g', 'h', 'h'], ['2023-01-07', '2023-01-08',
'2023-01-09']),
(4, [9, 10, 11], NULL, ['k', 'k', 'j'], ['2023-01-10', '2023-01-12', '2023-
01-11']),
(5, [12, 13, 14], [10.10, 11.11, 11.11], NULL, ['2023-01-13', '2023-01-14',
'2023-01-15']),
(6, [15, 16, 17], [14.14, 13.13, 14.14], ['m', 'o', 'o'], NULL),
(7, [18, 19, 20], [16.16, 16.16, 18.18], ['p', 'p', 'r'], ['2023-01-16',
NULL, '2023-01-18']),
(8, [21, 22, 23], [19.19, 20.20, 19.19], ['a', 't', 'a'], ['2023-01-19',
'2023-01-20', '2023-01-21']),
(9, [24, 25, 26], NULL, ['y', 'y', 'z'], ['2023-01-25', '2023-01-24',
'2023-01-26']),
(10, [24, 25, 26], NULL, ['y', 'y', 'z'], ['2023-01-25', NULL, '2023-01-
26']);


select * from test_array_sortby_multi order by id asc;
+------+------------+---------------------+---------------+----------------
--------------------------+
| id   | array_col1 | array_col2          | array_col3    | array_col4
|
+------+------------+---------------------+---------------+----------------
```

```
---------------------------+
|    1 | [4,3,5]    | [1.1,2.2,2.2]       | ["a","b","c"] | ["2023-01-
01","2023-01-02","2023-01-03"] |
|    2 | [6,7,8]    | [6.6,5.5,6.6]       | ["d","e","d"] | ["2023-01-
04","2023-01-05","2023-01-06"] |
|    3 | NULL       | [7.7,8.8,8.8]       | ["g","h","h"] | ["2023-01-
07","2023-01-08","2023-01-09"] |
|    4 | [9,10,11]  | NULL                | ["k","k","j"] | ["2023-01-
10","2023-01-12","2023-01-11"] |
|    5 | [12,13,14] | [10.1,11.11,11.11]  | NULL          | ["2023-01-
13","2023-01-14","2023-01-15"] |
|    6 | [15,16,17] | [14.14,13.13,14.14] | ["m","o","o"] | NULL
|
|    7 | [18,19,20] | [16.16,16.16,18.18] | ["p","p","r"] | ["2023-01-
16",null,"2023-01-18"]         |
|    8 | [21,22,23] | [19.19,20.2,19.19]  | ["a","t","a"] | ["2023-01-
19","2023-01-20","2023-01-21"] |
|    9 | [24,25,26] | NULL                | ["y","y","z"] | ["2023-01-
25","2023-01-24","2023-01-26"] |
|   10 | [24,25,26] | NULL                | ["y","y","z"] | ["2023-01-
25",null,"2023-01-26"]         |
+------+------------+---------------------+---------------+----------------
---------------------------+
```

Пример 1: Отсортировать `array_col1` в соответствии с `array_col2`, `array_col3`.

```
select id, array_col1, array_col2, array_col3, array_sortby(array_col1,
array_col2, array_col3) from test_array_sortby_multi order by id asc;
+------+------------+---------------------+---------------+----------------
----------------------------------------+
| id   | array_col1 | array_col2          | array_col3    |
array_sortby(array_col1, array_col2, array_col3)       |
+------+------------+---------------------+---------------+----------------
----------------------------------------+
|    1 | [4,3,5]    | [1.1,2.2,2.2]       | ["a","b","c"] | [4,3,5]
|
|    2 | [6,7,8]    | [6.6,5.5,6.6]       | ["d","e","d"] | [7,6,8]
|
|    3 | NULL       | [7.7,8.8,8.8]       | ["g","h","h"] | NULL
|
|    4 | [9,10,11]  | NULL                | ["k","k","j"] | [11,9,10]
|
|    5 | [12,13,14] | [10.1,11.11,11.11]  | NULL          | [12,13,14]
|
|    6 | [15,16,17] | [14.14,13.13,14.14] | ["m","o","o"] | [16,15,17]
|
|    7 | [18,19,20] | [16.16,16.16,18.18] | ["p","p","r"] | [18,19,20]
|
|    8 | [21,22,23] | [19.19,20.2,19.19]  | ["a","t","a"] | [21,23,22]
|
|    9 | [24,25,26] | NULL                | ["y","y","z"] | [24,25,26]
|
|   10 | [24,25,26] | NULL                | ["y","y","z"] | [24,25,26]
|
+------+------------+---------------------+---------------+----------------
----------------------------------------+
```

Пример 2: Отсортировать `array_col1` в соответствии с `array_col2`, `array_col3`, `array_col4`.

```
select id, array_col1, array_col2, array_col3, array_col4,
array_sortby(array_col1, array_col2, array_col3, array_col4) from
test_array_sortby_multi order by id asc;
+------+------------+---------------------+---------------+----------------
--------------------------+-------------------------------------------------
--------------------+
| id   | array_col1 | array_col2          | array_col3    | array_col4
| array_sortby(array_col1, array_col2, array_col3, array_col4)       |
+------+------------+---------------------+---------------+----------------
--------------------------+-------------------------------------------------
--------------------+
|    1 | [4,3,5]    | [1.1,2.2,2.2]       | ["a","b","c"] | ["2023-01-
01","2023-01-02","2023-01-03"] | [4,3,5]
|
|    2 | [6,7,8]    | [6.6,5.5,6.6]       | ["d","e","d"] | ["2023-01-
04","2023-01-05","2023-01-06"] | [7,6,8]
|
|    3 | NULL       | [7.7,8.8,8.8]       | ["g","h","h"] | ["2023-01-
07","2023-01-08","2023-01-09"] | NULL
|
|    4 | [9,10,11]  | NULL                | ["k","k","j"] | ["2023-01-
10","2023-01-12","2023-01-11"] | [11,9,10]
|
|    5 | [12,13,14] | [10.1,11.11,11.11]  | NULL          | ["2023-01-
13","2023-01-14","2023-01-15"] | [12,13,14]
|
|    6 | [15,16,17] | [14.14,13.13,14.14] | ["m","o","o"] | NULL
| [16,15,17]                                                          |
|    7 | [18,19,20] | [16.16,16.16,18.18] | ["p","p","r"] | ["2023-01-
16",null,"2023-01-18"]         | [19,18,20]
|
|    8 | [21,22,23] | [19.19,20.2,19.19]  | ["a","t","a"] | ["2023-01-
19","2023-01-20","2023-01-21"] | [21,23,22]
|
|    9 | [24,25,26] | NULL                | ["y","y","z"] | ["2023-01-
25","2023-01-24","2023-01-26"] | [25,24,26]
|
|   10 | [24,25,26] | NULL                | ["y","y","z"] | ["2023-01-
25",null,"2023-01-26"]         | [25,24,26]
|
+------+------------+---------------------+---------------+----------------
--------------------------+-------------------------------------------------
--------------------+
```

Ссылки

array_sort

Array_sum

Суммирует все элементы в массиве.

Начиная с Selena 2.5, array_sum() может принимать lambda-выражение в качестве аргумента. Однако она не может работать напрямую с Lambda-выражением. Она должна работать с результатом, преобразованным из array_map().

Синтаксис

```
array_sum(array(type))
array_sum(lambda_function, arr1,arr2...) =
array_sum(array_map(lambda_function, arr1,arr2...))
```

Параметры

- `array(type)`: массив, для которого вы хотите вычислить сумму. Элементы массива поддерживают следующие типы данных: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE и DECIMALV2.
- `lambda_function`: lambda-выражение, используемое для вычисления целевого массива для array_sum().

Возвращаемое значение

Возвращает числовое значение.

Примеры

Использование array_sum без lambda-функции

```
mysql> select array_sum([11, 11, 12]);
+-----------------------+
| array_sum([11,11,12]) |
+-----------------------+
| 34                    |
+-----------------------+

mysql> select array_sum([11.33, 11.11, 12.324]);
+---------------------------------+
| array_sum([11.33,11.11,12.324]) |
+---------------------------------+
| 34.764                          |
+---------------------------------+
```

Использование array_sum с lambda-функцией

```
-- Умножить [1,2,3] на [1,2,3] и суммировать элементы.
select array_sum(array_map(x->x*x,[1,2,3]));
+----------------------------------------------+
| array_sum(array_map(x -> x * x, [1, 2, 3])) |
+----------------------------------------------+
|                                           14 |
+----------------------------------------------+
```

Keyword

ARRAY_SUM,ARRAY

Array_to_bitmap

Преобразует массив в значения BITMAP. Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
BITMAP array_to_bitmap(array)
```

Параметры

`array`: Элементы массива могут быть типа BIGINT, INT, TINYINT или SMALLINT.

Возвращаемое значение

Возвращает значение типа BITMAP.

Примечания по использованию

- Если тип данных элементов входного массива недействителен, например STRING или DECIMAL, возвращается ошибка.
- Если введен пустой массив, возвращается пустое значение BITMAP.
- Если введен `NULL`, возвращается `NULL`.

Примеры

Пример 1: Преобразование массива в значения BITMAP. Эта функция должна быть вложена в `bitmap_to_array`, поскольку значения BITMAP не могут быть отображены.

```
MySQL > select bitmap_to_array(array_to_bitmap([1,2,3]));
+--------------------------------------------+
| bitmap_to_array(array_to_bitmap([1,2,3])) |
+--------------------------------------------+
| [1,2,3]                                    |
+--------------------------------------------+
```

Пример 2: Ввод пустого массива и возврат пустого массива.

```
MySQL > select bitmap_to_array(array_to_bitmap([]));
+---------------------------------------+
| bitmap_to_array(array_to_bitmap([])) |
+---------------------------------------+
| []                                    |
+---------------------------------------+
```

Пример 3: Ввод `NULL` и возврат `NULL`.

```
MySQL > select array_to_bitmap(NULL);
+-----------------------+
| array_to_bitmap(NULL) |
+-----------------------+
| NULL                  |
+-----------------------+
```

Array_unique_agg

Агрегирует уникальные значения (включая `NULL`) из столбца ARRAY в массив (из нескольких строк в одну строку).

Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
ARRAY_UNIQUE_AGG(col)
```

Параметры

- `col`: столбец, значения которого вы хотите агрегировать. Поддерживаемый тип данных - ARRAY.

Возвращаемое значение

Возвращает значение типа ARRAY.

Примечания по использованию

- Порядок элементов в массиве случайный.
- Тип данных элементов в возвращаемом массиве совпадает с типом данных элементов во входном столбце.
- Возвращает `NULL`, если нет совпадающих значений.

Примеры

Рассмотрим следующую таблицу данных в качестве примера:

```
mysql > select * from array_unique_agg_example;
+------+--------------+
| a    | b            |
+------+--------------+
|    2 | [1,null,2,4] |
|    2 | [1,null,3]   |
|    1 | [1,1,2,3]    |
|    1 | [2,3,4]      |
+------+--------------+
```

Пример 1: Группировка значений в столбце `a` и агрегация уникальных значений в столбце `b` в массив.

```
mysql > select a, array_unique_agg(b) from array_unique_agg_example group
by a;
+------+---------------------+
| a    | array_unique_agg(b) |
+------+---------------------+
|    1 | [4,1,2,3]           |
|    2 | [4,1,2,3,null]      |
+------+---------------------+
```

Пример 2: Агрегация значений в столбце `b` с использованием предложения WHERE. Если никакие данные не соответствуют условию фильтра, возвращается значение `NULL`.

```
mysql > select array_unique_agg(b) from array_unique_agg_example where a <
0;
+---------------------+
| array_unique_agg(b) |
+---------------------+
| NULL                |
+---------------------+
```

Ключевые слова

ARRAY_UNIQUE_AGG, ARRAY

Arrays_overlap

Проверяет, является ли пересечение двух массивов пустым. Указанные массивы должны быть одинакового типа данных. Если пересечение пустое, эта функция возвращает `0`. В противном случае эта функция возвращает `1`.

Синтаксис

```
arrays_overlap(input0, input1)
```

Параметры

- `input0`: один из двух массивов, которые вы хотите сравнить.
- `input1`: другой из двух массивов, которые вы хотите сравнить. Значение этого параметра должно быть того же типа данных, что и значение параметра `input0`.

Возвращаемое значение

Возвращает значение типа данных BOOLEAN.

Примеры

Пример 1:

```
mysql> select arrays_overlap([11, 9, 3, 2], [null, 11]);
+---------------------------------------+
| arrays_overlap([11,9,3,2], [NULL,11]) |
+---------------------------------------+
|                                     1 |
+---------------------------------------+
```

Пример 2:

```
mysql> select arrays_overlap([9, 3, 2], [null, 11]);
+-----------------------------------+
| arrays_overlap([9,3,2], [NULL,11]) |
+-----------------------------------+
|                                 0 |
+-----------------------------------+
```

Пример 3:

```
mysql> select arrays_overlap([9, 3, null, 2], [null, 11]);
+----------------------------------------+
| arrays_overlap([9,3,NULL,2], [NULL,11]) |
+----------------------------------------+
|                                      1 |
+----------------------------------------+
```

Пример 4:

```
mysql> select arrays_overlap([9, 3, "SQL", 2], [null, "SQL"]);
+----------------------------------------------+
| arrays_overlap([9,3,'SQL',2], [NULL,'SQL']) |
+----------------------------------------------+
|                                            1 |
+----------------------------------------------+
```

Cardinality

Возвращает количество элементов в массиве. Тип результата - INT. Если входной параметр - NULL, результат также NULL. NULL-элементы учитываются в длине.

Это псевдоним функции array_length().

Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
INT cardinality(any_array)
```

Параметры

`any_array`: значение ARRAY, из которого вы хотите получить количество элементов.

Возвращаемое значение

Возвращает значение типа INT.

Примеры

```
mysql> select cardinality([1,2,3]);
+-----------------------+
|  cardinality([1,2,3]) |
+-----------------------+
|                     3 |
+-----------------------+
1 row in set (0.00 sec)

mysql> select cardinality([1,2,3,null]);
+------------------------------+
| cardinality([1, 2, 3, NULL]) |
+------------------------------+
|                            4 |
+------------------------------+

mysql> select cardinality([[1,2], [3,4]]);
+----------------------------+
|  cardinality([[1,2],[3,4]]) |
+----------------------------+
|                          2 |
+----------------------------+
1 row in set (0.01 sec)
```

Ключевые слова

CARDINALITY, ARRAY_LENGTH, ARRAY

Element_at

Возвращает элемент в указанной позиции (индексе) из заданного массива. Если любой из параметров равен NULL или если позиция не существует, результатом будет NULL.

Эта функция является псевдонимом оператора индексации `[]`. Поддерживается начиная с версии v1.5.2.

Если вы хотите получить значение из пары ключ-значение в map, см. element_at.

Синтаксис

```
element_at(any_array, position)
```

Параметры

- `any_array`: выражение массива, из которого необходимо извлечь элементы.
- `position`: позиция элемента в массиве. Должна быть положительным целым числом. Диапазон значений: [1, длина массива]. Если `position` не существует, возвращается NULL.

Примеры

```
mysql> select element_at([2,3,11],3);
+---------------+
| [2, 3, 11][3] |
+---------------+
|            11 |
+---------------+

mysql> select element_at(["a","b","c"],1);
+--------------------+
| ['a', 'b', 'c'][1] |
+--------------------+
| a                  |
+--------------------+
```

Keyword

ELEMENT_AT, ARRAY

Reverse

Переворачивает строку или массив. Возвращает строку или массив с символами строки или элементами массива в обратном порядке.

Синтаксис

```
reverse(param)
```

Параметры

`param`: строка или массив для переворота. Может быть типа VARCHAR, CHAR или ARRAY.

В настоящее время эта функция поддерживает только одномерные массивы, и элементы массива не могут быть типа DECIMAL. Функция поддерживает следующие типы элементов массива: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, VARCHAR, DECIMALV2, DATETIME, DATE и JSON. **JSON поддерживается начиная с версии 2.5.**

Возвращаемое значение

Возвращаемый тип совпадает с типом `param`.

Примеры

Пример 1: Переворот строки.

```
MySQL > SELECT REVERSE('hello');
+------------------+
| REVERSE('hello') |
+------------------+
| olleh            |
+------------------+
1 row in set (0.00 sec)
```

Пример 2: Переворот массива.

```
MYSQL> SELECT REVERSE([4,1,5,8]);
+--------------------+
| REVERSE([4,1,5,8]) |
+--------------------+
| [8,5,1,4]          |
+--------------------+
```

Unnest

UNNEST — это табличная функция, которая принимает массив и преобразует элементы в нем в несколько строк таблицы. Такое преобразование также известно как «развертывание» (flattening).

Вы можете использовать Lateral Join с UNNEST для реализации распространенных преобразований, например, из STRING, ARRAY или BITMAP в несколько строк. Для получения дополнительной информации см. Lateral join.

Начиная с версии v1.5.2, UNNEST может принимать переменное количество параметров массивов. Массивы могут различаться по типу и длине (количеству элементов). Если массивы имеют разную длину, применяется наибольшая длина, что означает, что в массивы меньшей длины будут добавлены значения null. Для получения дополнительной информации см. Пример 2.

Начиная с версии v1.5.2, UNNEST можно использовать с LEFT JOIN ON TRUE, что позволяет сохранить все строки левой таблицы, даже если соответствующие строки правой таблицы пустые или содержат значения null. Для таких пустых или NULL строк возвращаются значения NULL. Для получения дополнительной информации см. Пример 3.

Синтаксис

```
unnest(array0[, array1 ...])
```

Параметры

`array`: массив, который вы хотите преобразовать. Должен быть массивом или выражением, вычисляющимся в тип данных ARRAY. Вы можете указать один или несколько массивов или выражений массивов.

Возвращаемое значение

Возвращает несколько строк, преобразованных из массива. Тип возвращаемого значения зависит от типов элементов в массиве.

Типы элементов, поддерживаемые в массиве, см. в ARRAY.

Примечания по использованию

- UNNEST — это табличная функция. Ее необходимо использовать с Lateral Join, но ключевое слово Lateral Join не нужно указывать явно.
- Если выражение массива вычисляется в NULL или если массив пуст, строки не будут возвращены (за исключением LEFT JOIN ON TRUE).
- Если элемент в массиве является NULL, для этого элемента возвращается NULL.

Примеры

Пример 1: UNNEST принимает один параметр

```
-- Создание таблицы student_score, где scores является столбцом ARRAY.
CREATE TABLE student_score
(
    `id` bigint(20) NULL COMMENT "",
    `scores` ARRAY<int> NULL COMMENT ""
)
DUPLICATE KEY (id)
DISTRIBUTED BY HASH(`id`);

-- Вставка данных в эту таблицу.
INSERT INTO student_score VALUES
(1, [80,85,87]),
(2, [77, null, 89]),
(3, null),
(4, []),
(5, [90,92]);

-- Запрос данных из этой таблицы.
SELECT * FROM student_score ORDER BY id;
+------+--------------+
| id   | scores       |
+------+--------------+
|    1 | [80,85,87]   |
|    2 | [77,null,89] |
|    3 | NULL         |
|    4 | []           |
|    5 | [90,92]      |
+------+--------------+

-- Использование UNNEST для развертывания столбца scores в несколько строк.
SELECT id, scores, unnest FROM student_score, unnest(scores) AS unnest;
+------+--------------+--------+
| id   | scores       | unnest |
+------+--------------+--------+
|    1 | [80,85,87]   |     80 |
|    1 | [80,85,87]   |     85 |
|    1 | [80,85,87]   |     87 |
|    2 | [77,null,89] |     77 |
|    2 | [77,null,89] |   NULL |
|    2 | [77,null,89] |     89 |
|    5 | [90,92]      |     90 |
```

```
|     5 | [90,92]       |      92 |
+-------+---------------+---------+
```

[80,85,87], соответствующий `id = 1`, преобразуется в три строки.

[77,null,89], соответствующий `id = 2`, сохраняет значение null.

`scores`, соответствующие `id = 3` и `id = 4`, являются NULL и пустыми, поэтому они пропускаются.

Пример 2: UNNEST принимает несколько параметров

```
-- Создание таблицы example_table, где столбцы type и scores различаются по типу.
CREATE TABLE example_table (
id varchar(65533) NULL COMMENT "",
type varchar(65533) NULL COMMENT "",
scores ARRAY<int> NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(id)
COMMENT "OLAP"
DISTRIBUTED BY HASH(id)
PROPERTIES (
"replication_num" = "3");

-- Вставка данных в таблицу.
INSERT INTO example_table VALUES
("1", "typeA;typeB", [80,85,88]),
("2", "typeA;typeB;typeC", [87,90,95]);

-- Запрос данных из таблицы.
SELECT * FROM example_table;
+------+-------------------+------------+
| id   | type              | scores     |
+------+-------------------+------------+
| 1    | typeA;typeB       | [80,85,88] |
| 2    | typeA;typeB;typeC | [87,90,95] |
+------+-------------------+------------+

-- Использование UNNEST для преобразования type и scores в несколько строк.
SELECT id, unnest.type, unnest.scores
FROM example_table, unnest(split(type, ";"), scores) AS
unnest(type,scores);
+------+-------+--------+
| id   | type  | scores |
+------+-------+--------+
| 1    | typeA |     80 |
| 1    | typeB |     85 |
| 1    | NULL  |     88 |
| 2    | typeA |     87 |
| 2    | typeB |     90 |
| 2    | typeC |     95 |
+------+-------+--------+
```

`type` и `scores` в `UNNEST` различаются по типу и длине.

`type` — это столбец VARCHAR, а `scores` — столбец ARRAY. Функция split() используется для преобразования `type` в ARRAY.

Для `id = 1`, `type` преобразуется в ["typeA","typeB"], который имеет два элемента.

Для `id = 2`, `type` преобразуется в ["typeA","typeB","typeC"], который имеет три элемента.

Чтобы обеспечить согласованное количество строк для каждого `id`, к ["typeA","typeB"] добавляется null элемент.

Пример 3: UNNEST с LEFT JOIN ON TRUE

```
-- Создание таблицы student_score, где scores является столбцом ARRAY.
CREATE TABLE student_score
(
`id` bigint(20) NULL COMMENT "",
`scores` ARRAY<int> NULL COMMENT ""
)
DUPLICATE KEY (id)
DISTRIBUTED BY HASH(`id`)
PROPERTIES (
"replication_num" = "1"
);

-- Вставка данных в эту таблицу.
INSERT INTO student_score VALUES
(1, [80,85,87]),
(2, [77, null, 89]),
(3, null),
(4, []),
(5, [90,92]);

-- Запрос данных из этой таблицы.
SELECT * FROM student_score ORDER BY id;
+------+--------------+
| id   | scores       |
+------+--------------+
|    1 | [80,85,87]   |
|    2 | [77,null,89] |
|    3 | NULL         |
|    4 | []           |
|    5 | [90,92]      |
+------+--------------+

-- Использование LEFT JOIN ON TRUE.
SELECT id, scores, unnest FROM student_score LEFT JOIN unnest(scores) AS
unnest ON TRUE ORDER BY 1, 3;
+------+--------------+--------+
| id   | scores       | unnest |
+------+--------------+--------+
|  1   | [80,85,87]   |     80 |
|  1   | [80,85,87]   |     85 |
|  1   | [80,85,87]   |     87 |
|  2   | [77,null,89] |   NULL |
|  2   | [77,null,89] |     77 |
```

```
|  2   | [77,null,89] |     89 |
|  3   | NULL         |   NULL |
|  4   | []           |   NULL |
|  5   | [90,92]      |     90 |
|  5   | [90,92]      |     92 |
+------+--------------+--------+
```

[80,85,87], соответствующий `id = 1`, преобразуется в три строки.

Значение null в [77,null,89], соответствующее `id = 2`, сохраняется.

`scores`, соответствующие `id = 3` и `id = 4`, являются NULL и пустыми. Left Join сохраняет эти две строки и возвращает для них NULL.

2.9 Binary

From_binary

Преобразует бинарное значение в строку VARCHAR на основе указанного бинарного формата (`binary_type`). Поддерживаются следующие бинарные форматы: `hex`, `encode64` и `utf8`. Если `binary_type` не указан, по умолчанию используется `hex`.

Синтаксис

```
from_binary(binary[, binary_type])
```

Параметры

- `binary`: входные бинарные данные для преобразования, обязательный параметр.
- `binary_type`: бинарный формат для преобразования, необязательный параметр.
  - `hex` (по умолчанию): `from_binary` использует метод `hex` для кодирования входных бинарных данных в строку VARCHAR.
  - `encode64`: `from_binary` использует метод `base64` для кодирования входных бинарных данных в строку VARCHAR.
  - `utf8`: `from_binary` преобразует входные бинарные данные в строку VARCHAR без каких-либо преобразований.

Возвращаемое значение

Возвращает строку VARCHAR.

Примеры

```
mysql> select from_binary(to_binary('ABAB', 'hex'), 'hex');
+----------------------------------------------+
| from_binary(to_binary('ABAB', 'hex'), 'hex') |
+----------------------------------------------+
| ABAB                                         |
+----------------------------------------------+
1 row in set (0.02 sec)

mysql> select from_base64(from_binary(to_binary('U1RBUlJPQ0tT',
'encode64'), 'encode64'));
+-------------------------------------------------------------------------
---+
| from_base64(from_binary(to_binary('U1RBUlJPQ0tT', 'encode64'),
'encode64')) |
+-------------------------------------------------------------------------
---+
| SELENA
|
+-------------------------------------------------------------------------
---+
1 row in set (0.01 sec)

mysql> select from_binary(to_binary('SELENA', 'utf8'), 'utf8');
+-----------------------------------------------------+
| from_binary(to_binary('SELENA', 'utf8'), 'utf8') |
+-----------------------------------------------------+
| SELENA                                              |
+-----------------------------------------------------+
1 row in set (0.01 sec)
```

Ссылки

- to_binary
- Тип данных BINARY/VARBINARY

To_binary

Преобразует строку VARCHAR в двоичное значение на основе указанного двоичного формата (`binary_type`) входной строки. Поддерживаются следующие двоичные форматы: `hex`, `encode64` и `utf8`. Если `binary_type` не указан, по умолчанию используется `hex`.

Синтаксис

```
to_binary(str[, binary_type])
```

Параметры

- `str`: строка для преобразования, обязательный параметр. Должна быть строкой VARCHAR.

- `binary_type`: двоичный формат для преобразования, необязательный параметр.

  - `hex` (по умолчанию): `to_binary` предполагает, что входная строка является шестнадцатеричной строкой, в которой все символы находятся в диапазоне '0123456789abcdef'. Если входная строка недействительна, возвращается пустое двоичное значение (исключения не генерируются). `to_binary` преобразует входную строку в двоичное представление напрямую. Например, `"abab"` будет преобразовано в `x'abab'`. Входная строка нечувствительна к регистру.
  - `encode64`: `to_binary` предполагает, что входная строка является строкой, закодированной в base64. Если входная строка недействительна, возвращается пустое двоичное значение (исключения не генерируются). `to_binary` декодирует строку base64 в двоичный результат. Например, `"YWJhYg=="` будет преобразовано в `x'abab'`.
  - `utf8`: `to_binary` преобразует входную строку в двоичное значение без какого-либо преобразования.

Возвращаемое значение

Возвращает значение типа VARCHAR.

Примеры

В следующих примерах предполагается, что при подключении к Selena из клиента MySQL включена опция `--binary-as-hex`. Таким образом, двоичные данные отображаются в шестнадцатеричной нотации.

```
mysql> select to_binary('ABAB', 'hex');
+-----------------------------------------------------+
| to_binary('ABAB', 'hex')                            |
+-----------------------------------------------------+
| 0xABAB                                              |
+-----------------------------------------------------+
1 row in set (0.01 sec)

mysql> select to_binary('U1RBUlJPQ0tT', 'encode64');
+-----------------------------------------------------------------------------
----+
| to_binary('U1RBUlJPQ0tT', 'encode64')
|
+-----------------------------------------------------------------------------
----+
| 0x53544152524F434B53
|
+-----------------------------------------------------------------------------
----+
1 row in set (0.00 sec)

mysql> select to_binary('SELENA', 'utf8');
+-----------------------------------------------------------------+
| to_binary('SELENA', 'utf8')                                     |
+-----------------------------------------------------------------+
| 0x53544152524F434B53                                            |
+-----------------------------------------------------------------+
1 row in set (0.00 sec)

-- Входная строка не соответствует двоичному формату, и возвращается пустое
двоичное значение.

mysql> select to_binary('U1RBUlJPQ0tT', 'hex');
+-----------------------------------------------------------------------+
| to_binary('U1RBUlJPQ0tT', 'hex')                                      |
+-----------------------------------------------------------------------+
| 0x                                                                    |
+-----------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Ссылки

- from_binary
- Тип данных BINARY/VARBINARY

2.10 Bit

Bit_shift_left

Сдвигает двоичное представление числового выражения влево на указанное количество бит.

Эта функция выполняет **арифметический сдвиг влево**, при котором длина в битах не изменяется, в конец добавляется 0, а старший бит остается неизменным. Например, сдвиг `10101` влево на один бит дает `11010`.

Синтаксис

```
bit_shift_left(value, shift)
```

Параметры

`value`: значение или числовое выражение для сдвига. Поддерживаемые типы данных: TINYINT, SMALLINT, INT, BIGINT и LARGEINT.

`shift`: количество бит для сдвига. Поддерживаемый тип данных — BIGINT.

Возвращаемое значение

Возвращает значение того же типа, что и `value`.

Примечания по использованию

- Если любой входной параметр равен NULL, возвращается NULL.
- Если `shift` меньше 0, возвращается 0.
- Сдвиг `value` на `0` всегда возвращает исходное значение `value`.
- Сдвиг `0` на любое значение `shift` всегда возвращает `0`.
- Если тип данных `value` является числовым, но не целым числом, значение будет приведено к целому числу. См. Примеры.
- Если тип данных `value` является строкой, значение будет приведено к целому числу, если это возможно. Если значение не может быть приведено к целому числу, оно будет обработано как NULL. См. Примеры.

Примеры

Использование этой функции для сдвига числовых значений.

```
SELECT bit_shift_left(2, 1);
+----------------------+
| bit_shift_left(2, 1) |
+----------------------+
|                    4 |
+----------------------+

SELECT bit_shift_left(2.2, 1);
+------------------------+
| bit_shift_left(2.2, 1) |
+------------------------+
|                      4 |
+------------------------+

SELECT bit_shift_left("2", 1);
+------------------------+
| bit_shift_left('2', 1) |
+------------------------+
|                      4 |
+------------------------+

SELECT bit_shift_left(-2, 1);
+-----------------------+
| bit_shift_left(-2, 1) |
+-----------------------+
|                    -4 |
+-----------------------+
```

См. также

- bit_shift_right
- bit_shift_right_logical

Bit_shift_right_logical

Сдвигает двоичное представление числового выражения вправо на указанное количество бит.

Эта функция выполняет **логический сдвиг вправо**, при котором длина в битах не изменяется, младший бит отбрасывается, а к старшему биту добавляется 0, независимо от того, является ли исходное значение положительным или отрицательным. **Логические** сдвиги являются беззнаковыми. Например, сдвиг `10101` на один бит дает `00101`.

bit_shift_right() и bit_shift_right_logical() возвращают одинаковый результат для положительных значений.

Синтаксис

```
bit_shift_right_logical(value, shift)
```

Параметры

`value`: значение или числовое выражение для сдвига. Поддерживаемые типы данных: TINYINT, SMALLINT, INT, BIGINT и LARGEINT.

`shift`: количество бит для сдвига. Поддерживаемый тип данных — BIGINT.

Возвращаемое значение

Возвращает значение того же типа, что и `value`.

Примечания по использованию

- Если любой входной параметр равен NULL, возвращается NULL.
- Если `shift` меньше 0, возвращается 0.
- Сдвиг `value` на `0` всегда возвращает исходное значение `value`.
- Сдвиг `0` на любое значение `shift` всегда возвращает `0`.
- Если тип данных `value` является числовым, но не целым числом, значение будет приведено к целому числу. См. Примеры.
- Если тип данных `value` является строкой, значение будет приведено к целому числу, если это возможно. Например, строка "2.3" будет приведена к 2. Если значение не может быть приведено к целому числу, оно будет обработано как NULL. См. Примеры.

Примеры

Использование этой функции для сдвига числовых значений.

```
SELECT bit_shift_right_logical(2, 1);
+-------------------------------+
| bit_shift_right_logical(2, 1) |
+-------------------------------+
|                             1 |
+-------------------------------+

SELECT bit_shift_right_logical(2.2, 1);
+---------------------------------+
| bit_shift_right_logical(2.2, 1) |
+---------------------------------+
|                               1 |
+---------------------------------+

SELECT bit_shift_right_logical("2", 1);
+---------------------------------+
| bit_shift_right_logical('2', 1) |
+---------------------------------+
|                               1 |
+---------------------------------+

SELECT bit_shift_right_logical(cast('-2' AS INTEGER(32)), 1);
+-------------------------------------------------+
| bit_shift_right_logical(CAST('-2' AS INT), 1) |
+-------------------------------------------------+
|                                      2147483647 |
+-------------------------------------------------+
```

См. также

- bit_shift_left
- bit_shift_right

Bit_shift_right

Сдвигает двоичное представление числового выражения вправо на указанное количество бит.

Эта функция выполняет **арифметический сдвиг вправо**, при котором длина в битах не изменяется, младший бит отбрасывается, а знаковый бит используется в качестве старшего бита. Например, сдвиг `10101` вправо на один бит дает `11010`.

Синтаксис

```
bit_shift_right(value, shift)
```

Параметры

`value`: значение или числовое выражение для сдвига. Поддерживаемые типы данных: TINYINT, SMALLINT, INT, BIGINT и LARGEINT.

`shift`: количество бит для сдвига. Поддерживаемый тип данных — BIGINT.

Возвращаемое значение

Возвращает значение того же типа, что и `value`.

Примечания по использованию

- Если любой входной параметр равен NULL, возвращается NULL.
- Если `shift` меньше 0, возвращается 0.
- Сдвиг `value` на `0` всегда возвращает исходное значение `value`.
- Сдвиг `0` на любое значение `shift` всегда возвращает `0`.
- Если тип данных `value` является числовым, но не целым числом, значение будет приведено к целому числу. См. Примеры.
- Если тип данных `value` является строкой, значение будет приведено к целому числу, если это возможно. Например, строка "2.3" будет приведена к 2. Если значение не может быть приведено к целому числу, оно будет обработано как NULL. См. Примеры.

Примеры

Использование этой функции для сдвига числовых значений.

```
SELECT bit_shift_right(2, 1);
+------------------------+
| bit_shift_right(2, 1) |
+------------------------+
|                      1 |
+------------------------+

SELECT bit_shift_right(2.2, 1);
+--------------------------+
| bit_shift_right(2.2, 1) |
+--------------------------+
|                        1 |
+--------------------------+

SELECT bit_shift_right("2", 1);
+--------------------------+
| bit_shift_right('2', 1) |
+--------------------------+
|                        1 |
+--------------------------+

SELECT bit_shift_right(-2, 1);
+------------------------+
| bit_shift_right(-2, 1) |
+------------------------+
|                     -1 |
+------------------------+
```

См. также

- bit_shift_left
- bit_shift_right_logical

Bitand

Возвращает результат побитовой операции AND для двух числовых выражений.

Синтаксис

```
BITAND(x,y);
```

Параметры

- `x`: выражение, которое должно иметь один из следующих типов данных: TINYINT, SMALLINT, INT, BIGINT, LARGEINT.
- `y`: выражение, которое должно иметь один из следующих типов данных: TINYINT, SMALLINT, INT, BIGINT, LARGEINT.

> `x` и `y` должны иметь одинаковый тип данных.

Возвращаемое значение

Возвращаемое значение имеет тот же тип, что и `x`. Если любое значение равно NULL, результат будет NULL.

Примеры

```
mysql> select bitand(3,0);
+--------------+
| bitand(3, 0) |
+--------------+
|            0 |
+--------------+
1 row in set (0.01 sec)
```

Bitnot

Возвращает результат побитового отрицания числового выражения.

Синтаксис

```
BITNOT(x);
```

Параметры

`x`: выражение, которое должно иметь один из следующих типов данных: TINYINT, SMALLINT, INT, BIGINT, LARGEINT.

Возвращаемое значение

Возвращаемое значение имеет тот же тип, что и `x`. Если значение равно NULL, результат будет NULL.

Примеры

```
mysql> select bitnot(3);
+-----------+
| bitnot(3) |
+-----------+
|        -4 |
+-----------+
```

Bitor

Возвращает результат побитовой операции OR для двух числовых выражений.

Синтаксис

```
BITOR(x,y);
```

Параметры

- `x`: выражение, которое должно иметь один из следующих типов данных: TINYINT, SMALLINT, INT, BIGINT, LARGEINT.
- `y`: выражение, которое должно иметь один из следующих типов данных: TINYINT, SMALLINT, INT, BIGINT, LARGEINT.

> `x` и `y` должны иметь одинаковый тип данных.

Возвращаемое значение

Возвращаемое значение имеет тот же тип, что и `x`. Если любое значение равно NULL, результат будет NULL.

Примеры

```
mysql> select bitor(3,0);
+-------------+
| bitor(3, 0) |
+-------------+
|           3 |
+-------------+
1 row in set (0.00 sec)
```

Bitxor

Возвращает результат побитовой операции XOR для двух числовых выражений.

Синтаксис

```
BITXOR(x,y);
```

Параметры

- `x`: выражение, которое должно иметь один из следующих типов данных: TINYINT, SMALLINT, INT, BIGINT, LARGEINT.
- `y`: выражение, которое должно иметь один из следующих типов данных: TINYINT, SMALLINT, INT, BIGINT, LARGEINT.

> `x` и `y` должны иметь одинаковый тип данных.

Возвращаемое значение

Возвращаемое значение имеет тот же тип, что и `x`. Если любое значение равно NULL, результат будет NULL.

Примеры

```
mysql> select bitxor(3,0);
+--------------+
| bitxor(3, 0) |
+--------------+
|            3 |
+--------------+
1 row in set (0.00 sec)
```

2.11 Bitmap

Base64_to_bitmap

Перед импортом данных bitmap в Selena необходимо сериализовать данные и закодировать их в строку Base64. При импорте строки Base64 в Selena необходимо преобразовать строку в данные bitmap. Эта функция используется для преобразования строк Base64 в данные bitmap.

Эта функция поддерживается начиная с версии 2.3.

Синтаксис

```
BITMAP base64_to_bitmap(VARCHAR bitmap)
```

Параметры

`bitmap`: поддерживаемый тип данных — VARCHAR. Перед загрузкой данных Bitmap в Selena вы можете использовать Java или C++ для создания объекта BitmapValue, добавления элемента, сериализации данных и кодирования их в строку Base64. Затем передайте строку Base64 в качестве входного параметра в эту функцию.

Возвращаемое значение

Возвращает значение типа BITMAP.

Примеры

Создайте базу данных с именем `bitmapdb` и таблицу с именем `bitmap`. Используйте Stream Load для импорта данных JSON в `bitmap_table`. В процессе импорта используйте

base64_to_bitmap для преобразования строки Base64 из файла JSON в данные bitmap.

1. Создайте базу данных и таблицу в Selena. В этом примере создается таблица Primary Key.

```
CREATE database bitmapdb;
USE bitmapdb;
CREATE TABLE `bitmap_table` (
`tagname` varchar(65533) NOT NULL COMMENT "Tag name",
`tagvalue` varchar(65533) NOT NULL COMMENT "Tag value",
`userid` bitmap NOT NULL COMMENT "User ID"
) ENGINE=OLAP
PRIMARY KEY(`tagname`, `tagvalue`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`tagname`)
PROPERTIES (
"replication_num" = "3",
"storage_format" = "DEFAULT"
);
```

2. Используйте Stream Load для импорта данных JSON в `bitmap_table`.

   Предположим, есть файл JSON с именем **simpledata**. Этот файл имеет следующее содержимое, и `userid` является строкой в кодировке Base64.

```
{
    "tagname": "Product", "tagvalue": "Insurance",
"userid":"AjowAAABAAAAAAACABAAAAABAAIAAwA="
}
```

   Используйте base64_to_bitmap для преобразования `userid` в значение bitmap.

```
curl --location-trusted -u <username>:<password>\
    -H "columns:
c1,c2,c3,tagname=c1,tagvalue=c2,userid=base64_to_bitmap(c3)"\
    -H "label:bitmap123"\
    -H "format: json"\
    -H "jsonpaths: [\"$.tagname\",\"$.tagvalue\",\"$.userid\"]"\
    -T simpleData
http://host:port/api/bitmapdb/bitmap_table/_stream_load
```

3. Запросите данные из `bitmap_table`.

```
mysql> select tagname,tagvalue,bitmap_to_string(userid) from
bitmap_table;
+--------------+----------+----------------------------+
| tagname      | tagvalue | bitmap_to_string(`userid`) |
+--------------+----------+----------------------------+
| Product      | Insurance      | 1,2,3                |
+--------------+----------+----------------------------+
1 rows in set (0.01 sec)
```

Bitmap_agg

Агрегирует значения (исключая NULL) в столбце в bitmap (несколько строк в одну строку).

Синтаксис

```
BITMAP_AGG(col)
```

Параметры

`col`: столбец, значения которого вы хотите агрегировать. Должен иметь один из следующих типов данных: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT и LARGEINT.

Возвращаемое значение

Возвращает значение типа BITMAP.

Примечания по использованию

Если значение в строке меньше 0 или больше 18446744073709551615, это значение будет проигнорировано и не добавлено в Bitmap (см. Пример 3).

Примеры

Рассмотрим следующую таблицу данных:

```
mysql> CREATE TABLE t1_test (
    c1 int,
    c2 boolean,
    c3 tinyint,
    c4 int,
    c5 bigint,
    c6 largeint
    )
DUPLICATE KEY(c1)
DISTRIBUTED BY HASH(c1)
BUCKETS 1
PROPERTIES ("replication_num" = "3");

INSERT INTO t1_test VALUES
    (1, true, 11, 111, 1111, 11111),
    (2, false, 22, 222, 2222, 22222),
    (3, true, 33, 333, 3333, 33333),
    (4, null, null, null, null, null),
    (5, -1, -11, -111, -1111, -11111),
    (6, null, null, null, null, "36893488147419103232");

select * from t1_test order by c1;
+------+------+------+------+-------+----------------------+
| c1   | c2   | c3   | c4   | c5    | c6                   |
+------+------+------+------+-------+----------------------+
|    1 |    1 |   11 |  111 |  1111 | 11111                |
|    2 |    0 |   22 |  222 |  2222 | 22222                |
|    3 |    1 |   33 |  333 |  3333 | 33333                |
|    4 | NULL | NULL | NULL |  NULL | NULL                 |
|    5 |    1 |  -11 | -111 | -1111 | -11111               |
|    6 | NULL | NULL | NULL |  NULL | 36893488147419103232 |
+------+------+------+------+-------+----------------------+
```

Пример 1: Агрегация значений в столбце `c1` в один bitmap.

```
mysql> select bitmap_to_string(bitmap_agg(c1)) from t1_test;
+----------------------------------+
| bitmap_to_string(bitmap_agg(c1)) |
+----------------------------------+
| 1,2,3,4,5,6                      |
+----------------------------------+
```

Пример 2: Агрегация значений в столбце `c2` в один bitmap (значения NULL игнорируются).

```
mysql> SELECT BITMAP_TO_STRING(BITMAP_AGG(c2)) FROM t1_test;
+----------------------------------+
| bitmap_to_string(bitmap_agg(c2)) |
+----------------------------------+
| 0,1                              |
+----------------------------------+
```

Пример 3: Агрегация значений в столбце `c6` в один bitmap (последние два значения, выходящие за пределы допустимого диапазона, игнорируются).

```
mysql> select bitmap_to_string(bitmap_agg(c6)) from t1_test;
+----------------------------------+
| bitmap_to_string(bitmap_agg(c6)) |
+----------------------------------+
| 11111,22222,33333                |
+----------------------------------+
```

Ключевые слова

BITMAP_AGG, BITMAP

Bitmap_and

Вычисляет пересечение двух входных bitmap и возвращает новый bitmap.

Синтаксис

```
BITMAP BITMAP_AND(BITMAP lhs, BITMAP rhs)
```

Примеры

```
MySQL > select bitmap_count(bitmap_and(to_bitmap(1), to_bitmap(2))) cnt;
+------+
| cnt  |
+------+
|    0 |
+------+

MySQL > select bitmap_count(bitmap_and(to_bitmap(1), to_bitmap(1))) cnt;
+------+
| cnt  |
+------+
|    1 |
+------+
```

Ключевые слова

BITMAP_AND,BITMAP

Bitmap_andnot

Возвращает значения bitmap, которые существуют в `lhs`, но не существуют в `rhs`, и возвращает новый bitmap.

Синтаксис

```
bitmap_andnot(BITMAP lhs, BITMAP rhs)
```

Примеры

```
mysql> select bitmap_to_string(bitmap_andnot(bitmap_from_string('1, 3'),
bitmap_from_string('2'))) cnt;
+------+
|cnt   |
+------+
|1,3   |
+------+

mysql> select bitmap_to_string(bitmap_andnot(bitmap_from_string('1,3,5'),
bitmap_from_string('1'))) cnt;
+------+
|cnt   |
+------+
|3,5   |
+------+
```

Ключевые слова

BITMAP_ANDNOT, BITMAP

Bitmap_contains

Проверяет, содержится ли входное значение в столбце bitmap, и возвращает логическое значение.

Синтаксис

```
BOOLEAN BITMAP_CONTAINS(BITMAP bitmap, BIGINT input)
```

Примеры

```
MySQL > select bitmap_contains(to_bitmap(1),2) cnt;
+------+
| cnt  |
+------+
|    0 |
+------+

MySQL > select bitmap_contains(to_bitmap(1),1) cnt;
+------+
| cnt  |
+------+
|    1 |
+------+
```

Ключевые слова

BITMAP_CONTAINS,BITMAP

Bitmap_count

Возвращает количество установленных битов (1-битов) во входном bitmap.

Синтаксис

```
INT BITMAP_COUNT(any_bitmap)
```

Примеры

```
MySQL > select bitmap_count(bitmap_from_string("1,2,4"));
+-------------------------------------------+
| bitmap_count(bitmap_from_string('1,2,4')) |
+-------------------------------------------+
|                                         3 |
+-------------------------------------------+

MySQL > select bitmap_count(NULL);
+--------------------+
| bitmap_count(NULL) |
+--------------------+
|                  0 |
+--------------------+
```

Ключевые слова

BITMAP,BITMAP_COUNT

Bitmap_empty

Возвращает пустой bitmap. Используется в основном для заполнения значений по умолчанию при insert или stream load. Например:

```
cat data | curl --location-trusted -u user:passwd -T - \
    -H "columns: dt,page,v1,v2=bitmap_empty()" \
    http://host:8410/api/test/testDb/_stream_load
```

Синтаксис

```
BITMAP BITMAP_EMPTY()
```

Примеры

```
MySQL > select bitmap_count(bitmap_empty());
+------------------------------+
| bitmap_count(bitmap_empty()) |
+------------------------------+
|                            0 |
+------------------------------+
```

Ключевые слова

BITMAP_EMPTY,BITMAP

Bitmap_from_binary

Преобразует двоичную строку определенного формата в bitmap.

Эта функция может использоваться для загрузки данных bitmap в Selena.

Эта функция поддерживается начиная с версии 3.0.

Синтаксис

```
BITMAP bitmap_from_binary(VARBINARY str)
```

Параметры

`str`: поддерживаемый тип данных — VARBINARY.

Возвращаемое значение

Возвращает значение типа BITMAP.

Примеры

Пример 1: Использование этой функции с другими функциями Bitmap.

```
mysql> select
bitmap_to_string(bitmap_from_binary(bitmap_to_binary(bitmap_from_string("0,1,
+-----------------------------------------------------------------------------
--+
|
bitmap_to_string(bitmap_from_binary(bitmap_to_binary(bitmap_from_string('0,1,
|
+-----------------------------------------------------------------------------
--+
| 0,1,2,3
|
+-----------------------------------------------------------------------------
--+
1 row in set (0.01 sec)
```

Bitmap_from_string

Преобразует строку в BITMAP. Строка состоит из набора чисел UINT64, разделенных запятыми. Например, строка "0, 1, 2" будет преобразована в Bitmap, в котором установлены биты 0, 1 и 2. Если входное поле недопустимо, возвращается NULL.

Эта функция удаляет дубликаты из входной строки во время преобразования. Она должна использоваться совместно с другими функциями, такими как bitmap_to_string, для возврата результатов на терминал.

Синтаксис

```
BITMAP BITMAP_FROM_STRING(VARCHAR input)
```

Возвращаемое значение

Возвращает значение типа BITMAP. Если входная строка недопустима, возвращается NULL. Если входная строка пуста, возвращается пустое значение.

Примеры

```
-- Входные данные пусты, возвращается пустое значение.
MySQL > select bitmap_to_string(bitmap_empty());
+-----------------------------------+
| bitmap_to_string(bitmap_empty()) |
+-----------------------------------+
|                                   |
+-----------------------------------+

-- Возвращается `0,1,2`.
MySQL > select bitmap_to_string(bitmap_from_string("0, 1, 2"));
+--------------------------------------------------+
| bitmap_to_string(bitmap_from_string('0, 1, 2')) |
+--------------------------------------------------+
| 0,1,2                                            |
+--------------------------------------------------+

-- `-1` является недопустимым значением, возвращается NULL.

MySQL > select bitmap_to_string(bitmap_from_string("-1, 0, 1, 2"));
+-----------------------------------+
| bitmap_from_string('-1, 0, 1, 2') |
+-----------------------------------+
| NULL                              |
+-----------------------------------+

-- 2^64 является недопустимым значением, возвращается NULL.
MySQL > select bitmap_to_string(bitmap_from_string("0,
18446744073709551616"));
+-------------------------------------------------------------------+
| bitmap_to_string(bitmap_from_string('0, 18446744073709551616')) |
+-------------------------------------------------------------------+
| NULL                                                              |
+-------------------------------------------------------------------+

-- Входная строка дедуплицируется.

MySQL > select bitmap_to_string(bitmap_from_string("0, 1, 1"));
+--------------------------------------------------+
| bitmap_to_string(bitmap_from_string('0, 1, 1')) |
+--------------------------------------------------+
| 0,1                                              |
+--------------------------------------------------+
```

Ключевые слова

BITMAP_FROM_STRING,BITMAP

Bitmap_has_any

Проверяет, есть ли пересекающиеся элементы между двумя столбцами Bitmap, и возвращает логическое значение.

Синтаксис

```
BOOLEAN BITMAP_HAS_ANY(BITMAP lhs, BITMAP rhs)
```

Примеры

```
MySQL > select bitmap_has_any(to_bitmap(1),to_bitmap(2)) cnt;
+------+
| cnt  |
+------+
|    0 |
+------+

MySQL > select bitmap_has_any(to_bitmap(1),to_bitmap(1)) cnt;
+------+
| cnt  |
+------+
|    1 |
+------+
```

Ключевые слова

BITMAP_HAS_ANY,BITMAP

Bitmap_hash

Вычисляет 32-битное хеш-значение для входных данных любого типа и возвращает bitmap, содержащий это хеш-значение. В основном используется для задач stream load для импорта нецелочисленных полей в поле bitmap таблицы Selena. Например:

```
cat data | curl --location-trusted -u user:passwd -T - \
    -H "columns: dt,page,device_id, device_id=bitmap_hash(device_id)" \
    http://host:8410/api/test/testDb/_stream_load
```

Синтаксис

```
BITMAP BITMAP_HASH(expr)
```

Примеры

```
MySQL > select bitmap_count(bitmap_hash('hello'));
+-------------------------------------+
| bitmap_count(bitmap_hash('hello')) |
+-------------------------------------+
|                                   1 |
+-------------------------------------+

select bitmap_to_string(bitmap_hash('hello'));
+----------------------------------------+
| bitmap_to_string(bitmap_hash('hello')) |
+----------------------------------------+
| 1321743225                             |
+----------------------------------------+
```

Ключевые слова

BITMAP_HASH,BITMAP

Bitmap_hash64

Вычисляет 64-битное хеш-значение для входных данных любого типа и возвращает bitmap, содержащий это хеш-значение. В основном используется для загрузки нецелочисленных данных в поле BITMAP таблицы Selena через Stream Load. Например:

```
cat data | curl --location-trusted -u user:passwd -T - \
    -H "columns: dt,page,device_id, device_id=bitmap_hash64(device_id)" \
    http://<fe_host>:8030/api/test_db/tbl1/_stream_load
```

Синтаксис

```
BITMAP BITMAP_HASH64(expr)
```

Параметры

`expr`: входные данные могут быть любого типа.

Возвращаемое значение

Возвращает значение типа данных BITMAP.

Примеры

```
MySQL > select bitmap_count(bitmap_hash64('hello'));
+---------------------------------------+
| bitmap_count(bitmap_hash64('hello')) |
+---------------------------------------+
|                                     1 |
+---------------------------------------+

select bitmap_to_string(bitmap_hash64('hello'));
+-------------------------------------------+
| bitmap_to_string(bitmap_hash64('hello')) |
+-------------------------------------------+
| 10760762337991515389                      |
+-------------------------------------------+
```

Ключевые слова

BITMAP_HASH64,BITMAP

Bitmap_intersect

Агрегатная функция, используемая для вычисления пересечения bitmap после группировки. Типичные сценарии использования: расчёт коэффициента удержания пользователей.

Синтаксис

```
BITMAP BITMAP_INTERSECT(BITMAP value)
```

Принимает набор значений bitmap, находит пересечение этого набора значений bitmap и возвращает результат.

Пример

Структура таблицы

```
KeysType: AGG_KEY
Columns: tag varchar, date datetime, user_id bitmap bitmap_union
```

```
-- Вычисление удержания пользователей по различным тегам за сегодня и
вчера.
select tag, bitmap_intersect(user_id)
from (
    select tag, date, bitmap_union(user_id) user_id
    from table
    where date in ('2020-05-18', '2020-05-19')
    group by tag, date) a
group by tag;
```

Использование вместе с функцией bitmap_to_string для получения конкретных данных пересечения.

```
-- Найти пользователей, удержанных по различным тегам за сегодня и вчера.
select tag, bitmap_to_string(bitmap_intersect(user_id))
from (
    select tag, date, bitmap_union(user_id) user_id
    from table where date in ('2020-05-18', '2020-05-19')
    group by tag, date) a
group by tag;
```

Ключевые слова

BITMAP_INTERSECT, BITMAP

Bitmap_max

Получает максимальное значение из bitmap. Если bitmap равен `NULL`, функция возвращает `NULL`. Если bitmap пустой, функция возвращает `NULL` по умолчанию.

Синтаксис

```
bitmap_max(bitmap)
```

Параметры

`bitmap`: bitmap, максимальное значение которого вы хотите получить. Поддерживается только тип данных BITMAP. Вы можете указать bitmap, созданный с помощью таких функций, как bitmap_from_string.

Возвращаемое значение

Возвращает значение типа данных LARGEINT.

Примеры

```
MySQL > select bitmap_max(bitmap_from_string("0, 1, 2, 3"));
+--------------------------------------------------+
|    bitmap_max(bitmap_from_string("0, 1, 2, 3")) |
+--------------------------------------------------+
|                          3                       |
+--------------------------------------------------+

MySQL > select bitmap_max(bitmap_from_string("-1, 0, 1, 2"));
+--------------------------------------------------+
|   bitmap_max(bitmap_from_string("-1, 0, 1, 2")) |
+--------------------------------------------------+
|                       NULL                       |
+--------------------------------------------------+

MySQL > select bitmap_max(bitmap_empty());
+-----------------------------------+
|       bitmap_max(bitmap_empty()) |
+-----------------------------------+
|                       NULL        |
+-----------------------------------+

mysql> select bitmap_max(bitmap_from_string("1, 16501189037412846863"));
+-----------------------------------------------------------------+
| bitmap_max(bitmap_from_string('1, 16501189037412846863')) |
+-----------------------------------------------------------------+
| 16501189037412846863                                            |
+-----------------------------------------------------------------+
1 row in set (0.02 sec)
```

Bitmap_min

Получает минимальное значение из bitmap. Если bitmap равен `NULL`, функция возвращает `NULL`. Если bitmap пустой, функция возвращает `NULL` по умолчанию.

Синтаксис

```
bitmap_min(bitmap)
```

Параметры

`bitmap`: bitmap, минимальное значение которого вы хотите получить. Поддерживается только тип данных BITMAP. Вы можете указать bitmap, созданный с помощью таких функций, как bitmap_from_string.

Возвращаемое значение

Возвращает значение типа данных LARGEINT.

Примеры

```
MySQL > select bitmap_min(bitmap_from_string("0, 1, 2, 3"));
+--------------------------------------------------+
|    bitmap_min(bitmap_from_string('0, 1, 2, 3')) |
+--------------------------------------------------+
|                                                0 |
+--------------------------------------------------+

MySQL > select bitmap_min(bitmap_from_string("-1, 0, 1, 2"));
+--------------------------------------------------+
|   bitmap_min(bitmap_from_string('-1, 0, 1, 2')) |
+--------------------------------------------------+
|                                             NULL |
+--------------------------------------------------+

MySQL > select bitmap_min(bitmap_empty());
+-----------------------------------+
|       bitmap_min(bitmap_empty()) |
+-----------------------------------+
|                              NULL |
+-----------------------------------+

mysql> select bitmap_min(bitmap_from_string("16501189037412846863"));
+-----------------------------------------------------------+
| bitmap_min(bitmap_from_string('16501189037412846863')) |
+-----------------------------------------------------------+
| 16501189037412846863                                      |
+-----------------------------------------------------------+
1 row in set (0.03 sec)
```

Bitmap_or

Вычисляет объединение двух входных bitmap и возвращает новый bitmap.

Синтаксис

```
BITMAP BITMAP_OR(BITMAP lhs, BITMAP rhs)
```

Примеры

```
MySQL > select bitmap_count(bitmap_or(to_bitmap(1), to_bitmap(2))) cnt;
+------+
| cnt  |
+------+
|    2 |
+------+

MySQL > select bitmap_count(bitmap_or(to_bitmap(1), to_bitmap(1))) cnt;
+------+
| cnt  |
+------+
|    1 |
+------+
```

Ключевые слова

BITMAP_OR,BITMAP

Bitmap_remove

Удаляет `input` из значения Bitmap `lhs` и возвращает результирующий набор.

Синтаксис

```
bitmap_remove(BITMAP lhs, BIGINT input)
```

Примеры

```
mysql> select bitmap_to_string(**bitmap_remove**(bitmap_from_string('1,
3'), 3)) cnt;
+------+
|cnt   |
+------+
|1     |
+------+

mysql> select bitmap_to_string(**bitmap_remove**
(bitmap_from_string('1,3,5'), 6)) cnt;
+------+
|cnt   |
+------+
|1,3,5 |
+------+
```

Ключевые слова

BITMAP_REMOVE, BITMAP

Bitmap_subset_in_range

Извлекает элементы из значения Bitmap в диапазоне от `start_range` до `end_range` (не включая конечное значение). Выходные элементы являются подмножеством значения Bitmap.

Эта функция в основном используется для таких сценариев, как постраничные запросы. Поддерживается начиная с версии v1.5.2.

Синтаксис

```
BITMAP bitmap_subset_in_range(BITMAP src, BIGINT start_range, BIGINT
end_range)
```

Параметры

- `src`: значение Bitmap, из которого нужно получить элементы.
- `start_range`: начало диапазона для извлечения элементов. Должно быть значением типа BIGINT. Если указанное начало диапазона превышает максимальную длину

значения BITMAP, возвращается NULL. См. пример 4.

- `end_range`: конец диапазона для извлечения элементов. Должно быть значением типа BIGINT. Если `end_range` равно или меньше `start range`, возвращается NULL. См. пример 3.

Возвращаемое значение

Возвращает значение типа BITMAP. NULL возвращается, если любой из входных параметров недействителен.

Примечания по использованию

Элементы подмножества включают `start_range`, но не включают `end_range`. См. пример 5.

Примеры

В следующих примерах входными данными для bitmap_subset_in_range() является результат bitmap_from_string. Например, `bitmap_from_string('1,1,3,1,5,3,5,7,7,9')` возвращает `1, 3, 5, 7, 9`. bitmap_subset_in_range() принимает это значение BITMAP в качестве входных данных.

Пример 1: Получить элементы подмножества из значения BITMAP со значениями элементов в диапазоне от 1 до 4. Значения в этом диапазоне: 1 и 3.

```
select
bitmap_to_string(bitmap_subset_in_range(bitmap_from_string('1,1,3,1,5,3,5,7,7
1, 4)) value;
+-------+
| value |
+-------+
| 1,3   |
+-------+
```

Пример 2: Получить элементы подмножества из значения BITMAP со значениями элементов в диапазоне от 1 до 100. Конечное значение превышает максимальную длину значения BITMAP, и возвращаются все соответствующие элементы.

```
select
bitmap_to_string(bitmap_subset_in_range(bitmap_from_string('1,1,3,1,5,3,5,7,7
0, 100)) value;
+-----------+
| value     |
+-----------+
| 1,3,5,7,9 |
+-----------+
```

Пример 3: NULL возвращается, потому что конец диапазона `3` меньше начала диапазона `4`.

```
select
bitmap_to_string(bitmap_subset_in_range(bitmap_from_string('1,1,3,1,5,3,5,7,7
4, 3)) value;
+-------+
| value |
+-------+
| NULL  |
+-------+
```

Пример 4: Начало диапазона 10 превышает максимальную длину (5) значения BITMAP `1,3,5,7,9`. Возвращается NULL.

```
select
bitmap_to_string(bitmap_subset_in_range(bitmap_from_string('1,1,3,1,5,3,5,7,7
10, 15)) value;
+-------+
| value |
+-------+
| NULL  |
+-------+
```

Пример 5: Возвращаемое подмножество включает начальное значение `1`, но исключает конечное значение `3`.

```
select
bitmap_to_string(bitmap_subset_in_range(bitmap_from_string('1,1,3,1,5,4,5,6,7
1, 3)) value;
+-------+
| value |
+-------+
| 1     |
+-------+
```

Ссылки

bitmap_subset_limit

Bitmap_subset_limit

Извлекает указанное количество элементов из значения BITMAP, начиная со значения элемента `start range`. Выходные элементы являются подмножеством `src`.

Эта функция в основном используется для таких сценариев, как постраничные запросы. Поддерживается начиная с версии v1.5.2.

Эта функция похожа на sub_bitmap. Разница в том, что эта функция извлекает элементы, начиная со значения элемента (`start_range`), тогда как sub_bitmap извлекает элементы, начиная со смещения.

Синтаксис

```
BITMAP bitmap_subset_limit(BITMAP src, BIGINT start_range, BIGINT limit)
```

Параметры

- `src`: значение BITMAP, из которого нужно получить элементы.
- `start_range`: начало диапазона для извлечения элементов. Должно быть значением типа BIGINT. Если указанное начало диапазона превышает максимальный элемент значения BITMAP и `limit` положительный, возвращается NULL. См. пример 4.
- `limit`: количество элементов для получения, начиная с `start_range`. Отрицательные значения limit отсчитываются справа налево. Если количество соответствующих элементов меньше значения `limit`, возвращаются все соответствующие элементы.

Возвращаемое значение

Возвращает значение типа BITMAP. NULL возвращается, если любой из входных параметров недействителен.

Примечания по использованию

- Элементы подмножества включают `start range`.
- Отрицательные значения limit отсчитываются справа налево. См. пример 3.

Примеры

В следующих примерах входными данными для bitmap_subset_limit() является результат bitmap_from_string. Например, `bitmap_from_string('1,1,3,1,5,3,5,7,7,9')` возвращает `1, 3, 5, 7, 9`. bitmap_subset_limit() принимает это значение BITMAP в качестве входных данных.

Пример 1: Получить 4 элемента из значения BITMAP со значениями элементов, начиная с 1.

```
select
bitmap_to_string(bitmap_subset_limit(bitmap_from_string('1,1,3,1,5,3,5,7,7,9'
1, 4)) value;
+---------+
|  value  |
+---------+
| 1,3,5,7 |
+---------+
```

Пример 2: Получить 100 элементов из значения BITMAP со значениями элементов, начиная с 1. Limit превышает длину значения BITMAP, и возвращаются все соответствующие элементы.

```
select
bitmap_to_string(bitmap_subset_limit(bitmap_from_string('1,1,3,1,5,3,5,7,7,9'
1, 100)) value;
+-----------+
| value     |
+-----------+
| 1,3,5,7,9 |
+-----------+
```

Пример 3: Получить -2 элемента из значения BITMAP со значениями элементов, начиная с 5 (отсчёт справа налево).

```
select
bitmap_to_string(bitmap_subset_limit(bitmap_from_string('1,1,3,1,5,3,5,7,7,9'
5, -2)) value;
+-----------+
| value     |
+-----------+
| 3,5       |
+-----------+
```

Пример 4: Начало диапазона 10 превышает максимальный элемент значения BITMAP `1,3,5,7,9`, и limit положительный. Возвращается NULL.

```
select
bitmap_to_string(bitmap_subset_in_range(bitmap_from_string('1,1,3,1,5,3,5,7,7
10, 15)) value;
+-------+
| value |
+-------+
| NULL  |
+-------+
```

Bitmap_to_array

Преобразует BITMAP в массив BIGINT.

Синтаксис

```
ARRAY<BIGINT> BITMAP_TO_ARRAY (bitmap)
```

Параметры

`bitmap`: bitmap, который вы хотите преобразовать.

Возвращаемое значение

Возвращает массив BIGINT.

Примеры

```
select bitmap_to_array(bitmap_from_string("1, 7"));
+-----------------------------------------------+
| bitmap_to_array(bitmap_from_string('1, 7'))  |
+-----------------------------------------------+
| [1,7]                                         |
+-----------------------------------------------+

select bitmap_to_array(NULL);
+------------------------+
| bitmap_to_array(NULL) |
+------------------------+
| NULL                   |
+------------------------+
```

Bitmap_to_base64

Преобразует bitmap в строку, закодированную в Base64. Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
VARCHAR bitmap_to_base64(BITMAP bitmap)
```

Параметры

`bitmap`: bitmap для преобразования. Этот параметр является обязательным. Если входное значение недействительно, возвращается ошибка.

Возвращаемое значение

Возвращает значение типа VARCHAR.

Примеры

Пример 1: Использование этой функции с другими функциями bitmap.

```
select bitmap_to_base64(bitmap_from_string("0, 1, 2, 3"));
+-----------------------------------------------------+
| bitmap_to_base64(bitmap_from_string('0, 1, 2, 3')) |
+-----------------------------------------------------+
| AjowAAABAAAAAAADABAAAAAAAAEAAgADAA==                |
+-----------------------------------------------------+
1 row in set (0.00 sec)


select bitmap_to_base64(to_bitmap(1));
+--------------------------------+
| bitmap_to_base64(to_bitmap(1)) |
+--------------------------------+
| AQEAAAA=                       |
+--------------------------------+
1 row in set (0.00 sec)


select bitmap_to_base64(bitmap_empty());
+----------------------------------+
| bitmap_to_base64(bitmap_empty()) |
+----------------------------------+
| AA==                             |
+----------------------------------+
1 row in set (0.00 sec)
```

Пример 2: Преобразование каждого значения в столбце BITMAP в строки, закодированные в Base64.

1. Создайте агрегатную таблицу `page_uv`, где `AGGREGATE KEY` - это (`page_id`, `visit_date`). Эта таблица содержит столбец BITMAP `visit_users`, значения которого будут агрегироваться.

```
CREATE TABLE `page_uv`
(`page_id` INT NOT NULL,
`visit_date` datetime NOT NULL,
`visit_users` BITMAP BITMAP_UNION NOT NULL
) ENGINE=OLAP
AGGREGATE KEY(`page_id`, `visit_date`)
DISTRIBUTED BY HASH(`page_id`)
PROPERTIES (
"replication_num" = "3",
"storage_format" = "DEFAULT"
);
```

2. Вставьте данные в эту таблицу.

```
insert into page_uv values
(1, '2020-06-23 01:30:30', to_bitmap(13)),
(1, '2020-06-23 01:30:30', to_bitmap(23)),
(1, '2020-06-23 01:30:30', to_bitmap(33)),
(1, '2020-06-23 02:30:30', to_bitmap(13)),
(2, '2020-06-23 01:30:30', to_bitmap(23));

select * from page_uv order by page_id;
+---------+---------------------+-------------+
| page_id | visit_date          | visit_users |
+---------+---------------------+-------------+
|       1 | 2020-06-23 01:30:30 | NULL        |
|       1 | 2020-06-23 02:30:30 | NULL        |
|       2 | 2020-06-23 01:30:30 | NULL        |
+---------+---------------------+-------------+
```

3. Преобразуйте каждое значение в столбце `visit_users` в строку, закодированную в Base64.

```
select page_id, bitmap_to_base64(visit_users) from page_uv;
+---------+-------------------------------------------+
| page_id | bitmap_to_base64(visit_users)             |
+---------+-------------------------------------------+
|       1 | CgMAAAANAAAAAAAAABcAAAAAAAAAIQAAAAAAAAA= |
|       1 | AQ0AAAA=                                  |
|       2 | ARcAAAA=                                  |
+---------+-------------------------------------------+
```

Bitmap_to_binary

Преобразует значения Bitmap в двоичную строку.

bitmap_to_binary в основном используется для экспорта данных Bitmap. Эффект сжатия лучше, чем у bitmap_to_base64.

Если вы планируете экспортировать данные напрямую в двоичный файл, такой как Parquet, рекомендуется использовать эту функцию.

Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
VARBINARY bitmap_to_binary(BITMAP bitmap)
```

Параметры

`bitmap`: Bitmap для преобразования, обязательный параметр. Если входное значение недействительно, возвращается ошибка.

Возвращаемое значение

Возвращает значение типа VARBINARY.

Примеры

Пример 1: Использование этой функции с другими функциями bitmap.

```
mysql> select hex(bitmap_to_binary(bitmap_from_string("0, 1, 2, 3")));
+-----------------------------------------------------------+
| hex(bitmap_to_binary(bitmap_from_string('0, 1, 2, 3'))) |
+-----------------------------------------------------------+
| 023A3000000100000000000300100000000000010002000300      |
+-----------------------------------------------------------+
1 row in set (0.01 sec)

mysql> select hex(bitmap_to_binary(to_bitmap(1)));
+-------------------------------------+
| hex(bitmap_to_binary(to_bitmap(1))) |
+-------------------------------------+
| 0101000000                          |
+-------------------------------------+
1 row in set (0.01 sec)

mysql> select hex(bitmap_to_binary(bitmap_empty()));
+----------------------------------------+
| hex(bitmap_to_binary(bitmap_empty())) |
+----------------------------------------+
| 00                                     |
+----------------------------------------+
1 row in set (0.01 sec)
```

Пример 2: Преобразование каждого значения в столбце BITMAP в двоичную строку.

1. Создайте агрегатную таблицу `page_uv`, где `AGGREGATE KEY` - это (`page_id`, `visit_date`). Эта таблица содержит столбец BITMAP `visit_users`, значения которого будут агрегироваться.

```
CREATE TABLE `page_uv`
(`page_id` INT NOT NULL,
`visit_date` datetime NOT NULL,
`visit_users` BITMAP BITMAP_UNION NOT NULL
) ENGINE=OLAP
AGGREGATE KEY(`page_id`, `visit_date`)
DISTRIBUTED BY HASH(`page_id`)
PROPERTIES (
"replication_num" = "3",
"storage_format" = "DEFAULT"
);
```

2. Вставьте данные в эту таблицу.

```
insert into page_uv values
(1, '2020-06-23 01:30:30', to_bitmap(13)),
(1, '2020-06-23 01:30:30', to_bitmap(23)),
(1, '2020-06-23 01:30:30', to_bitmap(33)),
(1, '2020-06-23 02:30:30', to_bitmap(13)),
(2, '2020-06-23 01:30:30', to_bitmap(23));

select * from page_uv order by page_id;
+---------+---------------------+-------------+
| page_id | visit_date          | visit_users |
+---------+---------------------+-------------+
|       1 | 2020-06-23 01:30:30 | NULL        |
|       1 | 2020-06-23 02:30:30 | NULL        |
|       2 | 2020-06-23 01:30:30 | NULL        |
+---------+---------------------+-------------+
```

3. Преобразуйте каждое значение в столбце `visit_users` в строку, закодированную в двоичном формате.

```
   mysql> select page_id, hex(bitmap_to_binary(visit_users)) from
page_uv;
   +---------+-------------------------------------------------------------
---+
   | page_id | hex(bitmap_to_binary(visit_users))
|
   +---------+-------------------------------------------------------------
---+
   |       1 |
0A030000000D000000000000001700000000000000210000000000000 |
   |       1 | 010D000000
|
   |       2 | 0117000000
|
   +---------+-------------------------------------------------------------
---+
   3 rows in set (0.01 sec)
```

Bitmap_to_string

Преобразует входной bitmap в строку, разделённую запятыми (,). Эта строка содержит все биты в bitmap. Если входное значение равно null, возвращается null.

Синтаксис

```
VARCHAR BITMAP_TO_STRING(BITMAP input)
```

Параметры

`input`: bitmap, который вы хотите преобразовать.

Возвращаемое значение

Возвращает значение типа VARCHAR.

Примеры

Пример 1: Входное значение равно null, и возвращается null.

```
MySQL > select bitmap_to_string(null);
+------------------------+
| bitmap_to_string(NULL) |
+------------------------+
| NULL                   |
+------------------------+
```

Пример 2: Входное значение пустое, и возвращается пустая строка.

```
MySQL > select bitmap_to_string(bitmap_empty());
+----------------------------------+
| bitmap_to_string(bitmap_empty()) |
+----------------------------------+
|                                  |
+----------------------------------+
```

Пример 3: Преобразование bitmap, содержащего один бит, в строку.

```
MySQL > select bitmap_to_string(to_bitmap(1));
+--------------------------------+
| bitmap_to_string(to_bitmap(1)) |
+--------------------------------+
| 1                              |
+--------------------------------+
```

Пример 4: Преобразование bitmap, содержащего два бита, в строку.

```
MySQL > select bitmap_to_string(bitmap_or(to_bitmap(1), to_bitmap(2)));
+------------------------------------------------------------+
| bitmap_to_string(bitmap_or(to_bitmap(1), to_bitmap(2))) |
+------------------------------------------------------------+
| 1,2                                                        |
+------------------------------------------------------------+
```

Bitmap_union

Вычисляет объединение bitmap для набора значений после группировки. Типичные сценарии использования включают расчёт PV и UV.

Синтаксис

```
BITMAP BITMAP_UNION(BITMAP value)
```

Примеры

```
select page_id, bitmap_union(user_id)
from table
group by page_id;
```

Используйте эту функцию вместе с bitmap_count() для получения UV веб-страницы.

```
select page_id, bitmap_count(bitmap_union(user_id))
from table
group by page_id;
```

Если `user_id` является целым числом, приведённый выше оператор запроса эквивалентен следующему:

```
select page_id, count(distinct user_id)
from table
group by page_id;
```

Ключевые слова

BITMAP_UNION, BITMAP

Bitmap_union_count

Возвращает объединение набора значений bitmap и возвращает кардинальность объединения. Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
BIGINT bitmap_union_count(BITMAP value)
```

Параметры

`value`: набор значений bitmap. Поддерживаемый тип данных - BITMAP.

Возвращаемое значение

Возвращает значение типа BIGINT.

Примеры

Расчёт уникальных просмотров (UV) веб-страницы. Если `user_id` имеет тип INT, последние два запроса эквивалентны.

```
mysql> select * from test
+---------+---------+
| page_id | user_id |
+---------+---------+
|       1 |       1 |
|       1 |       2 |
|       2 |       1 |
+---------+---------+

mysql> select page_id,count(distinct user_id) from test group by page_id;
+---------+-------------------------+
| page_id | count(DISTINCT user_id) |
+---------+-------------------------+
|       1 |                       2 |
|       2 |                       1 |
+---------+-------------------------+

mysql> select page_id,bitmap_union_count(to_bitmap(user_id)) from test
group by page_id;
+---------+----------------------------------------+
| page_id | bitmap_union_count(to_bitmap(user_id)) |
+---------+----------------------------------------+
|       1 |                                      2 |
|       2 |                                      1 |
+---------+----------------------------------------+
```

Bitmap_union_int

Подсчитывает количество различных значений в столбцах типа TINYINT, SMALLINT и INT, возвращает сумму, эквивалентную COUNT (DISTINCT expr).

Синтаксис

```
BIGINT bitmap_union_int(expr)
```

Параметры

`expr`: выражение столбца. Поддерживаемые типы столбцов: TINYINT, SMALLINT и INT.

Возвращаемое значение

Возвращает значение типа BIGINT.

Примеры

```
mysql> select bitmap_union_int(k1) from tbl1;
+------------------------+
| bitmap_union_int(`k1`) |
+------------------------+
|                      2 |
+------------------------+
```

Bitmap_xor

Вычисляет множество, состоящее из элементов, уникальных для `lhs` и `rhs`. Логически эквивалентно `bitmap_andnot(bitmap_or(lhs, rhs), bitmap_and(lhs, rhs))` (дополнительное множество).

Синтаксис

```
bitmap_xor(BITMAP lhs, BITMAP rhs)
```

Примеры

```
mysql> select bitmap_to_string(bitmap_xor(bitmap_from_string('1, 3'),
bitmap_from_string('2'))) cnt;
+------+
|cnt   |
+------+
|1,2,3 |
+------+
```

Ключевые слова

BITMAP_XOR, BITMAP

Intersect_count

Функция для нахождения размера пересечения (количества одинаковых элементов) двух значений Bitmap без требования ортогональности распределения данных. Первый параметр - столбец Bitmap, второй параметр - столбец измерения, используемый для фильтрации, а третий параметр - параметр переменной длины, означающий, что берутся разные значения столбца измерения фильтрации.

Если пересечения нет, возвращается 0.

Синтаксис

```
BIGINT INTERSECT_COUNT(bitmap_column, column_to_filter, filter_values)
```

Параметры

- `bitmap_column`: столбец bitmap для вычисления.
- `column_to_filter`: имя столбца, по которому выполняется пересечение.
- `filter_values`: различные значения столбцов измерений фильтрации.

Возвращаемое значение

Возвращает значение типа BIGINT.

Пример

Создайте агрегатную таблицу со столбцом Bitmap `user_id` и вставьте данные.

```
CREATE TABLE `tbl` (
  `dt` date NULL COMMENT "",
  `user_id` bitmap BITMAP_UNION NULL COMMENT ""

) ENGINE=OLAP
AGGREGATE KEY(`dt`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`dt`);

INSERT INTO `tbl` VALUES
('2020-10-01', to_bitmap(2)),
('2020-10-01', to_bitmap(3)),
('2020-10-01', to_bitmap(5)),
('2020-10-02', to_bitmap(2)),
('2020-10-02', to_bitmap(3)),
('2020-10-02', to_bitmap(4)),
('2020-10-03', to_bitmap(1)),
('2020-10-03', to_bitmap(5)),
('2020-10-03', to_bitmap(6));

SELECT dt, bitmap_to_string(user_id) from tbl;
+------------+---------------------------+
| dt         | bitmap_to_string(user_id) |
+------------+---------------------------+
| 2020-10-02 | 2,3,4                     |
| 2020-10-03 | 1,5,6                     |
| 2020-10-01 | 2,3,5                     |
+------------+---------------------------+
```

Вычислите количество одинаковых `user_id`, соответствующих `'2020-10-01'` и `'2020-10-02'`. Найдены два одинаковых элемента (2 и 3), и возвращается число `2`.

```
mysql>  select intersect_count(user_id, dt, '2020-10-01', '2020-10-02')
        from tbl
        where dt in ('2020-10-01', '2020-10-02');
+-------------------------------------------------------------+
| intersect_count(user_id, dt, '2020-10-01', '2020-10-02') |
+-------------------------------------------------------------+
|                                                           2 |
+-------------------------------------------------------------+
```

Вычислите количество одинаковых `user_id`, соответствующих `'2020-10-01'` и `'2020-10-03'`. Найден один элемент `5`, и возвращается число `1`.

```
mysql>  select intersect_count(user_id, dt, '2020-10-01', '2020-10-03')
        from tbl
        where dt in ('2020-10-01', '2020-10-03');
+-----------------------------------------------------------+
| intersect_count(user_id, dt, '2020-10-01', '2020-10-03') |
+-----------------------------------------------------------+
|                                                         1 |
+-----------------------------------------------------------+
```

Sub_bitmap

Извлекает `len` элементов из BITMAP-значения `src`, начиная с позиции, указанной в `offset`. Выходные элементы являются подмножеством `src`.

Эта функция в основном используется для сценариев, таких как постраничные запросы. Поддерживается с версии v1.5.2.

Эта функция похожа на bitmap_subset_limit. Отличие в том, что данная функция извлекает элементы, начиная со смещения, тогда как bitmap_subset_limit извлекает элементы, начиная со значения элемента (`start_range`).

Синтаксис

```
BITMAP sub_bitmap(BITMAP src, BIGINT offset, BIGINT len)
```

Параметры

- `src`: BITMAP-значение, из которого вы хотите получить элементы.
- `offset`: начальная позиция. Должно быть значением типа BIGINT. Обратите внимание на следующие особенности при использовании `offset`:
  - Смещение начинается с 0.
  - Отрицательные смещения отсчитываются справа налево. См. примеры 3 и 4.
  - Если начальная позиция, указанная в `offset`, превышает фактическую длину BITMAP-значения, возвращается NULL. См. пример 6.
- `len`: количество элементов для получения. Должно быть значением типа BIGINT больше или равным 1. Если количество подходящих элементов меньше значения

`len`, возвращаются все подходящие элементы. См. примеры 2, 3 и 7.

Возвращаемое значение

Возвращает значение типа BITMAP. Возвращается NULL, если любой из входных параметров недействителен.

Примеры

В следующих примерах входными данными для sub_bitmap() является результат функции bitmap_from_string. Например, `bitmap_from_string('1,1,3,1,5,3,5,7,7,9')` возвращает `1, 3, 5, 7, 9`. Функция sub_bitmap() принимает это BITMAP-значение как входные данные.

Пример 1: Получить два элемента из BITMAP-значения со смещением 0.

```
select
bitmap_to_string(sub_bitmap(bitmap_from_string('1,1,3,1,5,3,5,7,7,9'), 0,
2)) value;
+-------+
| value |
+-------+
| 1,3   |
+-------+
```

Пример 2: Получить 100 элементов из BITMAP-значения со смещением 0. 100 превышает длину BITMAP-значения, и возвращаются все подходящие элементы.

```
select
bitmap_to_string(sub_bitmap(bitmap_from_string('1,1,3,1,5,3,5,7,7,9'), 0,
100)) value;
+-----------+
| value     |
+-----------+
| 1,3,5,7,9 |
+-----------+
```

Пример 3: Получить 100 элементов из BITMAP-значения со смещением -3. 100 превышает длину BITMAP-значения, и возвращаются все подходящие элементы.

```
select
bitmap_to_string(sub_bitmap(bitmap_from_string('1,1,3,1,5,3,5,7,7,9'), -3,
100)) value;
+-------+
| value |
+-------+
| 5,7,9 |
+-------+
```

Пример 4: Получить два элемента из BITMAP-значения со смещением -3.

```
select
bitmap_to_string(sub_bitmap(bitmap_from_string('1,1,3,1,5,3,5,7,7,9'), -3,
2)) value;
+-------+
| value |
+-------+
| 5,7   |
+-------+
```

Пример 5: Возвращается NULL, поскольку `-10` является недопустимым значением для `len`.

```
select
bitmap_to_string(sub_bitmap(bitmap_from_string('1,1,3,1,5,3,5,7,7,9'), 0,
-10)) value;
+-------+
| value |
+-------+
| NULL  |
+-------+
```

Пример 6: Начальная позиция, указанная смещением 5, превышает длину BITMAP-значения `1,3,5,7,9`. Возвращается NULL.

```
select
bitmap_to_string(sub_bitmap(bitmap_from_string('1,1,3,1,5,3,5,7,7,9'), 5,
1)) value;
+-------+
| value |
+-------+
| NULL  |
+-------+
```

Пример 7: `len` установлен в 5, но только два элемента соответствуют условию. Все эти два элемента возвращаются.

```
select
bitmap_to_string(sub_bitmap(bitmap_from_string('1,1,3,1,5,3,5,7,7,9'), -2,
5)) value;
+-------+
| value |
+-------+
| 7,9   |
+-------+
```

Subdivide_bitmap

Разбивает большой bitmap на несколько под-bitmap.

Эта функция в основном используется для экспорта bitmap-ов. Слишком большие bitmap-ы превышают максимальный размер пакета, разрешённый протоколом MySQL.

Эта функция поддерживается с версии v1.5.2.

Синтаксис

```
BITMAP subdivide_bitmap(bitmap, length)
```

Параметры

`bitmap`: Bitmap, который необходимо разбить, обязательный параметр. `length`: Максимальная длина каждого под-bitmap, обязательный параметр. Bitmap-ы, превышающие это значение, будут разбиты на несколько маленьких bitmap-ов.

Возвращаемое значение

Возвращает несколько под-bitmap, размер которых не превышает `length`.

Примеры

Предположим, есть таблица `t1`, в которой столбец `c2` является столбцом типа BITMAP.

```
-- Используйте bitmap_to_string() для преобразования значений в `c2` в
строку.
mysql> select c1, bitmap_to_string(c2) from t1;
+------+----------------------+
| c1   | bitmap_to_string(c2) |
+------+----------------------+
|    1 | 1,2,3,4,5,6,7,8,9,10 |
+------+----------------------+

-- Разбить `c2` на маленькие bitmap-ы, максимальная длина которых равна 3.

mysql> select c1, bitmap_to_string(subdivide_bitmap) from t1,
subdivide_bitmap(c2, 3);
+------+-----------------------------------+
| c1   | bitmap_to_string(subdivide_bitmap) |
+------+-----------------------------------+
|    1 | 1,2,3                             |
|    1 | 4,5,6                             |
|    1 | 7,8,9                             |
|    1 | 10                                |
+------+-----------------------------------+
```

To_bitmap

Входное значение — беззнаковое bigint со значением в диапазоне от 0 до 18446744073709551615, а выходное — bitmap, содержащий этот элемент. Эта функция в основном используется для задачи stream load для импорта целочисленных полей в поле bitmap таблицы Selena. Например:

```
cat data | curl --location-trusted -u user:passwd -T - \
    -H "columns: dt,page,user_id, user_id=to_bitmap(user_id)" \
    http://host:8410/api/test/testDb/_stream_load
```

Синтаксис

```
BITMAP TO_BITMAP(expr)
```

Примеры

```
MySQL > select bitmap_count(to_bitmap(10));
+-----------------------------+
| bitmap_count(to_bitmap(10)) |
+-----------------------------+
|                           1 |
+-----------------------------+

select bitmap_to_string(to_bitmap(10));
+---------------------------------+
| bitmap_to_string(to_bitmap(10)) |
+---------------------------------+
| 10                              |
+---------------------------------+

select bitmap_to_string(to_bitmap(-5));
+---------------------------------+
| bitmap_to_string(to_bitmap(-5)) |
+---------------------------------+
| NULL                            |
+---------------------------------+

select bitmap_to_string(to_bitmap(null));
+-----------------------------------+
| bitmap_to_string(to_bitmap(NULL)) |
+-----------------------------------+
| NULL                              |
+-----------------------------------+
```

Keyword

TO_BITMAP,BITMAP

Unnest_bitmap

unnest_bitmap — это табличная функция, которая принимает bitmap и преобразует элементы в этом bitmap в несколько строк таблицы.

Lateral Join Selena можно использовать в сочетании с функцией unnest_bitmap для реализации распространённой логики преобразования столбца в строки.

Эту функцию можно использовать для замены `unnest(bitmap_to_array(bitmap))`. Она имеет лучшую производительность, использует меньше ресурсов памяти и не ограничена длиной массива.

Эта функция поддерживается с версии v1.5.2.

Синтаксис

```
unnest_bitmap(bitmap)
```

Параметры

`bitmap`: bitmap, который вы хотите преобразовать.

Возвращаемое значение

Возвращает несколько строк, преобразованных из bitmap. Тип возвращаемого значения — BIGINT.

Примеры

`c2` — это столбец bitmap в таблице `t1`.

```
-- Используйте функцию bitmap_to_string для преобразования значений в
столбце c2 в строку.
mysql> select c1, bitmap_to_string(c2) from t1;
+------+----------------------+
| c1   | bitmap_to_string(c2) |
+------+----------------------+
|    1 | 1,2,3,4,5,6,7,8,9,10 |
+------+----------------------+

-- Используйте функцию unnest_bitmap для разворачивания столбца bitmap в
несколько строк.
mysql> select c1, unnest_bitmap from t1, unnest_bitmap(c2);
+------+---------------+
| c1   | unnest_bitmap |
+------+---------------+
|    1 |             1 |
|    1 |             2 |
|    1 |             3 |
|    1 |             4 |
|    1 |             5 |
|    1 |             6 |
|    1 |             7 |
|    1 |             8 |
|    1 |             9 |
|    1 |            10 |
+------+---------------+
```

2.12 Condition

Case

CASE — это условное выражение. Оно возвращает результат из предложения THEN, если условие в предложении WHEN истинно. Если ни одно из условий не является истинным, возвращается результат из необязательного предложения ELSE. Если ELSE отсутствует, возвращается NULL.

Синтаксис

Выражение CASE имеет две формы:

- Простой CASE

```
CASE expression
    WHEN expression1 THEN result1
    [WHEN expression2 THEN result2]
    ...
    [WHEN expressionN THEN resultN]
    [ELSE result]
END
```

Для этого синтаксиса `expression` сравнивается с каждым выражением в предложении WHEN. Если найдено равное выражение, возвращается результат из предложения THEN. Если равное выражение не найдено, возвращается результат из предложения ELSE, если ELSE присутствует.

- Поисковый CASE

```
CASE WHEN condition1 THEN result1
    [WHEN condition2 THEN result2]
    ...
    [WHEN conditionN THEN resultN]
    [ELSE result]
END
```

Для этого синтаксиса каждое условие в предложении WHEN вычисляется до тех пор, пока одно из них не станет истинным, и возвращается соответствующий результат из предложения THEN. Если ни одно условие не является истинным, возвращается результат из предложения ELSE, если ELSE присутствует.

Первый CASE эквивалентен второму следующим образом:

```
CASE WHEN expression = expression1 THEN result1
    [WHEN expression = expression2 THEN result2]
    ...
    [WHEN expression = expressionN THEN resultN]
    [ELSE result]
END
```

Параметры

- `expressionN`: выражение для сравнения. Несколько выражений должны быть совместимы по типам данных.
- `conditionN`: условие, которое может быть вычислено в значение типа BOOLEAN.
- `resultN` должны быть совместимы по типам данных.

Возвращаемое значение

Возвращаемое значение имеет общий тип всех типов в предложении THEN.

Примеры

Предположим, таблица `test_case` содержит следующие данные:

```
CREATE TABLE test_case(
    name          STRING,
    gender         INT,
    ) DISTRIBUTED BY HASH(name);

INSERT INTO test_case VALUES
    ("Andy",1),
    ("Jules",0),
    ("Angel",-1),
    ("Sam",null);

SELECT * FROM test_case;
+-------+--------+
| name  | gender |
+-------+--------+
| Angel |     -1 |
| Andy  |      1 |
| Sam   |   NULL |
| Jules |      0 |
+-------+--------+-------+
```

Использование простого CASE

- ELSE указан, и результат из ELSE возвращается, если не найдено равное выражение.

```
mysql> select gender, case gender
                     when 1 then 'male'
                     when 0 then 'female'
                     else 'error'
                end gender_str
from test_case;
+--------+------------+
| gender | gender_str |
+--------+------------+
|   NULL | error      |
|      0 | female     |
|      1 | male       |
|     -1 | error      |
+--------+------------+
```

- ELSE не указан, и NULL возвращается, если ни одно условие не является истинным.

```
select gender, case gender
                     when 1 then 'male'
                     when 0 then 'female'
                end gender_str
from test_case;
+--------+------------+
| gender | gender_str |
+--------+------------+
|      1 | male       |
|     -1 | NULL       |
|   NULL | NULL       |
|      0 | female     |
+--------+------------+
```

Использование поискового CASE без указания ELSE

```
mysql> select gender, case when gender = 1 then 'male'
                           when gender = 0 then 'female'
                      end gender_str
from test_case;
+--------+------------+
| gender | gender_str |
+--------+------------+
|   NULL | NULL       |
|     -1 | NULL       |
|      1 | male       |
|      0 | female     |
+--------+------------+
```

Keywords

case when, case, case_when, case...when

Coalesce

Возвращает первое не-NULL выражение среди входных параметров. Возвращает NULL, если не-NULL выражения не найдены.

Синтаксис

```
coalesce(expr1,...);
```

Параметры

`expr1`: входные выражения, которые должны вычисляться в совместимые типы данных.

Возвращаемое значение

Возвращаемое значение имеет тот же тип, что и `expr1`.

Примеры

```
mysql> select coalesce(3,NULL,1,1);
+-------------------------+
| coalesce(3, NULL, 1, 1) |
+-------------------------+
|                       3 |
+-------------------------+
1 row in set (0.00 sec)
```

If

Если `expr1` вычисляется как TRUE, возвращает `expr2`. В противном случае возвращает `expr3`.

Синтаксис

```
if(expr1,expr2,expr3);
```

Параметры

`expr1`: условие. Должно быть значением типа BOOLEAN.

`expr2` и `expr3` должны быть совместимы по типу данных.

Возвращаемое значение

Возвращаемое значение имеет тот же тип, что и `expr2`.

Примеры

```
mysql> select if(true,1,2);
+----------------+
| if(TRUE, 1, 2) |
+----------------+
|              1 |
+----------------+

mysql> select if(false,2.14,2);
+--------------------+
| if(FALSE, 2.14, 2) |
+--------------------+
|               2.00 |
+--------------------+
```

Ifnull

Если `expr1` равно NULL, возвращает expr2. Если `expr1` не равно NULL, возвращает `expr1`.

Синтаксис

```
ifnull(expr1,expr2);
```

Параметры

`expr1` и `expr2` должны быть совместимы по типу данных.

Возвращаемое значение

Возвращаемое значение имеет тот же тип, что и `expr1`.

Примеры

```
mysql> select ifnull(2,4);
+--------------+
| ifnull(2, 4) |
+--------------+
|            2 |
+--------------+

mysql> select ifnull(NULL,2);
+-----------------+
| ifnull(NULL, 2) |
+-----------------+
|               2 |
+-----------------+
1 row in set (0.01 sec)
```

Nullif

Возвращает NULL, если `expr1` равно `expr2`. В противном случае возвращает `expr1`.

Синтаксис

```
nullif(expr1,expr2);
```

Параметры

`expr1` и `expr2` должны быть совместимы по типу данных.

Возвращаемое значение

Возвращаемое значение имеет тот же тип, что и `expr1`.

Примеры

```
mysql> select nullif(1,2);
+--------------+
| nullif(1, 2) |
+--------------+
|            1 |
+--------------+

mysql> select nullif(1,1);
+--------------+
| nullif(1, 1) |
+--------------+
|         NULL |
+--------------+
```

2.13 Cryptographic

Aes_decrypt

Использует алгоритм AES_128_ECB для расшифровки строки и возвращает бинарную строку.

AES — это сокращение от advanced encryption standard (расширенный стандарт шифрования), а ECB — это сокращение от electronic code book (электронная кодовая книга). Ключ, используемый для шифрования строки, представляет собой 128-битную строку.

Синтаксис

```
aes_decrypt(str,key_str);
```

Параметры

`str`: строка для расшифровки. Должна быть типа VARCHAR.

`key_str`: ключ, использованный для шифрования `str`. Должен быть типа VARCHAR.

Возвращаемое значение

Возвращает значение типа VARCHAR. Если входные данные недействительны, возвращается NULL.

Примеры

Декодируйте строку Base64 и используйте эту функцию для расшифровки декодированной строки в исходную строку.

```
mysql> select AES_DECRYPT(from_base64('uv/Lhzm74syo8JlfWarwKA==
'),'F3229A0B371ED2D9441B830D21A390C3');
+-----------------------------------------------------------------------
------------------+
| aes_decrypt(from_base64('uv/Lhzm74syo8JlfWarwKA==  '),
'F3229A0B371ED2D9441B830D21A390C3') |
+-----------------------------------------------------------------------
------------------+
| selena
|
+-----------------------------------------------------------------------
------------------+
1 row in set (0.00 sec)
```

Aes_encrypt

Использует алгоритм AES_128_ECB для шифрования строки и возвращает бинарную строку.

AES — это сокращение от advanced encryption standard (расширенный стандарт шифрования), а ECB — это сокращение от electronic code book (электронная кодовая книга). Ключ, используемый для шифрования строки, представляет собой 128-битную строку.

Синтаксис

```
aes_encrypt(str,key_str);
```

Параметры

`str`: строка для шифрования. Должна быть типа VARCHAR.

`key_str`: ключ для шифрования `str`. Должен быть типа VARCHAR.

Возвращаемое значение

Возвращает значение типа VARCHAR. Если входные данные равны NULL, возвращается NULL.

Примеры

Используйте эту функцию для AES-шифрования `selena` и преобразуйте зашифрованную строку в строку в кодировке Base64.

```
mysql> select
to_base64(AES_ENCRYPT('selena','F3229A0B371ED2D9441B830D21A390C3'));
+-----------------------------------------------------------------------+
| to_base64(aes_encrypt('selena', 'F3229A0B371ED2D9441B830D21A390C3')) |
+-----------------------------------------------------------------------+
| uv/Lhzm74syo8JlfWarwKA==                                              |
+-----------------------------------------------------------------------+
1 row in set (0.01 sec)
```

Base64_decode_binary

Декодирует строку в кодировке Base64 и возвращает BINARY.

Эта функция поддерживается с версии v1.5.2.

Синтаксис

```
base64_decode_binary(str);
```

Параметры

`str`: строка для декодирования. Должна быть типа VARCHAR.

Возвращаемое значение

Возвращает значение типа VARBINARY. Если входные данные равны NULL или являются недопустимой строкой Base64, возвращается NULL. Если входные данные пустые, возвращается ошибка.

Эта функция принимает только одну строку. Несколько входных строк вызывают ошибку.

Примеры

```
mysql> select hex(base64_decode_binary(to_base64("Hello Selena")));
+-----------------------------------------------------------+
| hex(base64_decode_binary(to_base64('Hello Selena'))) |
+-----------------------------------------------------------+
| 48656C6C6F2053746172526F636B73                            |
+-----------------------------------------------------------+

mysql> select base64_decode_binary(NULL);
+-------------------------------------------------------+
| base64_decode_binary(NULL)                            |
+-------------------------------------------------------+
| NULL                                                  |
+-------------------------------------------------------+
```

Base64_decode_string

Эта функция аналогична from_base64. Декодирует строку в кодировке Base64 и является обратной функцией к to_base64.

Эта функция поддерживается с версии v1.5.2.

Синтаксис

```
base64_decode_string(str);
```

Параметры

`str`: строка для декодирования. Должна быть типа VARCHAR.

Возвращаемое значение

Возвращает значение типа VARCHAR. Если входные данные равны NULL или являются недопустимой строкой Base64, возвращается NULL. Если входные данные пустые, возвращается ошибка.

Эта функция принимает только одну строку. Несколько входных строк вызывают ошибку.

Примеры

```
mysql> select base64_decode_string(to_base64("Hello Selena"));
+-----------------------------------------------------+
| base64_decode_string(to_base64('Hello Selena')) |
+-----------------------------------------------------+
| Hello Selena                                        |
+-----------------------------------------------------+
```

From_base64

Декодирует строку, закодированную в Base64. Эта функция является обратной к to_base64.

Синтаксис

```
from_base64(str);
```

Параметры

`str`: строка для декодирования. Должна быть типа VARCHAR.

Возвращаемое значение

Возвращает значение типа VARCHAR. Если входное значение равно NULL или является недопустимой строкой Base64, возвращается NULL. Если входное значение пустое, возвращается ошибка.

Эта функция принимает только одну строку. Передача более одной входной строки приводит к ошибке.

Примеры

```
mysql> select from_base64("selena");
+--------------------------+
| from_base64('selena') |
+--------------------------+
| ²®$                       |
+--------------------------+
1 row in set (0.00 sec)

mysql> select from_base64('c3RhcnJvY2tz');
+------------------------------+
| from_base64('c3RhcnJvY2tz') |
+------------------------------+
| selena                   |
+------------------------------+
```

Md5

Использует алгоритм MD5 Message-Digest для вычисления 128-битной контрольной суммы строки. Контрольная сумма представляется в виде 32-символьной шестнадцатеричной строки.

Синтаксис

```
md5(expr)
```

Параметры

`expr`: строка для вычисления. Должна быть типа VARCHAR.

Возвращаемое значение

Возвращает контрольную сумму типа VARCHAR, представляющую собой 32-символьную шестнадцатеричную строку.

Если входное значение равно NULL, возвращается NULL.

Примеры

```
select md5('abc');
```

```
+----------------------------------+
| md5('abc')                       |
+----------------------------------+
| 900150983cd24fb0d6963f7d28e17f72 |
+----------------------------------+
1 row in set (0.01 sec)
```

```
select md5(null);
```

```
+-----------+
| md5(NULL) |
+-----------+
| NULL      |
+-----------+
1 row in set (0.00 sec)
```

Ключевые слова

MD5, ENCRYPTION

Md5sum

Вычисляет 128-битную контрольную сумму нескольких строк. Контрольная сумма представляется в виде 32-символьной шестнадцатеричной строки.

Если передана только одна строка, результат будет таким же, как у md5.

По сравнению с md5(), md5sum() принимает больше параметров и поэтому более эффективна при вычислении контрольной суммы нескольких файлов.

Синтаксис

```
md5sum(expr,...);
```

Параметры

`expr`: строка для вычисления. Должна быть типа VARCHAR.

Возвращаемое значение

Возвращает контрольную сумму типа VARCHAR.

Примеры

```
mysql> select md5sum("selena");
+----------------------------------+
| md5sum('selena')                 |
+----------------------------------+
| f75523a916caf65f1ad487a9f8017f75 |
+----------------------------------+
1 row in set (0.01 sec)

mysql> select md5sum("selena","star");
+----------------------------------+
| md5sum('selena', 'star')         |
+----------------------------------+
| 7af4bfe35b8df2786ad133c57cb2aed8 |
+----------------------------------+
1 row in set (0.01 sec)

mysql> select md5sum(null);
+----------------------------------+
| md5sum(NULL)                     |
+----------------------------------+
| d41d8cd98f00b204e9800998ecf8427e |
+----------------------------------+
1 row in set (0.01 sec)
```

Md5sum_numeric

Вычисляет 128-битную контрольную сумму нескольких строк. Контрольная сумма представляется в виде десятичной строки цифр.

Синтаксис

```
LARGEINT md5sum_numeric(VARCHAR expr,...);
```

Параметры

`expr`: строка для вычисления. Должна быть типа VARCHAR.

Возвращаемое значение

Возвращает контрольную сумму типа LARGEINT. Если входное значение пустое, возвращается ошибка.

Примеры

```
mysql> select md5sum_numeric("selena");
+-----------------------------------------+
| md5sum_numeric('selena')                |
+-----------------------------------------+
| 328761383472799310362963866384446095221 |
+-----------------------------------------+
1 row in set (0.01 sec)

mysql> select md5sum_numeric("selena","star");
+-----------------------------------------+
| md5sum_numeric('selena', 'star')        |
+-----------------------------------------+
| 163436627872604162110984707546327199448 |
+-----------------------------------------+
1 row in set (0.01 sec)

mysql> select md5sum_numeric(null);
+-----------------------------------------+
| md5sum_numeric(NULL)                    |
+-----------------------------------------+
| 281949768489412648962353822266799178366 |
+-----------------------------------------+
1 row in set (0.01 sec)
```

Sha2

Вычисляет хеш-значение SHA-224, SHA-256, SHA-384 или SHA-512 для строки.

Синтаксис

```
VARCHAR sha2(VARCHAR expr, INT hash_length)
```

Параметры

- `Expr`: строка, для которой необходимо вычислить значение.
- `hash_length`: длина хеш-значения. Значение этого параметра может быть 224, 256, 384, 512 или 0. Значение 0 эквивалентно 256. Если вы установите любое другое значение, функция вернёт `NULL`.

Возвращаемое значение

Возвращает хеш-значение типа VARCHAR. Если любой из двух входных параметров равен `Null`, возвращается `Null`.

© ООО «Платформа Данных Селена», 2026

Примеры

```
mysql> select sha2('abc',224);

+----------------------------------------------------------+

| sha2('abc', 224)                                         |

+----------------------------------------------------------+

| 23097d223405d8228642a477bda255b32aadbce4bda0b3f7e36c9da7 |

+----------------------------------------------------------+

1 row in set (0.01 sec)



mysql> select sha2('abc', 384);

+------------------------------------------------------------------------------
---+

| sha2('abc', 384)
|

+------------------------------------------------------------------------------
---+

|
cb00753f45a35e8bb5a03d699ac65007272c32ab0eded1631a8b605a43ff5bed8086072ba1e7c
|

+------------------------------------------------------------------------------
---+

1 row in set (0.00 sec)



mysql> select sha2('abc', 1),sha2('abc', null), sha2(null, 384);

+----------------+------------------+-----------------+

| sha2('abc', 1) | sha2('abc', NULL) | sha2(NULL, 384) |
```

```
+----------------+-------------------+-----------------+

| NULL           | NULL              | NULL            |

+----------------+-------------------+-----------------+

1 row in set (0.01 sec)
```

Ключевые слова

SHA2, ENCRYPTION

Sm3

Использует алгоритм дайджеста SM3 для шифрования строки в 256-битную шестнадцатеричную строку. Каждые 32 бита разделяются пробелом.

Алгоритмы дайджеста имеют широкий спектр применения в цифровой подписи, аутентификации сообщений и проверке целостности данных. Алгоритм SM3 является улучшением SHA-256.

Синтаксис

```
SM3(str);
```

Параметры

`str`: строка для шифрования. Должна быть типа VARCHAR.

Возвращаемое значение

Возвращает значение типа VARCHAR. Если входное значение равно NULL, возвращается NULL. Если входное значение пустое, возвращается ошибка.

Эта функция принимает только одну строку. Передача более одной входной строки приводит к ошибке.

Примеры

```
mysql> select sm3('l');
+--------------------------------------------------------------------------+
| sm3('l')                                                                 |
+--------------------------------------------------------------------------+
| 1dec1a7a 94236240 49db411e 2c32c62d c0c93856 8208ac3a 09d395eb 2468b445 |
+--------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

To_base64

Преобразует строку в строку, закодированную в Base64. Эта функция является обратной к from_base64.

Синтаксис

```
to_base64(str);
```

Параметры

`str`: строка для кодирования. Должна быть типа VARCHAR.

Возвращаемое значение

Возвращает значение типа VARCHAR. Если входное значение равно NULL, возвращается NULL. Если входное значение пустое, возвращается ошибка.

Эта функция принимает только одну строку. Передача более одной входной строки приводит к ошибке.

Примеры

```
mysql> select to_base64("selena");
+------------------------+
| to_base64('selena') |
+------------------------+
| c3RhcnJvY2tz           |
+------------------------+
1 row in set (0.00 sec)

mysql> select to_base64(123);
+----------------+
| to_base64(123) |
+----------------+
| MTIz           |
+----------------+
```

2.14 Функции даты и времени

Add_months

Добавляет указанное количество месяцев к заданной дате (DATE или DATETIME). Функция months_add предоставляет аналогичный функционал.

Компонент дня в результирующем месяце остаётся таким же, как указано в `date`, за исключением случаев, когда в результирующем месяце меньше дней, чем компонент дня заданной даты — тогда день будет последним днём результирующего месяца. Например, `select add_months('2022-01-31', 1);` возвращает `2022-02-28 00:00:00`. Если в результирующем месяце больше дней, чем компонент дня заданной даты, результат имеет тот же компонент дня, что и `date`. Например, `select add_months('2022-02-28', 1)` возвращает `2022-03-28 00:00:00`.

Отличие от Oracle: в Oracle, если `date` является последним днём месяца, то результат будет последним днём результирующего месяца.

Возвращает NULL, если передана недопустимая дата или аргумент NULL.

Синтаксис

```
ADD_MONTH(date, months)
```

Параметры

- `date`: должно быть допустимым выражением даты или даты-времени.
- `months`: количество месяцев, которые вы хотите добавить. Должно быть целым числом. Положительное целое число добавляет месяцы к `date`. Отрицательное целое число вычитает месяцы из `date`.

Возвращаемое значение

Возвращает значение DATETIME. Если дата не существует, например, `2020-02-30`, возвращается NULL.

Если дата является значением DATE, оно будет преобразовано в значение DATETIME.

Примеры

```
> select add_months('2022-01-01', 2);
+-----------------------------+
| add_months('2022-01-01', 2) |
+-----------------------------+
| 2022-03-01 00:00:00         |
+-----------------------------+

> select add_months('2022-01-01', -5);
+------------------------------+
| add_months('2022-01-01', -5) |
+------------------------------+
| 2021-08-01 00:00:00          |
+------------------------------+

> select add_months('2022-01-31', 1);
+-----------------------------+
| add_months('2022-01-31', 1) |
+-----------------------------+
| 2022-02-28 00:00:00         |
+-----------------------------+

> select add_months('2022-01-31 17:01:02', -2);
+---------------------------------------+
| add_months('2022-01-31 17:01:02', -2) |
+---------------------------------------+
| 2021-11-30 17:01:02                   |
+---------------------------------------+

> select add_months('2022-02-28', 1);
+-----------------------------+
| add_months('2022-02-28', 1) |
+-----------------------------+
| 2022-03-28 00:00:00         |
+-----------------------------+
```

Adddate,date_add

Эта функция является синонимом date_add.

Convert_tz

Преобразует значение DATE или DATETIME из одного часового пояса в другой.

Эта функция может возвращать различные результаты для разных часовых поясов. Подробнее см. в разделе Настройка часового пояса.

Синтаксис

```
DATETIME CONVERT_TZ(DATETIME|DATE dt, VARCHAR from_tz, VARCHAR to_tz)
```

Параметры

- `dt`: значение DATE или DATETIME для преобразования.
- `from_tz`: исходный часовой пояс. Поддерживается VARCHAR. Часовой пояс может быть представлен в двух форматах: один — база данных часовых поясов (например, Asia/Shanghai), другой — смещение UTC (например, +08:00).
- `to_tz`: целевой часовой пояс. Поддерживается VARCHAR. Формат такой же, как у `from_tz`.

Возвращаемое значение

Возвращает значение типа данных DATETIME. Если входное значение — DATE, оно будет преобразовано в значение DATETIME. Эта функция возвращает NULL, если любой из входных параметров недопустим или равен NULL.

Примечания по использованию

Информацию о базе данных часовых поясов см. в Списке часовых поясов базы данных tz (в Википедии).

Примеры

Пример 1: Преобразование даты-времени из Шанхая в Лос-Анджелес.

```
select convert_tz('2019-08-01 13:21:03', 'Asia/Shanghai',
'America/Los_Angeles');
+-----------------------------------------------------------------------
-+
| convert_tz('2019-08-01 13:21:03', 'Asia/Shanghai', 'America/Los_Angeles')
|
+-----------------------------------------------------------------------
-+
| 2019-07-31 22:21:03
|
+-----------------------------------------------------------------------
-+
1 row in set (0.00 sec)
|
```

Пример 2: Преобразование даты из Шанхая в Лос-Анджелес.

```
select convert_tz('2019-08-01', 'Asia/Shanghai', 'America/Los_Angeles');
+-----------------------------------------------------------------+
| convert_tz('2019-08-01', 'Asia/Shanghai', 'America/Los_Angeles') |
+-----------------------------------------------------------------+
| 2019-07-31 09:00:00                                              |
+-----------------------------------------------------------------+
1 row in set (0.00 sec)
```

Пример 3: Преобразование даты-времени из UTC+08:00 в Лос-Анджелес.

```
select convert_tz('2019-08-01 13:21:03', '+08:00', 'America/Los_Angeles');
+-------------------------------------------------------------------+
| convert_tz('2019-08-01 13:21:03', '+08:00', 'America/Los_Angeles') |
+-------------------------------------------------------------------+
| 2019-07-31 22:21:03                                                |
+-------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Ключевые слова

CONVERT_TZ, timezone, time zone

Curdate,current_date

Возвращает текущую дату в виде значения типа DATE.

Синтаксис

```
DATE CURDATE()
```

Примеры

```
mysql> SELECT CURDATE();
+------------+
| curdate()  |
+------------+
| 2024-03-27 |
+------------+

mysql> SELECT CURRENT_DATE();
+----------------+
| current_date() |
+----------------+
| 2024-03-27     |
+----------------+

mysql> SELECT CURDATE() + 0;
+-----------------+
| (curdate()) + 0 |
+-----------------+
|        20240327 |
+-----------------+

mysql> SELECT CURDATE() +2;
+-----------------+
| (curdate()) + 2 |
+-----------------+
|        20240329 |
+-----------------+

mysql> SELECT CURDATE() - INTERVAL 5 DAY;
+----------------------------+
| curdate() - INTERVAL 5 DAY |
+----------------------------+
| 2024-03-22 00:00:00        |
+----------------------------+

mysql> SELECT QUARTER(CURDATE());
+--------------------+
| quarter(curdate()) |
+--------------------+
|                  1 |
+--------------------+
```

Current_timestamp

Возвращает текущую дату и время в виде значения типа DATETIME.

Эта функция является синонимом функции now().

Синтаксис

```
DATETIME CURRENT_TIMESTAMP()
DATETIME CURRENT_TIMESTAMP(INT p)
```

Параметры

`p`: необязательный, указанная точность, то есть количество цифр после секунд. Должно быть значением INT в диапазоне [1,6]. `select current_timestamp(0)` эквивалентно `select current_timestamp()`.

Возвращаемое значение

- Если `p` не указано, эта функция возвращает значение DATETIME с точностью до секунды.
- Если `p` указано, эта функция возвращает значение даты и времени с указанной точностью.

Примеры

Пример 1: Возврат текущего времени.

```
MySQL > select current_timestamp();
+---------------------+
| current_timestamp() |
+---------------------+
| 2019-05-27 15:59:33 |
+---------------------+
```

Пример 2: При создании таблицы вы можете использовать эту функцию для столбца, чтобы текущее время было значением по умолчанию для этого столбца.

```
CREATE TABLE IF NOT EXISTS sr_member (
    sr_id            INT,
    name             STRING,
    city_code        INT,
    reg_date         DATETIME DEFAULT current_timestamp,
    verified         BOOLEAN
);
```

Ключевые слова

CURRENT_TIMESTAMP,CURRENT,TIMESTAMP

Curtime,current_time

Возвращает текущее время в виде значения типа TIME.

Эта функция может возвращать различные результаты для разных часовых поясов. Подробнее см. в разделе Настройка часового пояса.

Синтаксис

```
TIME CURTIME()
```

Примеры

```
MySQL > select current_time();
+----------------+
| current_time() |
+----------------+
| 15:25:47       |
+----------------+
```

Ключевые слова

CURTIME,CURRENT_TIME

Date

Извлекает часть даты из выражения даты или даты-времени.

Синтаксис

```
DATE date(DATETIME|DATE expr)
```

Параметры

`expr`: выражение даты или даты-времени.

Возвращаемое значение

Возвращает значение типа DATE. Возвращается NULL, если входное значение равно NULL или недопустимо.

Примеры

Пример 1: Извлечение части даты из значения даты-времени.

```
SELECT DATE("2017-12-31 11:20:59");
+-----------------------------+
| date('2017-12-31 11:20:59') |
+-----------------------------+
| 2017-12-31                  |
+-----------------------------+
1 row in set (0.05 sec)
```

Пример 2: Извлечение части даты из значения даты.

```
SELECT DATE('2017-12-31');
+--------------------+
| date('2017-12-31') |
+--------------------+
| 2017-12-31         |
+--------------------+
1 row in set (0.08 sec)
```

Пример 3: Извлечение части даты из текущей временной метки.

```
SELECT DATE(current_timestamp());
+---------------------------+
| date(current_timestamp()) |
+---------------------------+
| 2022-11-08                |
+---------------------------+
1 row in set (0.05 sec)
```

Date_add,adddate

Добавляет указанный временной интервал к дате.

Синтаксис

```
DATETIME DATE_ADD(DATETIME|DATE date,INTERVAL expr type)
```

Параметры

- `date`: Должно быть допустимым выражением даты или даты-времени.
- `expr`: временной интервал, который вы хотите добавить. Должен быть типа INT.
- `type`: единица измерения временного интервала. Может быть установлен только в одно из следующих значений: YEAR, QUARTER, MONTH, DAY, HOUR, MINUTE, SECOND, MILLISECOND (с версии 3.1.7) и MICROSECOND (с версии 3.1.7).

Возвращаемое значение

Возвращает значение типа DATETIME. Если дата не существует, например, `2020-02-30`, возвращается NULL. Если дата является значением типа DATE, она будет преобразована в значение типа DATETIME.

Примеры

```
select date_add('2010-11-30 23:59:59', INTERVAL 2 DAY);
+--------------------------------------------------+
| date_add('2010-11-30 23:59:59', INTERVAL 2 DAY) |
+--------------------------------------------------+
| 2010-12-02 23:59:59                              |
+--------------------------------------------------+

select date_add('2010-12-03', INTERVAL 2 DAY);
+----------------------------------------+
| date_add('2010-12-03', INTERVAL 2 DAY) |
+----------------------------------------+
| 2010-12-05 00:00:00                    |
+----------------------------------------+

select date_add('2010-11-30 23:59:59', INTERVAL 2 QUARTER);
+-------------------------------------------------------+
| date_add('2010-11-30 23:59:59', INTERVAL 2 QUARTER) |
+-------------------------------------------------------+
| 2011-05-30 23:59:59                                   |
+-------------------------------------------------------+

select adddate('2023-10-31 23:59:59', INTERVAL 1 MILLISECOND);
+----------------------------------------------------------+
| adddate('2023-10-31 23:59:59', INTERVAL 1 MILLISECOND) |
+----------------------------------------------------------+
| 2023-10-31 23:59:59.001000                               |
+----------------------------------------------------------+

select adddate('2023-10-31 23:59:59', INTERVAL 1 MICROSECOND);
+----------------------------------------------------------+
| adddate('2023-10-31 23:59:59', INTERVAL 1 MICROSECOND) |
+----------------------------------------------------------+
| 2023-10-31 23:59:59.000001                               |
+----------------------------------------------------------+
```

Date_diff

Возвращает разницу между двумя значениями даты в указанных единицах. Эта функция возвращает значение `expr1 - expr2`, выраженное в единицах `unit`.

Например, `date_diff('second', '2010-11-30 23:59:59', '2010-11-30 20:58:59')` возвращает разницу между двумя значениями DATETIME в секундах.

Эта функция поддерживается с версии v1.5.2.

Разница между date_diff и datediff заключается в том, что datediff() не поддерживает параметр `unit`.

Синтаксис

```
BIGINT DATE_DIFF(VARCHAR unit, DATETIME expr1, DATETIME expr2)
```

Параметры

- `expr1` и `expr2`: два значения даты-времени, которые вы хотите сравнить, обязательные. Поддерживаемые типы данных: DATETIME и DATE.

- `unit`: единица измерения для выражения разницы во времени, обязательный. Поддерживаются следующие значения `unit`: year, quarter, month, week, day, hour, minute, second, millisecond.

Возвращаемое значение

Возвращает значение типа BIGINT.

Примечания по использованию

- Если `expr1` раньше, чем `expr2`, возвращается отрицательное значение.
- Если `unit` недопустим, возвращается ошибка.
- Если любое входное значение равно NULL, возвращается NULL.
- Если указанная дата не существует, например, `2022-11-31`, возвращается NULL.

Примеры

```
mysql> select date_diff('second', '2010-11-30 23:59:59', '2010-11-30
20:58:59');
+-----------------------------------------------------------------+
| date_diff('second', '2010-11-30 23:59:59', '2010-11-30 20:58:59') |
+-----------------------------------------------------------------+
|                                                           10860 |
+-----------------------------------------------------------------+

mysql> select date_diff('minute', '2010-11-30 23:59:59', '2010-11-30
20:58:59');
+-----------------------------------------------------------------+
| date_diff('minute', '2010-11-30 23:59:59', '2010-11-30 20:58:59') |
+-----------------------------------------------------------------+
|                                                             181 |
+-----------------------------------------------------------------+

mysql> select date_diff('hour', '2010-11-30 23:59:59', '2010-11-30
20:58:59');
+----------------------------------------------------------------+
| date_diff('hour', '2010-11-30 23:59:59', '2010-11-30 20:58:59') |
+----------------------------------------------------------------+
|                                                              3 |
+----------------------------------------------------------------+

mysql> select date_diff('millisecond', '2010-11-30 23:59:59', '2010-11-30
20:58:59');
+---------------------------------------------------------------------+
| date_diff('millisecond', '2010-11-30 23:59:59', '2010-11-30 20:58:59') |
+---------------------------------------------------------------------+
|                                                            10860000 |
+---------------------------------------------------------------------+
```

Ссылки

datediff

Date_format

Преобразует дату в строку в соответствии с указанным форматом. В настоящее время поддерживаются строки длиной до 128 байт. Если длина возвращаемого значения превышает 128, возвращается NULL.

Синтаксис

```
VARCHAR DATE_FORMAT(DATETIME date, VARCHAR format)
```

Параметры

- Параметр `date` должен быть допустимой датой или выражением даты.
- `format` задаёт выходной формат даты или времени.

Доступные форматы:

```
%a | Сокращённое название дня недели (Sun — Sat)
%b | Сокращённое название месяца (Jan — Dec)
%c | Числовое название месяца (0-12)
%D | День месяца с суффиксом на английском языке
%d | День месяца как числовое значение (00-31)
%e | День месяца как числовое значение (0-31)
%f | Микросекунды
%H | Час (00-23)
%h | Час (01-12)
%I | Час (01-12)
%i | Минуты (00-59)
%j | День года (001-366)
%k | Час (0-23)
%l | Час (1-12)
%M | Полное название месяца
%m | Название месяца как числовое значение (00-12)
%p | AM или PM
%r | Время в 12-часовом формате (hh:mm:ss AM или PM)
%S | Секунды (00-59)
%s | Секунды (00-59)
%T | Время в 24-часовом формате (hh:mm:ss)
%U | Неделя (00-53), где воскресенье — первый день недели
%u | Неделя (00-53), где понедельник — первый день недели
%V | Неделя (01-53), где воскресенье — первый день недели. Используется с
%X.
%v | Неделя (01-53), где понедельник — первый день недели. Используется с
%x.
%W | Полное название дня недели
%w | День недели, где воскресенье=0 и суббота=6
%X | Год для недели, где воскресенье — первый день недели. 4-значное
значение. Используется с %V.
%x | Год для недели, где понедельник — первый день недели. 4-значное
значение. Используется с %v.
%Y | Год. 4-значное значение.
%y | Год. 2-значное значение.
%% | Представляет %.
```

Примеры

```
MySQL > select date_format('2009-10-04 22:23:00', '%W %M %Y');
+-----------------------------------------------+
| date_format('2009-10-04 22:23:00', '%W %M %Y') |
+-----------------------------------------------+
| Sunday October 2009                           |
+-----------------------------------------------+

MySQL > select date_format('2007-10-04 22:23:00', '%H:%i:%s');
+-----------------------------------------------+
| date_format('2007-10-04 22:23:00', '%H:%i:%s') |
+-----------------------------------------------+
| 22:23:00                                      |
+-----------------------------------------------+

MySQL > select date_format('1900-10-04 22:23:00', '%D %y %a %d %m %b %j');
+-------------------------------------------------------------+
| date_format('1900-10-04 22:23:00', '%D %y %a %d %m %b %j') |
+-------------------------------------------------------------+
| 4th 00 Thu 04 10 Oct 277                                    |
+-------------------------------------------------------------+

MySQL > select date_format('1997-10-04 22:23:00', '%H %k %I %r %T %S %w');
+-------------------------------------------------------------+
| date_format('1997-10-04 22:23:00', '%H %k %I %r %T %S %w') |
+-------------------------------------------------------------+
| 22 22 10 10:23:00 PM 22:23:00 00 6                          |
+-------------------------------------------------------------+

MySQL > select date_format('1999-01-01 00:00:00', '%X %V');
+----------------------------------------------+
| date_format('1999-01-01 00:00:00', '%X %V') |
+----------------------------------------------+
| 1998 52                                      |
+----------------------------------------------+

MySQL > select date_format('2006-06-01', '%d');
+------------------------------------------+
| date_format('2006-06-01 00:00:00', '%d') |
+------------------------------------------+
| 01                                       |
+------------------------------------------+

MySQL > select date_format('2006-06-01', '%%%d');
```

```
+-------------------------------------------+
| date_format('2006-06-01 00:00:00', '%%%d') |
+-------------------------------------------+
| %01                                       |
+-------------------------------------------+
```

Keyword

DATE_FORMAT,DATE,FORMAT

Date_slice

Преобразует заданное время в начало или конец временного интервала на основе указанной временной гранулярности.

Эта функция поддерживается с версии v1.5.2.

Синтаксис

```
DATE date_slice(DATE dt, INTERVAL N type[, boundary])
```

Параметры

- `dt`: время для преобразования, DATE.
- `INTERVAL N type`: временная гранулярность, например, `interval 5 day`.
  - `N` — длина временного интервала. Должно быть значением типа INT.
  - `type` — единица измерения: YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, SECOND. Если `type` установлен в HOUR, MINUTE или SECOND для значения DATE, возвращается ошибка.
- `boundary`: необязательный параметр. Используется для указания, возвращать начало (`FLOOR`) или конец (`CEIL`) временного интервала. Допустимые значения: FLOOR, CEIL. Если этот параметр не указан, по умолчанию используется `FLOOR`.

Возвращаемое значение

Возвращает значение типа DATE.

Примечания по использованию

Временной интервал начинается с `0001-01-01 00:00:00` н.э.

Примеры

Пример 1: Преобразовать заданное время в начало 5-летнего временного интервала без указания параметра `boundary`.

```
select date_slice('2022-04-26', interval 5 year);
+--------------------------------------------------+
| date_slice('2022-04-26', INTERVAL 5 year, floor) |
+--------------------------------------------------+
| 2021-01-01                                       |
+--------------------------------------------------+
```

Пример 2: Преобразовать заданное время в конец 5-дневного временного интервала.

```
select date_slice('0001-01-07', interval 5 day, CEIL);
+-------------------------------------------------+
| date_slice('0001-01-07', INTERVAL 5 day, ceil) |
+-------------------------------------------------+
| 0001-01-11                                      |
+-------------------------------------------------+
```

Следующие примеры основаны на таблице `test_all_type_select`.

```
select * from test_all_type_select order by id_int;
+------------+---------------------+--------+
| id_date    | id_datetime         | id_int |
+------------+---------------------+--------+
| 2052-12-26 | 1691-12-23 04:01:09 |      0 |
| 2168-08-05 | 2169-12-18 15:44:31 |      1 |
| 1737-02-06 | 1840-11-23 13:09:50 |      2 |
| 2245-10-01 | 1751-03-21 00:19:04 |      3 |
| 1889-10-27 | 1861-09-12 13:28:18 |      4 |
+------------+---------------------+--------+
5 rows in set (0.06 sec)
```

Пример 3: Преобразовать заданное значение DATE в начало 5-секундного временного интервала.

```
select date_slice(id_date, interval 5 second, FLOOR)
from test_all_type_select
order by id_int;
ERROR 1064 (HY000): can't use date_slice for date with
time(hour/minute/second)
```

Возвращается ошибка, потому что система не может найти часть секунд в значении DATE.

Пример 4: Преобразовать заданное значение DATE в начало 5-дневного временного интервала.

```
select date_slice(id_date, interval 5 day, FLOOR)
from test_all_type_select
order by id_int;
+---------------------------------------------+
| date_slice(id_date, INTERVAL 5 day, floor) |
+---------------------------------------------+
| 2052-12-24                                  |
| 2168-08-03                                  |
| 1737-02-04                                  |
| 2245-09-29                                  |
| 1889-10-25                                  |
+---------------------------------------------+
5 rows in set (0.14 sec)
```

Пример 5: Преобразовать заданное значение DATE в конец 5-дневного временного интервала.

```
select date_slice(id_date, interval 5 day, CEIL)
from test_all_type_select
order by id_int;
+-------------------------------------------+
| date_slice(id_date, INTERVAL 5 day, ceil) |
+-------------------------------------------+
| 2052-12-29                                |
| 2168-08-08                                |
| 1737-02-09                                |
| 2245-10-04                                |
| 1889-10-30                                |
+-------------------------------------------+
5 rows in set (0.17 sec)
```

Date_sub,subdate

Вычитает указанный временной интервал из даты.

Синтаксис

```
DATETIME DATE_SUB(DATETIME|DATE date,INTERVAL expr type)
```

Параметры

- `date`: Должно быть допустимым выражением DATE или DATETIME.
- `expr`: временной интервал, который вы хотите вычесть. Должен быть типа INT.
- `type`: единица измерения временного интервала. Может быть установлен только в одно из следующих значений: YEAR, QUARTER, MONTH, DAY, HOUR, MINUTE, SECOND, MILLISECOND (с версии 3.1.7) и MICROSECOND (с версии 3.1.7).

Возвращаемое значение

Возвращает значение типа DATETIME. Если дата не существует, например, `2020-02-30`, или если дата не является значением DATE или DATETIME, возвращается NULL.

Примеры

```
select date_sub('2010-11-30 23:59:59', INTERVAL 2 DAY);
+--------------------------------------------------+
| date_sub('2010-11-30 23:59:59', INTERVAL 2 DAY) |
+--------------------------------------------------+
| 2010-11-28 23:59:59                              |
+--------------------------------------------------+

select date_sub('2010-11-30', INTERVAL 2 hour);
+-----------------------------------------+
| date_sub('2010-11-30', INTERVAL 2 HOUR) |
+-----------------------------------------+
| 2010-11-29 22:00:00                     |
+-----------------------------------------+

select date_sub('2010-02-30', INTERVAL 2 DAY);
+----------------------------------------+
| date_sub('2010-02-30', INTERVAL 2 DAY) |
+----------------------------------------+
| NULL                                   |
+----------------------------------------+

select date_sub('2010-11-30 23:59:59', INTERVAL 2 QUARTER);
+-------------------------------------------------------+
| date_sub('2010-11-30 23:59:59', INTERVAL 2 QUARTER) |
+-------------------------------------------------------+
| 2010-05-30 23:59:59                                   |
+-------------------------------------------------------+

select subdate('2010-11-30 23:59:59', INTERVAL 2 millisecond);
+---------------------------------------------------------+
| subdate('2010-11-30 23:59:59', INTERVAL 2 MILLISECOND) |
+---------------------------------------------------------+
| 2010-11-30 23:59:58.998000                              |
+---------------------------------------------------------+

select date_sub('2010-11-30 23:59:59', INTERVAL 2 microsecond);
+----------------------------------------------------------+
| date_sub('2010-11-30 23:59:59', INTERVAL 2 MICROSECOND) |
+----------------------------------------------------------+
| 2010-11-30 23:59:58.999998                               |
+----------------------------------------------------------+
```

Date_trunc

Усекает значение времени на основе указанной части даты, такой как год, день, час или минута.

Selena также предоставляет функции year, quarter, month, week, day и hour для извлечения указанной части даты.

Синтаксис

```
DATETIME|DATE date_trunc(VARCHAR fmt, DATETIME|DATE datetime)
```

Параметры

- `datetime`: время для усечения, которое может быть типа DATETIME или DATE. Дата и время должны существовать. В противном случае будет возвращён NULL. Например, `2021-02-29 11:12:13` не существует как дата, и будет возвращён NULL.

- `fmt`: часть даты, то есть до какой точности будет усечено `datetime`. Значение должно быть константой типа VARCHAR.

  `fmt` должен быть установлен в значение из следующей таблицы. Если значение некорректно, будет возвращена ошибка.

  Если `datetime` является значением DATE, `fmt` может быть только `year`, `quarter`, `month`, `week` или `day`. Если вы установите `fmt` в другие единицы даты, например `hour`, будет возвращена ошибка. См. пример 8.

Значение	Описание
microsecond	Усекает до микросекунды (с версии 3.1.7).
millisecond	Усекает до миллисекунды (с версии 3.1.7).
second	Усекает до секунды.
minute	Усекает до минуты. Часть секунд обнуляется.
hour	Усекает до часа. Части минут и секунд обнуляются.
day	Усекает до дня. Часть времени обнуляется.
week	Усекает до первой даты недели, в которую попадает `datetime`. Часть времени обнуляется.
month	Усекает до первой даты месяца, в который попадает `datetime`. Часть времени обнуляется.
quarter	Усекает до первой даты квартала, в который попадает `datetime`. Часть времени обнуляется.

Значение	Описание
year	Усекает до первой даты года, в который попадает `datetime`. Часть времени обнуляется.

Возвращаемое значение

Возвращает значение типа DATETIME или DATE.

Если `datetime` имеет тип DATE и `fmt` установлен в `hour`, `minute` или `second`, часть времени возвращаемого значения по умолчанию `00:00:00`.

Примеры

Пример 1: Усечь входное время до микросекунды.

```
select date_trunc('microsecond', '2023-10-31 23:59:59.001002');
+-----------------------------------------------------------+
| date_trunc('microsecond', '2023-10-31 23:59:59.001002') |
+-----------------------------------------------------------+
| 2023-10-31 23:59:59.001002                                |
+-----------------------------------------------------------+
```

Пример 2: Усечь входное время до миллисекунды.

```
select date_trunc('millisecond', '2023-10-31 23:59:59.001002');
+-----------------------------------------------------------+
| date_trunc('millisecond', '2023-10-31 23:59:59.001002') |
+-----------------------------------------------------------+
| 2023-10-31 23:59:59.001000                                |
+-----------------------------------------------------------+

select date_trunc('millisecond', '2023-10-31');
+-----------------------------------------+
| date_trunc('millisecond', '2023-10-31') |
+-----------------------------------------+
| 2023-10-31 00:00:00                     |
+-----------------------------------------+
```

Пример 3: Усечь входное время до минуты.

```
select date_trunc('minute', '2023-11-04 23:59:59.001002');
+-----------------------------------------------------+
| date_trunc('minute', '2023-11-04 23:59:59.001002') |
+-----------------------------------------------------+
| 2023-11-04 23:59:00                                 |
+-----------------------------------------------------+
```

Пример 4: Усечь входное время до часа.

```
select date_trunc("hour", "2020-11-04 11:12:13");
+-------------------------------------------+
| date_trunc('hour', '2020-11-04 11:12:13') |
+-------------------------------------------+
| 2020-11-04 11:00:00                       |
+-------------------------------------------+
```

Пример 5: Усечь входное время до первого дня недели.

```
select date_trunc("week", "2020-11-04 11:12:13");
+-------------------------------------------+
| date_trunc('week', '2020-11-04 11:12:13') |
+-------------------------------------------+
| 2020-11-02 00:00:00                       |
+-------------------------------------------+
```

Пример 6: Усечь входное время до первого дня квартала.

```
select date_trunc("quarter", "2020-11-04 11:12:13");
+----------------------------------------------+
| date_trunc('quarter', '2020-11-04 11:12:13') |
+----------------------------------------------+
| 2020-10-01 00:00:00                          |
+----------------------------------------------+
```

Пример 7: Усечь входное время до первого дня года.

```
select date_trunc('year', '2023-11-04 23:59:59.001002');
+---------------------------------------------------+
| date_trunc('year', '2023-11-04 23:59:59.001002') |
+---------------------------------------------------+
| 2023-01-01 00:00:00                               |
+---------------------------------------------------+
```

Пример 8: Усечь значение DATE до часа. Возвращается ошибка.

```
select date_trunc("hour", cast("2020-11-04" as date));

ERROR 1064 (HY000): Getting analyzing error from line 1, column 26 to line
1, column 51. Detail message: date_trunc function can't support argument
other than year|quarter|month|week|day.
```

Пример 9: Усечь входную дату до первого дня года.

```
select date_trunc("month", cast("2020-11-04" as date));
+-------------------------------------------------+
| date_trunc('month', CAST('2020-11-04' AS DATE)) |
+-------------------------------------------------+
| 2020-11-01                                      |
+-------------------------------------------------+
```

Datediff

Вычисляет разницу между двумя значениями даты (`expr1 - expr2`) и возвращает результат в днях. `expr1` и `expr2` — это допустимые выражения DATE или DATETIME.

Примечание: В вычислении используются только части дат значений.

Разница между datediff и days_diff:

Функция	Поведение	Пример
datediff	Точность до дня	Разница между '2020-12-25 23:00:00' и '2020-12-24 23:00:01' равна 1.
days_diff	Точность до секунды с округлением вниз до ближайшего целого	Разница между '2020-12-25 23:00:00' и '2020-12-24 23:00:01' равна 0.

Разница между datediff и date_diff заключается в том, что date_diff поддерживает параметр `unit`, который может возвращать разницу между двумя значениями даты в указанных единицах.

Синтаксис

```
INT DATEDIFF(DATETIME expr1,DATETIME expr2)
```

Примеры

```
MySQL > select datediff(CAST('2007-12-31 23:59:59' AS DATETIME),
CAST('2007-12-30' AS DATETIME));
+-----------------------------------------------------------------------------
---------+
| datediff(CAST('2007-12-31 23:59:59' AS DATETIME), CAST('2007-12-30' AS
DATETIME)) |
+-----------------------------------------------------------------------------
---------+
|
1 |
+-----------------------------------------------------------------------------
---------+

MySQL > select datediff(CAST('2010-11-30 23:59:59' AS DATETIME),
CAST('2010-12-31' AS DATETIME));
+-----------------------------------------------------------------------------
---------+
| datediff(CAST('2010-11-30 23:59:59' AS DATETIME), CAST('2010-12-31' AS
DATETIME)) |
+-----------------------------------------------------------------------------
---------+
|
-31 |
+-----------------------------------------------------------------------------
---------+
```

Ссылки

date_diff

Day

Извлекает часть дня из выражения даты или даты-времени и возвращает значение в диапазоне от 1 до 31.

Параметр `date` должен быть типа DATE или DATETIME.

Синтаксис

```
INT DAY(DATETIME|DATE date)
```

Примеры

```
MySQL > select day('1987-01-31');
+----------------------------+
| day('1987-01-31 00:00:00') |
+----------------------------+
|                         31 |
+----------------------------+

MySQL > select day('1987-01-31 20:10:59');
+----------------------------+
| day('1987-01-31 20:10:59') |
+----------------------------+
|                         31 |
+----------------------------+
```

Keywords

DAY, day

Dayofweek_iso

Возвращает день недели по стандарту ISO для указанной даты как целое число в диапазоне от `1` до `7`. В этом стандарте `1` представляет понедельник, а `7` — воскресенье.

Синтаксис

```
INT DAY_OF_WEEK_ISO(DATETIME date)
```

Параметры

`date`: дата, которую вы хотите преобразовать. Должна быть типа DATE или DATETIME.

Примеры

Следующий пример возвращает день недели по стандарту ISO для даты `2023-01-01`:

```
MySQL > select dayofweek_iso('2023-01-01');
+-----------------------------+
| dayofweek_iso('2023-01-01') |
+-----------------------------+
|                           7 |
+-----------------------------+
```

Keywords

DAY_OF_WEEK_ISO

Dayname

Возвращает название дня недели для заданной даты.

Параметр `date` должен быть типа DATE или DATETIME.

Синтаксис

```
VARCHAR DAYNAME(date)
```

Примеры

```
MySQL > select dayname('2007-02-03 00:00:00');
+--------------------------------+
| dayname('2007-02-03 00:00:00') |
+--------------------------------+
| Saturday                       |
+--------------------------------+
```

Keyword

DAYNAME

Dayofmonth

Получает часть дня из даты и возвращает значение в диапазоне от 1 до 31.

Параметр `date` должен быть типа DATE или DATETIME.

Синтаксис

```
INT DAYOFMONTH(DATETIME date)
```

Примеры

```
MySQL > select dayofmonth('1987-01-31');
+-----------------------------------+
| dayofmonth('1987-01-31 00:00:00') |
+-----------------------------------+
|                                31 |
+-----------------------------------+
```

Keyword

DAYOFMONTH

Dayofweek

Возвращает индекс дня недели для заданной даты. Например, индекс для воскресенья — 1, для понедельника — 2, для субботы — 7.

Параметр `date` должен быть типа DATE или DATETIME, либо допустимым выражением, которое может быть преобразовано в значение DATE или DATETIME.

Синтаксис

```
INT dayofweek(DATETIME date)
```

Примеры

```
MySQL > select dayofweek('2019-06-25');
+-----------------------------------+
| dayofweek('2019-06-25 00:00:00') |
+-----------------------------------+
|                                 3 |
+-----------------------------------+

MySQL > select dayofweek(cast(20190625 as date));
+------------------------------------+
| dayofweek(CAST(20190625 AS DATE)) |
+------------------------------------+
|                                  3 |
+------------------------------------+
```

Keyword

DAYOFWEEK

Dayofyear

Возвращает номер дня в году для заданной даты.

Параметр `date` должен быть типа DATE или DATETIME.

Синтаксис

```
INT DAYOFYEAR(DATETIME|DATE date)
```

Примеры

```
MySQL > select dayofyear('2007-02-03 00:00:00');
+-----------------------------------+
| dayofyear('2007-02-03 00:00:00') |
+-----------------------------------+
|                                34 |
+-----------------------------------+
```

Keyword

DAYOFYEAR

Days_add

Добавляет указанное количество дней к заданной дате или дате-времени.

Синтаксис

```
DATETIME days_add(DATETIME|DATE date, INT n);
```

Параметры

`date`: выражение DATE или DATETIME.

`n`: количество дней для добавления.

Возвращаемое значение

Возвращает значение типа DATETIME.

Если любой входной параметр равен NULL или недействителен, возвращается NULL.

Если выходная дата выходит за пределы диапазона [0000-01-01 00:00:00, 9999-12-31 00:00:00], возвращается NULL.

Примеры

```
select days_add('2023-10-31 23:59:59', 1);
+-----------------------------------+
| days_add('2023-10-31 23:59:59', 1) |
+-----------------------------------+
| 2023-11-01 23:59:59               |
+-----------------------------------+

select days_add('2023-10-31 23:59:59', 1000);
+---------------------------------------+
| days_add('2023-10-31 23:59:59', 1000) |
+---------------------------------------+
| 2026-07-27 23:59:59                   |
+---------------------------------------+

select days_add('2023-10-31', 1);
+---------------------------+
| days_add('2023-10-31', 1) |
+---------------------------+
| 2023-11-01 00:00:00       |
+---------------------------+
```

Keywords

DAY,day

Days_diff

Возвращает разницу в днях между двумя выражениями даты (`expr1` – `expr2`), точность до дня.

Разница между days_diff и datediff:

Функция	Поведение	Пример
days_diff	Вычисление с точностью до секунды и округлением вниз до ближайшего целого.	days_diff между '2020-12-25 23:00:00' и '2020-12-24 23:00:01' равен 0.
datediff	Вычисление с точностью до дня.	datediff между '2020-12-25 23:00:00' и '2020-12-24 23:00:01' равен 1.

Синтаксис

```
BIGINT days_diff(DATETIME expr1,DATETIME expr2);
```

Параметры

- `expr1`: конечное время. Должно быть типа DATETIME или DATE.
- `expr2`: начальное время. Должно быть типа DATETIME или DATE.

Возвращаемое значение

Возвращает значение типа BIGINT.

Возвращается NULL, если дата не существует, например, 2022-02-29. days_diff() учитывает части часов, минут и секунд в вычислении. Если разница меньше 1 дня, возвращается 0.

Примеры

```
select days_diff('2010-11-30 23:00:00', '2010-11-29 23:00:00')
+-----------------------------------------------------------+
| days_diff('2010-11-30 23:00:00', '2010-11-29 23:00:00') |
+-----------------------------------------------------------+
| 1                                                         |
+-----------------------------------------------------------+

select days_diff('2010-11-30 23:00:00', '2010-11-29 23:10:00')
+-----------------------------------------------------------+
| days_diff('2010-11-30 23:00:00', '2010-11-29 23:10:00') |
+-----------------------------------------------------------+
| 0                                                         |
+-----------------------------------------------------------+
```

Days_sub

Вычитает указанное количество дней из даты или даты-времени для получения новой даты-времени.

Синтаксис

```
DATETIME days_sub(DATETIME|DATE d, INT n);
```

Параметры

`d`: выражение даты или даты-времени.

`n`: количество дней для вычитания из `d`.

Возвращаемое значение

Возвращает значение типа datetime, NULL возвращается, если любой параметр равен NULL или недействителен.

NULL возвращается, если результат выходит за пределы диапазона [0000-01-01 00:00:00, 9999-12-31 00:00:00].

Примеры

```
SELECT DAYS_SUB('2022-12-20 12:00:00', 10);
+-----------------------------------------------------+
| days_sub('2022-12-20 12:00:00', INTERVAL 10 DAY) |
+-----------------------------------------------------+
| 2022-12-10 12:00:00                                 |
+-----------------------------------------------------+

SELECT DAYS_SUB('2022-12-20 12:00:00', -10);
+-----------------------------------------------------+
| days_sub('2022-12-20 12:00:00', INTERVAL -10 DAY) |
+-----------------------------------------------------+
| 2022-12-30 12:00:00                                 |
+-----------------------------------------------------+

SELECT DAYS_SUB('2022-12-20 12:00:00', 738874);
+-------------------------------------------------------+
| days_sub('2022-12-20 12:00:00', INTERVAL 738874 DAY) |
+-------------------------------------------------------+
| 0000-01-01 12:00:00                                   |
+-------------------------------------------------------+

SELECT DAYS_SUB('2022-12-20 12:00:00', 738875);
+-------------------------------------------------------+
| days_sub('2022-12-20 12:00:00', INTERVAL 738875 DAY) |
+-------------------------------------------------------+
| NULL                                                  |
+-------------------------------------------------------+

SELECT DAYS_SUB('2022-12-20 12:00:00', -2913550);
+---------------------------------------------------------+
| days_sub('2022-12-20 12:00:00', INTERVAL -2913550 DAY) |
+---------------------------------------------------------+
| 9999-12-31 12:00:00                                     |
+---------------------------------------------------------+

SELECT DAYS_SUB('2022-12-20 12:00:00', -2913551);
+---------------------------------------------------------+
| days_sub('2022-12-20 12:00:00', INTERVAL -2913551 DAY) |
+---------------------------------------------------------+
| NULL                                                    |
+---------------------------------------------------------+
```

Keywords

DAY,day

From_days

Возвращает дату, вычисленную от даты 0000-01-01.

Синтаксис

```
DATE FROM_DAYS(INT N)
```

Примеры

```
MySQL > select from_days(730669);
+-------------------+
| from_days(730669) |
+-------------------+
| 2000-07-03        |
+-------------------+
```

Ключевые слова

FROM_DAYS,FROM,DAYS

From_unixtime

Преобразует временную метку UNIX в требуемый формат времени. Формат по умолчанию - `yyyy-MM-dd HH:mm:ss`. Также поддерживаются форматы из date_format.

В настоящее время `string_format` поддерживает следующие форматы:

```
%Y: Год  например: 2014, 1900
%m: Месяц   например: 12, 09
%d: День  например: 11, 01
%H: Час  например: 23, 01, 12
%i: Минута  например: 05, 11
%s: Секунда  например: 59, 01
```

Другие форматы недопустимы и возвращают NULL.

Синтаксис

```
VARCHAR from_unixtime(BIGINT unix_timestamp[, VARCHAR string_format])
```

Параметры

- `unix_timestamp`: временная метка UNIX, которую вы хотите преобразовать. Должна быть типа BIGINT. Если указанная временная метка меньше 0 или больше 253402243199, возвращается NULL. То есть диапазон для временной метки - от `1970-01-01 00:00:00` до `9999-12-30 11:59:59` (варьируется в зависимости от часового пояса).

- `string_format`: требуемый формат времени.

Возвращаемое значение

Возвращает значение DATETIME или DATE типа VARCHAR. Если `string_format` указывает формат DATE, возвращается значение DATE типа VARCHAR.

Если временная метка превышает допустимый диапазон или если `string_format` недопустим, возвращается NULL.

Примеры

```
MySQL > select from_unixtime(1196440219);
+----------------------------+
| from_unixtime(1196440219) |
+----------------------------+
| 2007-12-01 00:30:19       |
+----------------------------+

MySQL > select from_unixtime(1196440219, 'yyyy-MM-dd HH:mm:ss');
+---------------------------------------------------+
| from_unixtime(1196440219, 'yyyy-MM-dd HH:mm:ss') |
+---------------------------------------------------+
| 2007-12-01 00:30:19                               |
+---------------------------------------------------+

MySQL > select from_unixtime(1196440219, '%Y-%m-%d');
+-----------------------------------------+
| from_unixtime(1196440219, '%Y-%m-%d')   |
+-----------------------------------------+
| 2007-12-01                              |
+-----------------------------------------+

MySQL > select from_unixtime(1196440219, '%Y-%m-%d %H:%i:%s');
+---------------------------------------------------+
| from_unixtime(1196440219, '%Y-%m-%d %H:%i:%s')   |
+---------------------------------------------------+
| 2007-12-01 00:30:19                               |
+---------------------------------------------------+
```

Ключевые слова

FROM_UNIXTIME,FROM,UNIXTIME

Hour

Возвращает час для заданной даты. Возвращаемое значение находится в диапазоне от 0 до 23.

Параметр `date` должен быть типа DATE или DATETIME.

Синтаксис

```
INT HOUR(DATETIME|DATE date)
```

Примеры

```
MySQL > select hour('2018-12-31 23:59:59');
+-----------------------------+
| hour('2018-12-31 23:59:59') |
+-----------------------------+
|                          23 |
+-----------------------------+
```

Ключевые слова

HOUR

Hours_add

Добавляет часы к дате или datetime.

Синтаксис

```
DATETIME hours_add(DATETIME|DATE date, INT hours);
```

Параметры

`date`: базовая дата или datetime.

`hours`: количество часов для добавления.

Возвращаемое значение

Возвращает значение типа DATETIME.

Если любой из входных параметров равен NULL, возвращается NULL.

Примеры

```
select hours_add('2022-01-01 01:01:01', 2);
+-------------------------------------+
| hours_add('2022-01-01 01:01:01', 2) |
+-------------------------------------+
| 2022-01-01 03:01:01                 |
+-------------------------------------+

select hours_add('2022-01-01 01:01:01', -1);
+--------------------------------------+
| hours_add('2022-01-01 01:01:01', -1) |
+--------------------------------------+
| 2022-01-01 00:01:01                  |
+--------------------------------------+

select hours_add('2022-01-01', 1);
+----------------------------+
| hours_add('2022-01-01', 1) |
+----------------------------+
| 2022-01-01 01:00:00        |
+----------------------------+
```

Hours_diff

Возвращает разницу в часах между двумя выражениями даты (`expr1` – `expr2`) с точностью до часа.

Синтаксис

```
BIGINT hours_diff(DATETIME expr1, DATETIME expr2);
```

Параметры

- `expr1`: время окончания. Должно быть типа DATETIME.
- `expr2`: время начала. Должно быть типа DATETIME.

Возвращаемое значение

Возвращает значение типа BIGINT.

Если дата не существует, например 2022-02-29, возвращается NULL.

Примеры

```
select hours_diff('2010-11-30 23:59:59', '2010-11-30 20:58:59');
+-----------------------------------------------------------+
| hours_diff('2010-11-30 23:59:59', '2010-11-30 20:58:59') |
+-----------------------------------------------------------+
|                                                         3 |
+-----------------------------------------------------------+
```

Hours_sub

Уменьшает указанную дату и время на заданное количество часов.

Синтаксис

```
DATETIME hours_sub(DATETIME|DATE date, INT hours);
```

Параметры

- `date`: должно быть допустимым выражением DATE или DATETIME.
- `hours`: количество часов для вычитания. Поддерживаемый тип данных - INT.

Возвращаемое значение

Возвращает значение типа DATETIME. Если дата не существует, например `2022-21-01`, или если дата не является значением DATE или DATETIME, возвращается NULL.

Примеры

```
select hours_sub('2022-01-01 01:01:01', 1);
+-------------------------------------+
| hours_sub('2022-01-01 01:01:01', 1) |
+-------------------------------------+
| 2022-01-01 00:01:01                 |
+-------------------------------------+

select hours_sub('2022-01-01 01:01:01', -1);
+--------------------------------------+
| hours_sub('2022-01-01 01:01:01', -1) |
+--------------------------------------+
| 2022-01-01 02:01:01                  |
+--------------------------------------+

select hours_sub('2022-01-01', 1);
+----------------------------+
| hours_sub('2022-01-01', 1) |
+----------------------------+
| 2021-12-31 23:00:00        |
+----------------------------+

select hours_sub('2022-01-01', -1);
+-----------------------------+
| hours_sub('2022-01-01', -1) |
+-----------------------------+
| 2022-01-01 01:00:00         |
+-----------------------------+

Случай ошибки:
select hours_sub('2022-21-01', -1);
+--------------------------------------+
| hours_sub('2022-21-01', -1) |
+--------------------------------------+
| NULL                                 |
+--------------------------------------+
```

Jodatime_format

Преобразует указанную дату в строку в указанном формате шаблона Joda DateTimeFormat.

Синтаксис

```
VARCHAR JODATIME_FORMAT(DATETIME | DATE date, VARCHAR format)
```

Параметры

- `date`: дата, которую вы хотите преобразовать. Должна быть допустимой датой или выражением даты.
- `format`: формат возвращаемой даты и времени. Информацию о доступных форматах см. в Joda-Time format.

Возвращаемое значение

В настоящее время возвращаемое значение представляет собой строку максимальной длины 128 байт. Если длина строки, преобразованной из указанной даты, превышает 128 байт, возвращается `NULL`.

Примеры

Пример 1: Преобразовать дату `2020-06-25 15:58:51` в строку в формате `yyyy-MM-dd`.

```
MySQL > select jodatime_format('2020-06-25 15:58:51', 'yyyy-MM-dd');
+-----------------------------------------------------+
| jodatime_format('2020-06-25 15:58:51', 'yyyy-MM-dd') |
+-----------------------------------------------------+
| 2020-06-25                                          |
+-----------------------------------------------------+
```

Пример 2: Преобразовать дату `2020-06-25 15:58:51` в строку в формате `yyyy-MM-dd HH:mm:ss`.

```
MySQL > select jodatime_format('2020-06-25 15:58:51', 'yyyy-MM-dd
HH:mm:ss');
+-------------------------------------------------------------+
| jodatime_format('2020-06-25 15:58:51', 'yyyy-MM-dd HH:mm:ss') |
+-------------------------------------------------------------+
| 2020-06-25 15:58:51                                         |
+-------------------------------------------------------------+
```

Пример 3: Преобразовать дату `2020-06-25 15:58:51` в строку в формате `MM dd ee EE`.

```
MySQL > select jodatime_format('2020-06-25 15:58:51', 'MM dd ee EE');
+-------------------------------------------------------+
| jodatime_format('2020-06-25 15:58:51', 'MM dd ee EE') |
+-------------------------------------------------------+
| 06 25 04 Thu                                          |
+-------------------------------------------------------+
```

Пример 4: Преобразовать дату `2020-06-25 15:58:51` в строку в формате `MMM dd ee EEE`.

```
MySQL > select jodatime_format('2020-06-25 15:58:51', 'MMM dd ee EEE');
+---------------------------------------------------------+
| jodatime_format('2020-06-25 15:58:51', 'MMM dd ee EEE') |
+---------------------------------------------------------+
| Jun 25 04 Thu                                           |
+---------------------------------------------------------+
```

Пример 5: Преобразовать дату `2020-06-25 15:58:51` в строку в формате `MMMM dd ee EEEE`.

```
MySQL > select jodatime_format('2020-06-25 15:58:51', 'MMMM dd ee EEEE');
+-----------------------------------------------------------+
| jodatime_format('2020-06-25 15:58:51', 'MMMM dd ee EEEE') |
+-----------------------------------------------------------+
| June 25 04 Thursday                                       |
+-----------------------------------------------------------+
```

Пример 6: Преобразовать дату `2023-06-25 12:00:00` в строку в формате `KK:mm:ss a`.

```
MySQL > select jodatime_format('2023-06-25 12:00:00', 'KK:mm:ss a');
+-------------------------------------------------------+
| jodatime_format('2023-06-25 12:00:00', 'KK:mm:ss a') |
+-------------------------------------------------------+
| 00:00:00 PM                                           |
+-------------------------------------------------------+
```

Пример 7: Преобразовать дату `2023-06-25 12:00:00` в строку в формате `hh:mm:ss a`.

```
MySQL > select jodatime_format('2023-06-25 12:00:00', 'hh:mm:ss a');
+-------------------------------------------------------+
| jodatime_format('2023-06-25 12:00:00', 'hh:mm:ss a') |
+-------------------------------------------------------+
| 12:00:00 PM                                           |
+-------------------------------------------------------+
```

Пример 8: Преобразовать дату `2023-06-25 00:00:00` в строку в формате `yyyyMMdd ''selena''`.

```
MySQL > select jodatime_format('2023-06-25 00:00:00', 'yyyyMMdd
''selena''');
+-------------------------------------------------------------------+
| jodatime_format('2023-06-25 00:00:00', 'yyyyMMdd \'selena\'') |
+-------------------------------------------------------------------+
| 20230625 selena                                                   |
+-------------------------------------------------------------------+
```

Ключевые слова

JODATIME_FORMAT, JODA, FORMAT

Last_day

Возвращает последний день входного выражения DATE или DATETIME на основе указанной части даты. Например, `last_day('2023-05-10', 'month')` возвращает последний день месяца, в котором находится '2023-05-10'.

Если часть даты не указана, эта функция возвращает последний день месяца для заданной даты.

Эта функция поддерживается с версии 3.1.

Синтаксис

```
DATE last_day(DATETIME|DATE date_expr[, VARCHAR unit])
```

Параметры

- `date_expr`: выражение DATE или DATETIME, обязательный параметр.

- `unit`: часть даты, необязательный параметр. Допустимые значения: `month`, `quarter` и `year`, по умолчанию `month`. Если `unit` недопустим, возвращается ошибка.

Возвращаемое значение

Возвращает значение типа DATE.

Примеры

```
MySQL > select last_day('2023-05-10', 'month');
+-----------------------------------+
| last_day('2023-05-10', 'month')  |
+-----------------------------------+
| 2023-05-31                        |
+-----------------------------------+

MySQL > select last_day('2023-05-10');
+------------------------+
| last_day('2023-05-10') |
+------------------------+
| 2023-05-31             |
+------------------------+

MySQL > select last_day('2023-05-10', 'quarter');
+------------------------------------+
| last_day('2023-05-10', 'quarter') |
+------------------------------------+
| 2023-06-30                         |
+------------------------------------+

MySQL > select last_day('2023-05-10', 'year');
+----------------------------------------+
| last_day('2023-05-10', 'year')        |
+----------------------------------------+
| 2023-12-31                             |
+----------------------------------------+
```

Ключевые слова

LAST_DAY, LAST

Makedate

Создает и возвращает дату на основе заданных значений года и дня года.

Эта функция поддерживается с версии 3.1.

Синтаксис

```
DATE makedate(INT year, INT dayOfYear);
```

Параметры

- `year`: диапазон от 0 до 9999. Если этот диапазон превышен, возвращается NULL. Поддерживаемый тип данных - INT.
- `dayOfYear`: день года. Поддерживаемый тип данных - INT. Для поддержания той же семантики, что и функция dayofyear, если это число больше 366 или равно 366 в невисокосном году, это не является днем года.

Возвращаемое значение

Возвращает дату на dayOfYear-й день заданного года.

- `year` должен быть в диапазоне [0,9999]. В противном случае возвращается NULL.
- `dayOfYear` должен быть между 1 и количеством дней в текущем году (365 в обычные годы, 366 в високосные). В противном случае возвращается NULL.
- Результат также равен NULL, если любой входной параметр равен NULL.

Примеры

```
mysql> select makedate(2023,0);
+-------------------+
| makedate(2023, 0) |
+-------------------+
| NULL              |
+-------------------+

mysql> select makedate(2023,32);
+--------------------+
| makedate(2023, 32) |
+--------------------+
| 2023-02-01         |
+--------------------+

mysql> select makedate(2023,365);
+---------------------+
| makedate(2023, 365) |
+---------------------+
| 2023-12-31          |
+---------------------+

mysql> select makedate(2023,366);
+---------------------+
| makedate(2023, 366) |
+---------------------+
| NULL                |
+---------------------+

mysql> select makedate(9999,365);
+---------------------+
| makedate(9999, 365) |
+---------------------+
| 9999-12-31          |
+---------------------+

mysql> select makedate(10000,1);
+--------------------+
| makedate(10000, 1) |
+--------------------+
| NULL               |
+--------------------+
```

Ключевые слова

MAKEDATE,MAKE

Microseconds_add

Добавляет временной интервал к значению даты. Временной интервал указывается в микросекундах.

Синтаксис

```
DATETIME microseconds_add(DATETIME expr1,INT expr2);
```

Параметры

`expr1`: временное выражение. Должно быть типа DATETIME.

`expr2`: временной интервал, который вы хотите добавить, в микросекундах. Должен быть типа INT.

Возвращаемое значение

Возвращает значение типа DATETIME. Если входное значение имеет тип DATE, части часов, минут и секунд обрабатываются как `00:00:00`.

Примеры

```
select microseconds_add('2010-11-30 23:50:50', 2);
+---------------------------------------------+
| microseconds_add('2010-11-30 23:50:50', 2) |
+---------------------------------------------+
| 2010-11-30 23:50:50.000002                 |
+---------------------------------------------+
1 row in set (0.00 sec)

select microseconds_add('2010-11-30', 2);
+-----------------------------------+
| microseconds_add('2010-11-30', 2) |
+-----------------------------------+
| 2010-11-30 00:00:00.000002        |
+-----------------------------------+
```

Microseconds_sub

Вычитает временной интервал из значения даты. Временной интервал указывается в микросекундах.

Синтаксис

```
DATETIME microseconds_sub(DATETIME expr1,INT expr2);
```

Параметры

`expr1`: временное выражение. Должно быть типа DATETIME.

`expr2`: временной интервал, который вы хотите вычесть, в микросекундах. Должен быть типа INT.

Возвращаемое значение

Возвращает значение типа DATETIME. Если входное значение имеет тип DATE, части часов, минут и секунд обрабатываются как `00:00:00`.

Примеры

```
select microseconds_sub('2010-11-30 23:50:50', 2);
+---------------------------------------------+
| microseconds_sub('2010-11-30 23:50:50', 2) |
+---------------------------------------------+
| 2010-11-30 23:50:49.999998                  |
+---------------------------------------------+

select microseconds_sub('2010-11-30', 2);
+-----------------------------------+
| microseconds_sub('2010-11-30', 2) |
+-----------------------------------+
| 2010-11-29 23:59:59.999998        |
+-----------------------------------+
```

Milliseconds_diff

Возвращает разницу во времени между датой начала и датой окончания в миллисекундах.

Эта функция поддерживается с версии 3.2.4.

Синтаксис

```
BIGINT milliseconds_diff(DATETIME expr1, DATETIME expr2);
```

Параметры

- `expr1`: время окончания. Должно быть типа DATETIME. Если `expr1` раньше `expr2`, возвращается отрицательное значение.
- `expr2`: время начала. Должно быть типа DATETIME.

Возвращаемое значение

Возвращает значение типа BIGINT. Если входная дата не существует, возвращается NULL.

Примеры

```
select milliseconds_diff('2024-01-25 21:00:00.423','2024-01-25
21:00:00.123');
+-----------------------------------------------------------------------------+
| milliseconds_diff('2024-01-25 21:00:00.423', '2024-01-25 21:00:00.123') |
+-----------------------------------------------------------------------------+
|                                                                         300 |
+-----------------------------------------------------------------------------+

select milliseconds_diff('2024-01-25 21:00:01', '2024-01-25 21:00:00');
+-------------------------------------------------------------------+
| milliseconds_diff('2024-01-25 21:00:01', '2024-01-25 21:00:00') |
+-------------------------------------------------------------------+
|                                                              1000 |
+-------------------------------------------------------------------+

select milliseconds_diff('2024-01-25 00:00:01', '2024-01-25');
+-------------------------------------------------------+
| milliseconds_diff('2024-01-25 00:00:01', '2024-01-25') |
+-------------------------------------------------------+
|                                                   1000 |
+-------------------------------------------------------+
```

Minute

Возвращает минуту для заданной даты. Возвращаемое значение находится в диапазоне от 0 до 59.

Параметр `date` должен быть типа DATE или DATETIME.

Синтаксис

```
INT MINUTE(DATETIME|DATE date)
```

Примеры

```
MySQL > select minute('2018-12-31 23:59:59');
+-----------------------------+
|minute('2018-12-31 23:59:59')|
+-----------------------------+
|                           59 |
+-----------------------------+
```

Ключевые слова

MINUTE

Minutes_add

Добавляет указанное количество минут к дате с точностью до минуты.

Синтаксис

```
DATETIME minutes_add(DATETIME|DATE date, INT minutes);
```

Параметры

- `date`: время начала. Должно быть типа DATETIME или DATE.
- `minutes`: количество минут для добавления. Должно быть типа INT, может быть больше, равно или меньше нуля.

Возвращаемое значение

Возвращает значение типа DATETIME.

Возвращает NULL, если `date` или `minutes` равны NULL.

Примеры

```
select minutes_add('2022-01-01 01:01:01', 2);
+----------------------------------------+
| minutes_add('2022-01-01 01:01:01', 2) |
+----------------------------------------+
| 2022-01-01 01:03:01                    |
+----------------------------------------+

select minutes_add('2022-01-01 01:01:01', -1);
+-----------------------------------------+
| minutes_add('2022-01-01 01:01:01', -1) |
+-----------------------------------------+
| 2022-01-01 01:00:01                     |
+-----------------------------------------+

select minutes_add('2022-01-01', 1);
+------------------------------+
| minutes_add('2022-01-01', 1) |
+------------------------------+
| 2022-01-01 00:01:00          |
+------------------------------+
```

Minutes_diff

Возвращает разницу в минутах между двумя выражениями даты (`expr1` – `expr2`) с точностью до минуты.

Синтаксис

```
BIGINT minutes_diff(DATETIME expr1,DATETIME expr2);
```

Параметры

- `expr1`: время окончания. Должно быть типа DATETIME.

- `expr2`: время начала. Должно быть типа DATETIME.

Возвращаемое значение

Возвращает значение типа BIGINT.

Если дата не существует, например 2022-02-29, возвращается NULL.

Примеры

```
select minutes_diff('2010-11-30 23:59:59', '2010-11-30 20:58:59');
+-----------------------------------------------------------+
| minutes_diff('2010-11-30 23:59:59', '2010-11-30 20:58:59') |
+-----------------------------------------------------------+
|                                                       181 |
+-----------------------------------------------------------+
```

Minutes_sub

Вычитает указанное количество минут из даты с точностью до минуты.

Синтаксис

```
DATETIME minutes_sub(DATETIME|DATE date, INT minutes);
```

Параметры

- `date`: базовая дата. Должна быть типа DATETIME или DATE.
- `minutes`: количество минут для вычитания. Должно быть типа INT, может быть больше, равно или меньше нуля.

Возвращаемое значение

Возвращает значение типа DATETIME.

Возвращает NULL, если `date` или `minutes` равны NULL.

Примеры

```
select minutes_sub('2022-01-01 01:03:01', 2);
+----------------------------------------+
| minutes_sub('2022-01-01 01:03:01', 2) |
+----------------------------------------+
| 2022-01-01 01:01:01                    |
+----------------------------------------+
select minutes_sub('2022-01-01 01:01:01', -1);
+-----------------------------------------+
| minutes_sub('2022-01-01 01:01:01', -1) |
+-----------------------------------------+
| 2022-01-01 01:02:01                     |
+-----------------------------------------+
select minutes_sub('2022-01-01', 1);
+-------------------------------+
| minutes_sub('2022-01-01', 1) |
+-------------------------------+
| 2021-12-31 23:59:00           |
+-------------------------------+
```

Month

Возвращает месяц для заданной даты. Возвращаемое значение находится в диапазоне от 1 до 12.

Параметр `date` должен быть типа DATE или DATETIME.

Синтаксис

```
INT MONTH(DATETIME date)
```

Примеры

```
MySQL > select month('1987-01-01');
+-----------------------------+
|month('1987-01-01 00:00:00') |
+-----------------------------+
|                           1 |
+-----------------------------+
```

Ключевые слова

MONTH

Monthname

Возвращает название месяца для заданной даты.

Параметр `date` должен быть типа DATE или DATETIME.

Синтаксис

```
VARCHAR MONTHNAME(date)
```

Примеры

```
MySQL > select monthname('2008-02-03 00:00:00');
+----------------------------------+
| monthname('2008-02-03 00:00:00') |
+----------------------------------+
| February                         |
+----------------------------------+
```

Ключевые слова

MONTHNAME, monthname

Months_add

Добавляет указанное количество месяцев к дате с точностью до месяца.

Функция add_months предоставляет аналогичную функциональность.

Синтаксис

```
DATETIME months_add(DATETIME expr1, INT expr2);
```

Параметры

- `expr1`: время начала. Должно быть типа DATETIME или DATE.

- `expr2`: количество месяцев для добавления. Должно быть типа INT. Может быть больше, равно или меньше нуля. Отрицательное значение вычитает месяцы из `date`.

Возвращаемое значение

Возвращает значение типа DATETIME.

Примеры

```
select months_add('2019-08-01 13:21:03', 8);
+---------------------------------------+
| months_add('2019-08-01 13:21:03', 8) |
+---------------------------------------+
| 2020-04-01 13:21:03                   |
+---------------------------------------+

select months_add('2019-08-01', 8);
+-----------------------------+
| months_add('2019-08-01', 8) |
+-----------------------------+
| 2020-04-01 00:00:00         |
+-----------------------------+

select months_add('2019-08-01 13:21:03', -8);
+----------------------------------------+
| months_add('2019-08-01 13:21:03', -8) |
+----------------------------------------+
| 2018-12-01 13:21:03                    |
+----------------------------------------+

select months_add('2019-02-28 13:21:03', 1);
+---------------------------------------+
| months_add('2019-02-28 13:21:03', 1) |
+---------------------------------------+
| 2019-03-28 13:21:03                   |
+---------------------------------------+

select months_add('2019-01-30 13:21:03', 1);
+---------------------------------------+
| months_add('2019-01-30 13:21:03', 1) |
+---------------------------------------+
| 2019-02-28 13:21:03                   |
+---------------------------------------+
```

Months_diff

Возвращает разницу в месяцах между двумя выражениями даты (`expr1` – `expr2`) с точностью до месяца.

Синтаксис

```
BIGINT months_diff(DATETIME expr1,DATETIME expr2);
```

Параметры

- `expr1`: конечное время. Должно быть типа DATETIME или DATE.
- `expr2`: начальное время. Должно быть типа DATETIME или DATE.

Возвращаемое значение

Возвращает значение типа BIGINT.

Возвращается NULL, если дата не существует, например, 2022-02-29.

Примеры

```
select months_diff('2010-11-30 23:59:59', '2010-1-1 23:59:59');
+-----------------------------------------------------------+
| months_diff('2010-11-30 23:59:59', '2010-1-1 23:59:59') |
+-----------------------------------------------------------+
|                                                        10 |
+-----------------------------------------------------------+

select months_diff('2010-11-30', '2010-1-1');
+---------------------------------------+
| months_diff('2010-11-30', '2010-1-1') |
+---------------------------------------+
|                                    10 |
+---------------------------------------+
```

Months_sub

Вычитает указанное количество месяцев из даты.

Синтаксис

```
DATETIME months_sub(DATETIME expr1, INT expr2);
```

Параметры

- `expr1`: указанная дата. Должно быть типа DATETIME или DATE.
- `expr2`: количество месяцев для вычитания. Должно быть типа INT.

Возвращаемое значение

Возвращает значение типа DATETIME.

Примеры

```
select months_sub('2022-02-28 15:04:05', 1);
+--------------------------------------+
| months_sub('2022-02-28 15:04:05', 1) |
+--------------------------------------+
| 2022-01-28 15:04:05                  |
+--------------------------------------+

select months_sub('2022-02-28', 1);
+-----------------------------+
| months_sub('2022-02-28', 1) |
+-----------------------------+
| 2022-01-28 00:00:00         |
+-----------------------------+
```

Next_day

Возвращает дату первого указанного дня недели (DOW), который наступает после входной даты (DATE или DATETIME). Например, `next_day('2023-04-06', 'Monday')` возвращает дату следующего понедельника после '2023-04-06'.

Эта функция поддерживается с версии 3.1. Она противоположна функции previous_day.

Синтаксис

```
DATE next_day(DATETIME|DATE date_expr, VARCHAR dow)
```

Параметры

- `date_expr`: входная дата. Должно быть допустимым выражением типа DATE или DATETIME.

- `dow`: день недели. Допустимые значения включают ряд сокращений, которые чувствительны к регистру:

DOW_FULL	DOW_2	DOW_3
Sunday	Su	Sun
Monday	Mo	Mon
Tuesday	Tu	Tue
Wednesday	We	Wed
Thursday	Th	Thu
Friday	Fr	Fri
Saturday	Sa	Sat

Возвращаемое значение

Возвращает значение типа DATE.

Любое недопустимое значение `dow` вызовет ошибку. Параметр `dow` чувствителен к регистру.

Возвращает NULL, если передана недопустимая дата или аргумент NULL.

Примеры

```
-- Возвращает дату следующего понедельника после 2023-04-06. 2023-04-06 —
это четверг, и дата следующего понедельника — 2023-04-10.

MySQL > select next_day('2023-04-06', 'Monday');
+----------------------------------+
| next_day('2023-04-06', 'Monday') |
+----------------------------------+
| 2023-04-10                       |
+----------------------------------+

MySQL > select next_day('2023-04-06', 'Tue');
+-------------------------------+
| next_day('2023-04-06', 'Tue') |
+-------------------------------+
| 2023-04-11                    |
+-------------------------------+

MySQL > select next_day('2023-04-06 20:13:14', 'Fr');
+---------------------------------------+
| next_day('2023-04-06 20:13:14', 'Fr') |
+---------------------------------------+
| 2023-04-07                            |
+---------------------------------------+
```

Ключевые слова

NEXT_DAY, NEXT

Now, current_timestamp, localtime, localtimestamp

Возвращает текущую дату и время.

Начиная с версии 3.1.6, эта функция может принимать параметр точности (`p`), который представляет количество цифр для сохранения после секунд. Максимум можно сохранить 6 цифр (точность до микросекунд). Цифры после `p` дополняются нулями. Если `p` не указан, возвращается значение DATETIME с точностью до секунды.

Следующий пример показывает возвращаемые значения даты и времени при указании различной точности.

```
mysql > select now(),now(1),now(2),now(3),now(4),now(5),now(6)\G
*************************** 1. row ***************************
 now(): 2023-12-08 13:46:45
now(1): 2023-12-08 13:46:45.100000
now(2): 2023-12-08 13:46:45.110000
now(3): 2023-12-08 13:46:45.115000
now(4): 2023-12-08 13:46:45.115800
now(5): 2023-12-08 13:46:45.115840
now(6): 2023-12-08 13:46:45.115843
```

Эта функция может возвращать разные результаты для разных часовых поясов. Подробнее см. в разделе Настройка часового пояса.

Синтаксис

```
DATETIME NOW()
DATETIME NOW(INT p)
```

Параметры

`p`: необязательный параметр, указанная точность, то есть количество цифр для сохранения после секунд. Должно быть значение типа INT в диапазоне [1,6]. `select now(0)` эквивалентно `select now()`.

Возвращаемое значение

- Если `p` не указан, функция возвращает значение DATETIME с точностью до секунды.
- Если `p` указан, функция возвращает значение даты и времени с заданной точностью.

Примеры

```
MySQL > select now();
+---------------------+
| now()               |
+---------------------+
| 2019-05-27 15:58:25 |
+---------------------+

MySQL > select now(),now(1),now(2),now(3),now(4),now(5),now(6)\G
*************************** 1. row ***************************
 now(): 2023-12-08 13:46:45
now(1): 2023-12-08 13:46:45.100000
now(2): 2023-12-08 13:46:45.110000
now(3): 2023-12-08 13:46:45.115000
now(4): 2023-12-08 13:46:45.115800
now(5): 2023-12-08 13:46:45.115840
now(6): 2023-12-08 13:46:45.115843
```

Ключевые слова

NOW, now

Previous_day

Возвращает дату первого указанного дня недели (DOW), который наступает до входной даты (DATE или DATETIME). Например, `previous_day('2023-04-06', 'Monday')` возвращает дату предыдущего понедельника до '2023-04-06'.

Эта функция поддерживается с версии 3.1. Она противоположна функции next_day.

Синтаксис

```
DATE previous_day(DATETIME|DATE date_expr, VARCHAR dow)
```

Параметры

- `date_expr`: входная дата. Должно быть допустимым выражением типа DATE или DATETIME.

- `dow`: день недели. Допустимые значения включают ряд сокращений, которые чувствительны к регистру:

DOW_FULL	DOW_2	DOW_3
Sunday	Su	Sun
Monday	Mo	Mon
Tuesday	Tu	Tue
Wednesday	We	Wed
Thursday	Th	Thu
Friday	Fr	Fri
Saturday	Sa	Sat

Возвращаемое значение

Возвращает значение типа DATE.

Любое недопустимое значение `dow` вызовет ошибку. Параметр `dow` чувствителен к регистру.

Возвращает NULL, если передана недопустимая дата или аргумент NULL.

Примеры

```
-- Возвращает дату предыдущего понедельника до 2023-04-06. 2023-04-06 — это
четверг, и дата предыдущего понедельника — 2023-04-03.

MySQL > select previous_day('2023-04-06', 'Monday');
+--------------------------------------+
| previous_day('2023-04-06', 'Monday') |
+--------------------------------------+
| 2023-04-03                           |
+--------------------------------------+

MySQL > select previous_day('2023-04-06', 'Tue');
+-----------------------------------+
| previous_day('2023-04-06', 'Tue') |
+-----------------------------------+
| 2023-04-04                        |
+-----------------------------------+

MySQL > select previous_day('2023-04-06 20:13:14', 'Fr');
+--------------------------------------------+
| previous_day('2023-04-06 20:13:14', 'Fr') |
+--------------------------------------------+
| 2023-03-31                                 |
+--------------------------------------------+
```

Ключевые слова

PREVIOUS_DAY, PREVIOUS, previousday

Quarter

Возвращает квартал из даты в диапазоне от 1 до 4.

Синтаксис

```
INT quarter(DATETIME|DATE date);
```

Параметры

`date`: должно быть типа DATETIME или DATE.

Возвращаемое значение

Возвращает значение типа INT.

Возвращает NULL в следующих случаях:

- Дата не является допустимым значением DATETIME или DATE.
- Входное значение пустое.
- Дата не существует, например, 2022-02-29.

Примеры

Пример 1: Возвращает квартал из значения DATETIME.

```
SELECT QUARTER("2022-10-09 15:59:33");
+--------------------------------+
| quarter('2022-10-09 15:59:33') |
+--------------------------------+
|                              4 |
+--------------------------------+
```

Пример 2: Возвращает квартал из значения DATE.

```
SELECT QUARTER("2022-10-09");
+-----------------------+
| quarter('2022-10-09') |
+-----------------------+
|                     4 |
+-----------------------+
```

Пример 3: Возвращает квартал, соответствующий текущему времени или дате.

```
SELECT QUARTER(NOW());
+----------------+
| quarter(now()) |
+----------------+
|              4 |
+----------------+

SELECT QUARTER(CURDATE());
+--------------------+
| quarter(curdate()) |
+--------------------+
|                  4 |
+--------------------+
```

Second

Возвращает секунды из заданной даты. Возвращаемое значение находится в диапазоне от 0 до 59.

Параметр `date` должен быть типа DATE или DATETIME.

Синтаксис

```
INT SECOND(DATETIME date)
```

Примеры

```
MySQL > select second('2018-12-31 23:59:59');
+-----------------------------+
|second('2018-12-31 23:59:59')|
+-----------------------------+
|                           59 |
+-----------------------------+
```

Ключевые слова

SECOND

Seconds_add

Описание

Синтаксис

```
DATETIME SECONDS_ADD(DATETIME expr1,INT expr2)
```

Добавляет указанный временной интервал к дате. Единица измерения — секунды.

Параметр expr1 — допустимое выражение datetime.

Параметр expr2 — количество секунд для добавления.

Примеры

```
select seconds_add('2010-11-30 23:50:50', 2);
+---------------------------------------+
| seconds_add('2010-11-30 23:50:50', 2) |
+---------------------------------------+
| 2010-11-30 23:50:52                   |
+---------------------------------------+

select seconds_add('2010-11-30', 2);
+------------------------------+
| seconds_add('2010-11-30', 2) |
+------------------------------+
| 2010-11-30 00:00:02          |
+------------------------------+
```

Ключевые слова

SECONDS_ADD,ADD

Seconds_diff

Возвращает разницу в секундах между двумя выражениями даты (`expr1` – `expr2`) с точностью до секунды.

Синтаксис

```
BIGINT seconds_diff(DATETIME expr1,DATETIME expr2);
```

Параметры

- `expr1`: конечное время. Должно быть типа DATETIME.
- `expr2`: начальное время. Должно быть типа DATETIME.

Возвращаемое значение

Возвращает значение типа BIGINT.

Возвращается NULL, если дата не существует, например, 2022-02-29.

Примеры

```
select seconds_diff('2010-11-30 23:59:59', '2010-11-30 20:59:59');
+------------------------------------------------------------+
| seconds_diff('2010-11-30 23:59:59', '2010-11-30 20:59:59') |
+------------------------------------------------------------+
|                                                      10800 |
+------------------------------------------------------------+
```

Seconds_sub

Вычитает временной интервал из значения даты. Временной интервал указывается в секундах.

Синтаксис

```
DATETIME seconds_sub(DATETIME|DATE date, INT seconds);
```

Параметры

`date`: выражение времени. Должно быть типа DATETIME.

`seconds`: временной интервал для вычитания, в секундах. Должен быть типа INT.

Возвращаемое значение

Возвращает значение типа DATETIME.

Если входное значение имеет тип DATE, части часов, минут и секунд обрабатываются как `00:00:00`.

Примеры

```
select seconds_sub('2022-01-01 01:01:01', 2);
+----------------------------------------+
| seconds_sub('2022-01-01 01:01:01', 2) |
+----------------------------------------+
| 2022-01-01 01:00:59                    |
+----------------------------------------+

select seconds_sub('2022-01-01 01:01:01', -1);
+-----------------------------------------+
| seconds_sub('2022-01-01 01:01:01', -1) |
+-----------------------------------------+
| 2022-01-01 01:01:02                     |
+-----------------------------------------+

select seconds_sub('2022-01-01', 1);
+------------------------------+
| seconds_sub('2022-01-01', 1) |
+------------------------------+
| 2021-12-31 23:59:59          |
+------------------------------+
```

Str2date

Преобразует строку в значение DATE в соответствии с указанным форматом. Если преобразование не удалось, возвращается NULL.

Формат должен соответствовать описанному в date_format.

Эта функция эквивалентна str_to_date, но имеет другой тип возвращаемого значения.

Синтаксис

```
DATE str2date(VARCHAR str, VARCHAR format);
```

Параметры

`str`: выражение времени для преобразования. Должно быть типа VARCHAR.

`format`: формат, используемый для возврата значения. Поддерживаемые форматы см. в date_format.

Возвращаемое значение

Возвращает значение типа DATE.

Возвращается NULL, если `str` или `format` равен NULL.

Примеры

```
select str2date('2010-11-30 23:59:59', '%Y-%m-%d %H:%i:%s');
+-------------------------------------------------------+
| str2date('2010-11-30 23:59:59', '%Y-%m-%d %H:%i:%s') |
+-------------------------------------------------------+
| 2010-11-30                                            |
+-------------------------------------------------------+
1 row in set (0.01 sec)
```

Str_to_date

Преобразует строку в значение DATETIME в соответствии с указанным форматом. Если преобразование не удалось, возвращается NULL.

Формат должен соответствовать описанному в date_format.

Эта функция является обратной к date_format.

Синтаксис

```
DATETIME STR_TO_DATE(VARCHAR str, VARCHAR format)
```

Параметры

`str`: выражение времени для преобразования. Должно быть типа VARCHAR.

`format`: формат, используемый для возврата значения. Поддерживаемые форматы см. в date_format.

Возвращаемое значение

Возвращает значение типа DATETIME. Если `format` указывает только дату, возвращается значение DATE.

Возвращается NULL, если `str` или `format` равен NULL.

Примеры

Пример 1: Преобразование входных данных в значение DATETIME.

```
MySQL > select str_to_date('2014-12-21 12:34:56', '%Y-%m-%d %H:%i:%s');
+-----------------------------------------------------------+
| str_to_date('2014-12-21 12:34:56', '%Y-%m-%d %H:%i:%s') |
+-----------------------------------------------------------+
| 2014-12-21 12:34:56                                       |
+-----------------------------------------------------------+
```

Пример 2: Преобразование входных данных в значение DATE.

```
MySQL > select str_to_date('2014-12-21 12:34:56', '%Y-%m-%d');
+-------------------------------------------------------------+
| str_to_date('2014-12-21 12:34:56', '%Y-%m-%d')              |
+-------------------------------------------------------------+
| 2014-12-21                                                  |
+-------------------------------------------------------------+
```

Пример 3: Преобразование входных данных "200442 Monday" в значение DATE.

```
MySQL > select str_to_date('200442 Monday', '%X%V %W');
+-----------------------------------------+
| str_to_date('200442 Monday', '%X%V %W') |
+-----------------------------------------+
| 2004-10-18                              |
+-----------------------------------------+
```

Ключевые слова

STR_TO_DATE,STR,TO,DATE

Str_to_jodatime

Преобразует строку в формате Joda в значение DATETIME в указанном формате Joda DateTime, например `yyyy-MM-dd HH:mm:ss`.

Синтаксис

```
DATETIME str_to_jodatime(VARCHAR str, VARCHAR format)
```

Параметры

- `str`: выражение времени для преобразования. Должно быть типа VARCHAR.
- `format`: формат Joda DateTime возвращаемого значения DATETIME. Информацию о доступных форматах см. в Joda DateTime.

Возвращаемое значение

- Если разбор входной строки успешен, возвращается значение DATETIME.
- Если разбор входной строки не удался, возвращается `NULL`.

Примеры

Пример 1: Преобразование строки `2014-12-21 12:34:56` в значение DATETIME в формате `yyyy-MM-dd HH:mm:ss`.

```
MySQL > select str_to_jodatime('2014-12-21 12:34:56', 'yyyy-MM-dd
HH:mm:ss');
+-----------------------------------------------------------------+
| str_to_jodatime('2014-12-21 12:34:56', 'yyyy-MM-dd HH:mm:ss') |
+-----------------------------------------------------------------+
| 2014-12-21 12:34:56                                             |
+-----------------------------------------------------------------+
```

Пример 2: Преобразование строки `21/December/23 12:34:56` с текстовым названием месяца в значение DATETIME в формате `dd/MMMM/yy HH:mm:ss`.

```
MySQL > select str_to_jodatime('21/December/23 12:34:56', 'dd/MMMM/yy
HH:mm:ss');
+-----------------------------------------------------------------+
| str_to_jodatime('21/December/23 12:34:56', 'dd/MMMM/yy HH:mm:ss') |
+-----------------------------------------------------------------+
| 2023-12-21 12:34:56                                              |
+-----------------------------------------------------------------+
```

Пример 3: Преобразование строки `21/December/23 12:34:56.123` с точностью до миллисекунд в значение DATETIME в формате `dd/MMMM/yy HH:mm:ss.SSS`.

```
MySQL > select str_to_jodatime('21/December/23 12:34:56.123', 'dd/MMMM/yy
HH:mm:ss.SSS');
+-------------------------------------------------------------------------
+
| str_to_jodatime('21/December/23 12:34:56.123', 'dd/MMMM/yy HH:mm:ss.SSS')
|
+-------------------------------------------------------------------------
+
| 2023-12-21 12:34:56.123000
|
+-------------------------------------------------------------------------
+
```

Ключевые слова

STR_TO_JODATIME, DATETIME

Time_format

Форматирует значения типа TIME в указанном формате.

Синтаксис

```
VARCHAR TIME_FORMAT(TIME time, VARCHAR format)
```

Параметры

- `time` (обязательный): значение времени (типа TIME) для форматирования.
- `format` (обязательный): используемый формат. Допустимые значения:

```
%f      Микросекунды (000000 до 999999)
%H      Час (00 до 23)
%h      Час (00 до 12)
%i      Минуты (00 до 59)
%p      AM или PM
%S      Секунды (00 до 59)
%s      Секунды (00 до 59)
```

Примеры

```
mysql> SELECT TIME_FORMAT("19:30:10", "%h %i %s %p");
+----------------------------------------+
| time_format('19:30:10', '%h %i %s %p') |
+----------------------------------------+
| 12 00 00 AM                            |
+----------------------------------------+
1 row in set (0.01 sec)
```

Time_slice

Преобразует заданное время в начало или конец временного интервала на основе указанной гранулярности времени.

Эта функция поддерживается с версии 2.3.

В версии 2.5 добавлена поддержка преобразования заданного времени в конец временного интервала.

Синтаксис

```
DATETIME time_slice(DATETIME dt, INTERVAL N type[, boundary])
```

Параметры

- `dt`: время для преобразования, DATETIME.
- `INTERVAL N type`: гранулярность времени, например, `interval 5 second`.
  - `N` — длина временного интервала. Должно быть значение типа INT.

- `type` — единица измерения, которая может быть YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, SECOND, MILLISECOND (с версии 3.1.7) и MICROSECOND (с версии 3.1.7).

- `boundary`: необязательный параметр. Используется для указания, возвращать ли начало (`FLOOR`) или конец (`CEIL`) временного интервала. Допустимые значения: FLOOR, CEIL. Если этот параметр не указан, по умолчанию используется `FLOOR`. Этот параметр поддерживается с версии 2.5.

Возвращаемое значение

Возвращает значение типа DATETIME.

Примечания по использованию

Временной интервал начинается с `0001-01-01 00:00:00` н.э.

Примеры

Следующие примеры основаны на таблице `test_all_type_select`.

```
select * from test_all_type_select order by id_int;
+------------+---------------------+--------+
| id_date    | id_datetime         | id_int |
+------------+---------------------+--------+
| 2052-12-26 | 1691-12-23 04:01:09 |      0 |
| 2168-08-05 | 2169-12-18 15:44:31 |      1 |
| 1737-02-06 | 1840-11-23 13:09:50 |      2 |
| 2245-10-01 | 1751-03-21 00:19:04 |      3 |
| 1889-10-27 | 1861-09-12 13:28:18 |      4 |
+------------+---------------------+--------+
5 rows in set (0.06 sec)
```

Пример 1: Преобразование заданного значения DATETIME в начало 5-секундного временного интервала без указания параметра `boundary`.

```
select time_slice(id_datetime, interval 5 second)
from test_all_type_select
order by id_int;
+-----------------------------------------------------+
| time_slice(id_datetime, INTERVAL 5 second, floor) |
+-----------------------------------------------------+
| 1691-12-23 04:01:05                                 |
| 2169-12-18 15:44:30                                 |
| 1840-11-23 13:09:50                                 |
| 1751-03-21 00:19:00                                 |
| 1861-09-12 13:28:15                                 |
+-----------------------------------------------------+
5 rows in set (0.16 sec)
```

Пример 2: Преобразование заданного значения DATETIME в начало 5-дневного временного интервала с `boundary`, установленным в `FLOOR`.

```
select time_slice(id_datetime, interval 5 day, FLOOR)
from test_all_type_select
order by id_int;
+-------------------------------------------------+
| time_slice(id_datetime, INTERVAL 5 day, floor) |
+-------------------------------------------------+
| 1691-12-22 00:00:00                             |
| 2169-12-16 00:00:00                             |
| 1840-11-21 00:00:00                             |
| 1751-03-18 00:00:00                             |
| 1861-09-12 00:00:00                             |
+-------------------------------------------------+
5 rows in set (0.15 sec)
```

Пример 3: Преобразование заданного значения DATETIME в конец 5-дневного временного интервала.

```
select time_slice(id_datetime, interval 5 day, CEIL)
from test_all_type_select
order by id_int;
+-----------------------------------------------+
| time_slice(id_datetime, INTERVAL 5 day, ceil) |
+-----------------------------------------------+
| 1691-12-27 00:00:00                           |
| 2169-12-21 00:00:00                           |
| 1840-11-26 00:00:00                           |
| 1751-03-23 00:00:00                           |
| 1861-09-17 00:00:00                           |
+-----------------------------------------------+
```

Пример 4: Преобразование заданного значения DATETIME в конец 1-миллисекундного временного интервала.

```
select id_datetime, time_slice(id_datetime, interval 1 millisecond, CEIL)
from test_all_type_select
order by id_int;
+---------------------+-------------------------------------------------------
---+
| id_datetime         | time_slice(id_datetime, INTERVAL 1 millisecond,
ceil) |
+---------------------+-------------------------------------------------------
---+
| 1691-12-23 04:01:09 | 1691-12-23 04:01:09.001000
|
| 2169-12-18 15:44:31 | 2169-12-18 15:44:31.001000
|
| 1840-11-23 13:09:50 | 1840-11-23 13:09:50.001000
|
| 1751-03-21 00:19:04 | 1751-03-21 00:19:04.001000
|
| 1861-09-12 13:28:18 | 1861-09-12 13:28:18.001000
|
+---------------------+-------------------------------------------------------
---+
```

Пример 5: Преобразование заданного значения DATETIME в конец 1-микросекундного временного интервала.

```
select id_datetime, time_slice(id_datetime, interval 1 microsecond, CEIL)
from test_all_type_select
order by id_int;
+---------------------+-----------------------------------------------------
---+
| id_datetime         | time_slice(id_datetime, INTERVAL 1 microsecond,
ceil) |
+---------------------+-----------------------------------------------------
---+
| 1691-12-23 04:01:09 | 1691-12-23 04:01:09.000001
|
| 2169-12-18 15:44:31 | 2169-12-18 15:44:31.000001
|
| 1840-11-23 13:09:50 | 1840-11-23 13:09:50.000001
|
| 1751-03-21 00:19:04 | 1751-03-21 00:19:04.000001
|
| 1861-09-12 13:28:18 | 1861-09-12 13:28:18.000001
|
+---------------------+-----------------------------------------------------
---+
```

Time_to_sec

Преобразует значение времени в количество секунд. Для преобразования используется следующая формула:

Час x 3600 + Минута x 60 + Секунда

Синтаксис

```
BIGINT time_to_sec(TIME time)
```

Параметры

`time`: должен быть типа TIME.

Возвращаемое значение

Возвращает значение типа BIGINT. Если входные данные некорректны, возвращается NULL.

Примеры

```
select time_to_sec('12:13:14');
+-----------------------------+
| time_to_sec('12:13:14')     |
+-----------------------------+
|                       43994|
+-----------------------------+
```

Timediff

Возвращает разницу между двумя выражениями DATETIME.

Возвращаемое значение должно быть типа TIME.

Синтаксис

```
TIME TIMEDIFF(DATETIME expr1, DATETIME expr2)`
```

Примеры

```
MySQL > SELECT TIMEDIFF(now(),utc_timestamp());
+----------------------------------+
| timediff(now(), utc_timestamp()) |
+----------------------------------+
| 08:00:00                         |
+----------------------------------+

MySQL > SELECT TIMEDIFF('2019-07-11 16:59:30','2019-07-11 16:59:21');
+----------------------------------------------------------+
| timediff('2019-07-11 16:59:30', '2019-07-11 16:59:21') |
+----------------------------------------------------------+
| 00:00:09                                                 |
+----------------------------------------------------------+

MySQL > SELECT TIMEDIFF('2019-01-01 00:00:00', NULL);
+----------------------------------------+
| timediff('2019-01-01 00:00:00', NULL) |
+----------------------------------------+
| NULL                                   |
+----------------------------------------+
```

Keyword

TIMEDIFF

Timestamp

Возвращает значение DATETIME из выражения даты или даты-времени.

Синтаксис

```
DATETIME timestamp(DATETIME|DATE expr);
```

Параметры

`expr`: выражение времени, которое необходимо преобразовать. Должно быть типа DATETIME или DATE.

Возвращаемое значение

Возвращает значение DATETIME. Если входное время пустое или не существует, например `2021-02-29`, возвращается NULL.

Примеры

```
select timestamp("2019-05-27");
+-------------------------+
| timestamp('2019-05-27') |
+-------------------------+
| 2019-05-27 00:00:00     |
+-------------------------+
1 row in set (0.00 sec)
```

Timestampadd

Добавляет целочисленный интервал выражения к выражению даты или даты-времени `datetime_expr`.

Единица измерения интервала должна быть одной из следующих:

MILLISECOND (начиная с версии 3.2), SECOND, MINUTE, HOUR, DAY, WEEK, MONTH или YEAR.

Синтаксис

```
DATETIME TIMESTAMPADD(unit, interval, DATETIME datetime_expr)
```

Параметры

- `datetime_expr`: значение DATE или DATETIME, к которому необходимо добавить временной интервал.
- `interval`: целочисленное выражение, указывающее количество интервалов для добавления.
- `unit`: единица измерения добавляемого временного интервала. Поддерживаемые единицы: MILLISECOND (начиная с версии 3.2), SECOND, MINUTE, HOUR, DAY, WEEK, MONTH и YEAR.

Возвращаемое значение

Возвращает значение того же типа данных, что и `datetime_expr`.

Примеры

```
MySQL > SELECT TIMESTAMPADD(MINUTE,1,'2019-01-02');
+-------------------------------------------------+
| timestampadd(MINUTE, 1, '2019-01-02 00:00:00') |
+-------------------------------------------------+
| 2019-01-02 00:01:00                             |
+-------------------------------------------------+

MySQL > SELECT TIMESTAMPADD(WEEK,1,'2019-01-02');
+-------------------------------------------------+
| timestampadd(WEEK, 1, '2019-01-02 00:00:00') |
+-------------------------------------------------+
| 2019-01-09 00:00:00                             |
+-------------------------------------------------+

MySQL > SELECT TIMESTAMPADD(MILLISECOND,1,'2019-01-02');
+-------------------------------------------+
| timestampadd(MILLISECOND, 1, '2019-01-02') |
+-------------------------------------------+
| 2019-01-02 00:00:00.001000                |
+-------------------------------------------+
```

Keyword

TIMESTAMPADD

Timestampdiff

Возвращает интервал от `datetime_expr2` до `datetime_expr1`. `datetime_expr1` и `datetime_expr2` должны быть типа DATE или DATETIME.

Единица измерения для целочисленного результата и интервала должна быть одной из следующих:

MILLISECOND (начиная с версии 3.2), SECOND, MINUTE, HOUR, DAY, WEEK, MONTH или YEAR.

Синтаксис

```
BIGINT TIMESTAMPDIFF(unit,DATETIME datetime_expr1, DATETIME datetime_expr2)
```

Параметры

- `datetime_expr`: два выражения DATE или DATETIME, которые необходимо сравнить.
- `unit`: единица измерения разницы во времени. Поддерживаемые единицы: MILLISECOND (начиная с версии 3.2), SECOND, MINUTE, HOUR, DAY, WEEK, MONTH и YEAR.

Возвращаемое значение

Возвращает значение типа BIGINT.

Примеры

```
MySQL> SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
+-----------------------------------------------------------------------+
| timestampdiff(MONTH, '2003-02-01 00:00:00', '2003-05-01 00:00:00') |
+-----------------------------------------------------------------------+
|                                                                     3 |
+-----------------------------------------------------------------------+

MySQL> SELECT TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01');
+-----------------------------------------------------------------------+
| timestampdiff(YEAR, '2002-05-01 00:00:00', '2001-01-01 00:00:00') |
+-----------------------------------------------------------------------+
|                                                                    -1 |
+-----------------------------------------------------------------------+

MySQL> SELECT TIMESTAMPDIFF(MINUTE,'2003-02-01','2003-05-01 12:05:55');
+-----------------------------------------------------------------------+
| timestampdiff(MINUTE, '2003-02-01 00:00:00', '2003-05-01 12:05:55') |
+-----------------------------------------------------------------------+
|                                                                128885 |
+-----------------------------------------------------------------------+

MySQL > SELECT TIMESTAMPDIFF(MILLISECOND,'2003-02-01','2003-05-01');
+-----------------------------------------------------------+
| timestampdiff(MILLISECOND, '2003-02-01', '2003-05-01') |
+-----------------------------------------------------------+
|                                              7689600000 |
+-----------------------------------------------------------+
```

Keyword

TIMESTAMPDIFF

To_date

Преобразует значение DATETIME в дату.

Синтаксис

```
DATE TO_DATE(DATETIME datetime)
```

Примеры

```
MySQL > select to_date("2020-02-02 00:00:00");
+--------------------------------+
| to_date('2020-02-02 00:00:00') |
+--------------------------------+
| 2020-02-02                     |
+--------------------------------+
```

Keyword

TO_DATE

To_datetime

Преобразует Unix-метку времени в значение типа DATETIME **на основе текущих настроек часового пояса**.

Подробные инструкции по настройке часового пояса см. в разделе Настройка часового пояса.

Если вы хотите преобразовать Unix-метку времени в значение типа DATETIME независимо от настроек часового пояса текущей сессии, используйте функцию to_datetime_ntz.

Синтаксис

```
DATETIME to_datetime(BIGINT unix_ts, INT scale)
```

Параметры

Имя	Тип	Обязательный	Описание
`unix_ts`	BIGINT	Да	Unix-метка времени для преобразования. Например, `1598306400` (секунды) и `1598306400123` (миллисекунды).
`scale`	INT	Нет	Точность времени. Допустимые значения: • `0` — секунды (по умолчанию). • `3` — миллисекунды. • `6` — микросекунды.

Возвращаемое значение

- При успехе: возвращает значение `DATETIME` на основе часового пояса текущей сессии.
- При неудаче: возвращает `NULL`. Распространённые причины:
  - Недопустимое значение `scale` (не 0, 3 или 6)
  - Значение выходит за пределы диапазона DATETIME (0001-01-01 — 9999-12-31)

Примеры

```
SET time_zone = 'Asia/Shanghai';

SELECT to_datetime(1598306400);
-- Возвращает: 2020-08-25 06:00:00

SELECT to_datetime(1598306400123, 3);
-- Возвращает: 2020-08-25 06:00:00.123000
```

To_datetime_ntz

Преобразует Unix-метку времени в значение типа DATETIME. **Эта функция всегда использует UTC+0, независимо от настроек часового пояса.**

Если вы хотите преобразовать Unix-метку времени в значение типа DATETIME на основе текущего часового пояса, используйте функцию to_datetime.

Синтаксис

```
DATETIME to_datetime_ntz(BIGINT unix_ts, INT scale)
```

Параметры

Имя	Тип	Обязательный	Описание
`unix_ts`	BIGINT	Да	Unix-метка времени для преобразования. Например, `1598306400` (секунды) и `1598306400123` (миллисекунды).
`scale`	INT	Нет	Точность времени. Допустимые значения: • `0` — секунды (по умолчанию). • `3` — миллисекунды. • `6` — микросекунды.

Возвращаемое значение

- При успехе: возвращает значение `DATETIME` на основе UTC+0.
- При неудаче: возвращает `NULL`. Распространённые причины:
  - Недопустимое значение `scale` (не 0, 3 или 6)
  - Значение выходит за пределы диапазона DATETIME (0001-01-01 — 9999-12-31)

Примеры

```
SELECT to_datetime_ntz(1598306400);
-- Возвращает: 2020-08-24 22:00:00

SELECT to_datetime_ntz(1598306400123456, 6);
-- Возвращает: 2020-08-24 22:00:00.123456
```

To_days

Возвращает количество дней между датой и 0000-01-01.

Параметр `date` должен быть типа DATE или DATETIME.

Синтаксис

```
INT TO_DAYS(DATETIME date)
```

Примеры

```
MySQL > select to_days('2007-10-07');
+-----------------------+
| to_days('2007-10-07') |
+-----------------------+
|                733321 |
+-----------------------+
```

Keyword

TO_DAYS,TO,DAYS

To_iso8601

Преобразует указанную дату в строку в формате ISO 8601.

Синтаксис

```
VARCHAR TO_ISO8601(DATETIME | DATE date)
```

Параметры

`date`: дата, которую необходимо преобразовать. Должна быть допустимой датой или выражением даты.

Примеры

Пример 1: Преобразование даты `2020-01-01` в строку в формате ISO 8601.

```
MySQL > select to_iso8601(date'2020-01-01');
+--------------------------+
| to_iso8601('2020-01-01') |
+--------------------------+
| 2020-01-01               |
+--------------------------+
```

Пример 2: Преобразование даты `2020-01-01 00:00:00.01` в строку в формате ISO 8601.

```
MySQL > select to_iso8601(datetime'2020-01-01 00:00:00.01');
+-------------------------------------------+
| to_iso8601('2020-01-01 00:00:00.010000') |
+-------------------------------------------+
| 2020-01-01T00:00:00.010000                |
+-------------------------------------------+
```

Keywords

TO_ISO8601

To_tera_date

Разбирает строку даты или времени в соответствии с указанным форматом и преобразует строку в значение DATE.

Синтаксис

```
DATE to_tera_date(VARCHAR str, VARCHAR format)
```

Параметры

- `str`: выражение времени для преобразования. Должно быть типа VARCHAR.
- `format`: спецификатор формата даты для `str`. Используется для разбора и преобразования входной строки. `format` должен соответствовать `string`. В противном случае возвращается NULL. Если `format` недопустим, возвращается ошибка.

  В следующей таблице описаны элементы формата.

Элемент	Описание
[\r \n \t - / , . ;]	Знаки пунктуации, игнорируемые при преобразовании
dd	День месяца (1 - 31)
hh	Час дня (1 - 12)
hh24	Час дня (0 - 23)
mi	Минута (0 - 59)
mm	Месяц (01 - 12)
ss	Секунда (0 - 59)
yyyy	4-значный год.
yy	2-значный год.
am	Индикатор времени суток.
pm	Индикатор времени суток.

Примеры

```
select to_tera_date("1988/04/08","yyyy/mm/dd");
+-----------------------------------------+
| to_tera_date('1988/04/08', 'yyyy/mm/dd') |
+-----------------------------------------+
| 1988-04-08                              |
+-----------------------------------------+

select to_tera_date("04-08-1988","mm-dd-yyyy");
+-----------------------------------------+
| to_tera_date('04-08-1988', 'mm-dd-yyyy') |
+-----------------------------------------+
| 1988-04-08                              |
+-----------------------------------------+

select to_tera_date(";198804:08",";yyyymm:dd");
+-----------------------------------------+
| to_tera_date(';198804:08', ';yyyymm:dd') |
+-----------------------------------------+
| 1988-04-08                              |
+-----------------------------------------+

select to_tera_date("2020-02-02 00:00:00", "yyyy-mm-dd");
+----------------------------------------------------+
| to_tera_date('2020-02-02 00:00:00', 'yyyy-mm-dd') |
+----------------------------------------------------+
| 2020-02-02                                         |
+----------------------------------------------------+

-- Входные данные содержат только год без месяца и дня. Возвращается первый
день этого года.
select to_tera_date("1988","yyyy");
+------------------------------+
| to_tera_date('1988', 'yyyy') |
+------------------------------+
| 1988-01-01                   |
+------------------------------+
```

Keywords

TO_TERA_DATE

To_tera_timestamp

Разбирает строку даты или времени в соответствии с указанным форматом и преобразует строку в значение DATETIME.

Синтаксис

```
DATETIME to_tera_timestamp(VARCHAR str, VARCHAR format)
```

Параметры

- `str`: выражение времени для преобразования. Должно быть типа VARCHAR.
- `format`: спецификатор формата времени для `str`. Используется для разбора и преобразования входной строки. `format` должен соответствовать `string`. В противном случае возвращается NULL. Если `format` недопустим, возвращается ошибка.

  В следующей таблице описаны элементы формата.

Элемент	Описание
[\r \n \t - / , . ;]	Знаки пунктуации, игнорируемые при преобразовании
dd	День месяца (1 - 31)
hh	Час дня (1 - 12)
hh24	Час дня (0 - 23)
mi	Минута (0 - 59)
mm	Месяц (01 - 12)
ss	Секунда (0 - 59)
yyyy	4-значный год.
yy	2-значный год.
am	Индикатор времени суток.
pm	Индикатор времени суток.

Примеры

```
select to_tera_timestamp("1988/04/08","yyyy/mm/dd");
+-----------------------------------------------+
| to_tera_timestamp('1988/04/08', 'yyyy/mm/dd') |
+-----------------------------------------------+
| 1988-04-08 00:00:00                           |
+-----------------------------------------------+

select to_tera_timestamp("04-08-1988","mm-dd-yyyy");
+-----------------------------------------------+
| to_tera_timestamp('04-08-1988', 'mm-dd-yyyy') |
+-----------------------------------------------+
| 1988-04-08 00:00:00                           |
+-----------------------------------------------+

select to_tera_timestamp("04.1988,08","mm.yyyy,dd");
+-----------------------------------------------+
| to_tera_timestamp('04.1988,08', 'mm.yyyy,dd') |
+-----------------------------------------------+
| 1988-04-08 00:00:00                           |
+-----------------------------------------------+

select to_tera_timestamp("1988/04/08 2","yyyy/mm/dd hh");
+-----------------------------------------------------+
| to_tera_timestamp('1988/04/08 2', 'yyyy/mm/dd hh') |
+-----------------------------------------------------+
| 1988-04-08 02:00:00                                 |
+-----------------------------------------------------+

select to_tera_timestamp("1988/04/08 14","yyyy/mm/dd hh24");
+-------------------------------------------------------+
| to_tera_timestamp('1988/04/08 14', 'yyyy/mm/dd hh24') |
+-------------------------------------------------------+
| 1988-04-08 14:00:00                                   |
+-------------------------------------------------------+

select to_tera_timestamp("1988/04/08 14:15","yyyy/mm/dd hh24:mi");
+---------------------------------------------------------------+
| to_tera_timestamp('1988/04/08 14:15', 'yyyy/mm/dd hh24:mi') |
+---------------------------------------------------------------+
| 1988-04-08 14:15:00                                           |
+---------------------------------------------------------------+

select to_tera_timestamp("1988/04/08 2:3:4","yyyy/mm/dd hh24:mi:ss");
```

```
+-----------------------------------------------------------------+
| to_tera_timestamp('1988/04/08 2:3:4', 'yyyy/mm/dd hh24:mi:ss') |
+-----------------------------------------------------------------+
| 1988-04-08 02:03:04                                             |
+-----------------------------------------------------------------+

select to_tera_timestamp("1988/04/08 02 am:3:4","yyyy/mm/dd hh am:mi:ss");
+-----------------------------------------------------------------------+
| to_tera_timestamp('1988/04/08 02 am:3:4', 'yyyy/mm/dd hh am:mi:ss') |
+-----------------------------------------------------------------------+
| 1988-04-08 02:03:04                                                   |
+-----------------------------------------------------------------------+
```

Keywords

TO_TERA_TIMESTAMP

Unix_timestamp

Преобразует значение DATE или DATETIME в UNIX-метку времени.

Если параметр не указан, функция преобразует текущее время в UNIX-метку времени.

Параметр `date` должен быть типа DATE или DATETIME.

Для времени до 1970-01-01 00:00:00 или после 2038-01-19 11:14:07 функция возвращает 0.

Дополнительную информацию о формате даты см. в разделе date_format.

Эта функция может возвращать разные результаты для разных часовых поясов. Дополнительную информацию см. в разделе Настройка часового пояса.

Синтаксис

```
BIGINT UNIX_TIMESTAMP()
BIGINT UNIX_TIMESTAMP(DATETIME date)
BIGINT UNIX_TIMESTAMP(DATETIME date, STRING fmt)
```

Примеры

```
MySQL > select unix_timestamp();
+------------------+
| unix_timestamp() |
+------------------+
|       1558589570 |
+------------------+

MySQL > select unix_timestamp('2007-11-30 10:30:19');
+----------------------------------------+
| unix_timestamp('2007-11-30 10:30:19') |
+----------------------------------------+
|                             1196389819 |
+----------------------------------------+

MySQL > select unix_timestamp('2007-11-30 10:30-19', '%Y-%m-%d %H:%i-%s');
+----------------------------------------+
| unix_timestamp('2007-11-30 10:30-19') |
+----------------------------------------+
|                             1196389819 |
+----------------------------------------+

MySQL > select unix_timestamp('2007-11-30 10:30%3A19', '%Y-%m-%d
%H:%i%%3A%s');
+----------------------------------------+
|unix_timestamp('2007-11-30 10:30%3A19')|
+----------------------------------------+
|                             1196389819 |
+----------------------------------------+

MySQL > select unix_timestamp('1969-01-01 00:00:00');
+----------------------------------------+
| unix_timestamp('1969-01-01 00:00:00') |
+----------------------------------------+
|                                      0 |
+----------------------------------------+
```

Keyword

UNIX_TIMESTAMP,UNIX,TIMESTAMP

Utc_time

Возвращает текущее время UTC в формате 'HH:MM:SS'.

Синтаксис

```
TIME UTC_TIME()
```

Примеры

```
MySQL > select utc_time();
+----------------+
| utc_time()     |
+----------------+
| 12:31:18       |
+----------------+
```

Keyword

UTC_TIME,UTC,TIME

Utc_timestamp

Возвращает текущую дату и время UTC в формате 'YYYY-MM-DD HH:MM:SS' или 'YYYYMMDDHHMMSS' в зависимости от контекста использования функции, например, в строковом или числовом контексте.

Синтаксис

```
DATETIME UTC_TIMESTAMP()
```

Примеры

```
MySQL > select utc_timestamp(),utc_timestamp() + 1;
+---------------------+---------------------+
| utc_timestamp()     | utc_timestamp() + 1 |
+---------------------+---------------------+
| 2019-07-10 12:31:18 |      20190710123119 |
+---------------------+---------------------+
```

`utc_timestamp() + N` означает добавление `N` секунд к текущему времени.

Keyword

UTC_TIMESTAMP,UTC,TIMESTAMP

Week

Возвращает номер недели для заданной даты. Эта функция работает так же, как функция WEEK в MySQL.

Функция поддерживается начиная с версии 2.3.

Синтаксис

```
INT WEEK(DATETIME|DATE date, INT mode)
```

Параметры

- `Date`: поддерживаемые типы данных — DATETIME и DATE.
- `Mode`: необязательный. Поддерживаемый тип данных — INT. Этот параметр используется для указания логики вычисления номера недели, то есть начинается ли неделя с воскресенья или понедельника, и находится ли возвращаемое значение в диапазоне 0–53 или 1–53. Диапазон значений: 0–7. Значение по умолчанию: `0`. Если этот параметр не указан, по умолчанию используется режим `0`. В следующей таблице описано, как работает этот параметр.

Режим	Первый день недели	Диапазон	Неделя 1 — это первая неделя …
0	Воскресенье	0-53	с воскресеньем в этом году
1	Понедельник	0-53	с 4 или более днями в этом году
2	Воскресенье	1-53	с воскресеньем в этом году
3	Понедельник	1-53	с 4 или более днями в этом году
4	Воскресенье	0-53	с 4 или более днями в этом году
5	Понедельник	0-53	с понедельником в этом году
6	Воскресенье	1-53	с 4 или более днями в этом году
7	Понедельник	1-53	с понедельником в этом году

Возвращаемое значение

Возвращает значение типа INT. Диапазон значений: 0–53. Конкретный диапазон определяется на основе параметра `mode`. `NULL` возвращается, если значение `date` недопустимо или входное значение пустое.

Примеры

Вычисление номера недели для `2007-01-01`. `2007-01-01` — понедельник по календарю.

- `Mode` установлен в `0`, возвращается `0`. Неделя начинается с воскресенья. `2007-01-01` — понедельник и не может быть неделей 1. Поэтому возвращается `0`.

```
mysql> SELECT WEEK('2007-01-01', 0);
+-----------------------+
| week('2007-01-01', 0) |
+-----------------------+
|                     0 |
+-----------------------+
1 row in set (0.02 sec)
```

- `Mode` установлен в `1`, возвращается `1`. Неделя начинается с понедельника, и `2007-01-01` — понедельник.

```
mysql> SELECT WEEK('2007-01-01', 1);
+-----------------------+
| week('2007-01-01', 1) |
+-----------------------+
|                     1 |
+-----------------------+
1 row in set (0.02 sec)
```

- `Mode` установлен в `2`, возвращается `53`. Неделя начинается с воскресенья. Однако `2007-01-01` — понедельник, а диапазон значений 1–53. Поэтому возвращается `53`, что означает последнюю неделю предыдущего года.

```
mysql> SELECT WEEK('2007-01-01', 2);
+-----------------------+
| week('2007-01-01', 2) |
+-----------------------+
|                    53 |
+-----------------------+
1 row in set (0.01 sec)
```

Week_iso

Возвращает номер недели по стандарту ISO для указанной даты в виде целого числа в диапазоне от `1` до `53`.

Синтаксис

```
INT WEEK_ISO(DATETIME date)
```

Параметры

`date`: дата, которую необходимо преобразовать. Должна быть типа DATE или DATETIME.

Примеры

Следующий пример возвращает номер недели по стандарту ISO для даты `2008-02-20 00:00:00`:

```
MySQL > select week_iso ('2008-02-20 00:00:00');
+-----------------------------------+
| week_iso('2008-02-20 00:00:00')   |
+-----------------------------------+
|                                 8 |
+-----------------------------------+
```

Keywords

WEEK_ISO

Weekofyear

Возвращает номер недели для заданной даты в пределах года.

Параметр `date` должен быть типа DATE или DATETIME.

Синтаксис

```
INT WEEKOFYEAR(DATETIME date)
```

Примеры

```
MySQL > select weekofyear('2008-02-20 00:00:00');
+-----------------------------------+
| weekofyear('2008-02-20 00:00:00') |
+-----------------------------------+
|                                 8 |
+-----------------------------------+
```

Keyword

WEEKOFYEAR

Weeks_add

Возвращает значение с добавлением указанного количества недель к дате.

Синтаксис

```
DATETIME weeks_add(DATETIME expr1, INT expr2);
```

Параметры

- `expr1`: исходная дата. Должна быть типа `DATETIME`.
- `expr2`: количество недель. Должно быть типа `INT`.

Возвращаемое значение

Возвращает `DATETIME`.

`NULL` возвращается, если дата не существует.

Примеры

```
select weeks_add('2022-12-20',2);
+----------------------------+
| weeks_add('2022-12-20', 2) |
+----------------------------+
|        2023-01-03 00:00:00 |
+----------------------------+
```

Weeks_diff

Возвращает разницу в неделях между двумя выражениями даты (`expr1` – `expr2`) с точностью до недели.

Синтаксис

```
BIGINT weeks_diff(DATETIME expr1,DATETIME expr2);
```

Параметры

- `expr1`: время окончания. Должно быть типа DATETIME или DATE.
- `expr2`: время начала. Должно быть типа DATETIME или DATE.

Возвращаемое значение

Возвращает значение типа BIGINT.

NULL возвращается, если дата не существует, например, 2022-02-29.

Примеры

```
select weeks_diff('2010-11-30 23:59:59', '2010-1-1 23:59:59');
+-----------------------------------------------------------+
| weeks_diff('2010-11-30 23:59:59', '2010-1-1 23:59:59') |
+-----------------------------------------------------------+
|                                                        47 |
+-----------------------------------------------------------+

select weeks_diff(current_time(), '2010-11-30 23:59:59');
+-----------------------------------------------------+
| weeks_diff(current_time(), '2010-11-30 23:59:59') |
+-----------------------------------------------------+
|                                                 619 |
+-----------------------------------------------------+

select weeks_diff('2010-11-30 23:59:59', '2010-11-24 23:59:59');
+-------------------------------------------------------------+
| weeks_diff('2010-11-30 23:59:59', '2010-11-24 23:59:59') |
+-------------------------------------------------------------+
|                                                           0 |
+-------------------------------------------------------------+
```

Weeks_sub

Вычитает указанное количество недель из значения даты или даты-времени.

Синтаксис

```
DATETIME weeks_sub(DATETIME expr1, INT expr2);
```

Параметры

- `expr1`: исходная дата. Должна быть типа `DATETIME`.
- `expr2`: количество недель. Должно быть типа `INT`.

Возвращаемое значение

Возвращает значение типа `DATETIME`.

Возвращает `NULL`, если дата не существует.

Примеры

```
select weeks_sub('2022-12-22',2);
+----------------------------+
| weeks_sub('2022-12-22', 2) |
+----------------------------+
|        2022-12-08 00:00:00 |
+----------------------------+
```

Year

Возвращает компонент года из даты в диапазоне от 1000 до 9999.

Параметр `date` должен быть типа DATE или DATETIME.

Синтаксис

```
INT YEAR(DATETIME date)
```

Примеры

```
MySQL > select year('1987-01-01');
+----------------------------+
| year('1987-01-01 00:00:00') |
+----------------------------+
|                        1987 |
+----------------------------+
```

Ключевые слова

YEAR

Years_add

Описание

Синтаксис

```
DATETIME YEARS_ADD(DATETIME expr1,INT expr2)
```

Добавляет указанный временной интервал к дате. Единица измерения - год.

Параметр expr1 должен быть допустимым выражением даты-времени.

Параметр expr2 - количество лет, которое нужно добавить.

Примеры

```
select years_add('2010-11-30 23:50:50', 2);
+-------------------------------------+
| years_add('2010-11-30 23:50:50', 2) |
+-------------------------------------+
| 2012-11-30 23:50:50                 |
+-------------------------------------+

select years_add('2010-11-30', 2);
+----------------------------+
| years_add('2010-11-30', 2) |
+----------------------------+
| 2012-11-30 00:00:00        |
+----------------------------+
```

Ключевые слова

YEARS_ADD,ADD

Years_diff

Возвращает разницу в годах между двумя выражениями дат (`expr1` - `expr2`) с точностью до года.

Синтаксис

```
BIGINT years_diff(DATETIME expr1,DATETIME expr2);
```

Параметры

- `expr1`: конечное время. Должно быть типа DATETIME или DATE.
- `expr2`: начальное время. Должно быть типа DATETIME или DATE.

Возвращаемое значение

Возвращает значение типа BIGINT.

Возвращает NULL, если дата не существует, например, 2022-02-29.

Примеры

```
select years_diff('2010-11-30 23:59:59', '2000-11-1 23:59:59');
+-----------------------------------------------------------+
| years_diff('2010-11-30 23:59:59', '2000-11-1 23:59:59') |
+-----------------------------------------------------------+
|                                                        10 |
+-----------------------------------------------------------+

select years_diff('2010-11-30', '2000-11-1');
+---------------------------------------+
| years_diff('2010-11-30', '2000-11-1') |
+---------------------------------------+
|                                    10 |
+---------------------------------------+
```

Years_sub

Вычитает указанное количество лет из указанной даты или даты-времени.

Синтаксис

```
DATETIME YEARS_SUB(DATETIME date, INT years)
```

Параметры

`date`: исходная дата-время, типа DATETIME или DATE.

`years`: количество лет для вычитания. Значение может быть отрицательным, но год даты минус years не должен превышать 10000. Например, если год даты 2022, то years не может быть меньше -7979. В то же время years не может превышать значение года даты, например, если год даты 2022, то years не может быть больше 2022.

Возвращаемое значение

Тип возвращаемого значения совпадает с типом параметра `date`. Возвращает NULL, если результирующий год выходит за пределы диапазона [0, 9999].

Примеры

```
select years_sub("2022-12-20 15:50:21", 2);
+-------------------------------------+
| years_sub('2022-12-20 15:50:21', 2) |
+-------------------------------------+
| 2020-12-20 15:50:21                 |
+-------------------------------------+
```

Yearweek

Возвращает год и номер недели для заданной даты. Эта функция работает так же, как функция yearweek() в MySQL.

Эта функция поддерживается начиная с версии 3.3.

Синтаксис

```
INT YEARWEEK(DATETIME|DATE date[, INT mode])
```

Параметры

- `date`: поддерживаемые типы данных - DATETIME и DATE.
- `mode`: необязательный. Поддерживаемый тип данных - INT. Этот параметр используется для указания логики вычисления номера недели года: начинается ли неделя с воскресенья или понедельника, и находится ли возвращаемое значение в диапазоне 0~~53 или 1~~53. Диапазон значений: 0~7. Значение по умолчанию: `0`. Если этот параметр не указан, по умолчанию используется режим `0`. В следующей таблице описано, как работает этот параметр.

Режим	Первый день недели	Диапазон	Неделя 1 - это первая неделя...
0	Воскресенье	0-53	с воскресеньем в этом году
1	Понедельник	0-53	с 4 или более днями в этом году
2	Воскресенье	1-53	с воскресеньем в этом году

Режим	Первый день недели	Диапазон	Неделя 1 - это первая неделя...
3	Понедельник	1-53	с 4 или более днями в этом году
4	Воскресенье	0-53	с 4 или более днями в этом году
5	Понедельник	0-53	с понедельником в этом году
6	Воскресенье	1-53	с 4 или более днями в этом году
7	Понедельник	1-53	с понедельником в этом году

Возвращаемое значение

Возвращает значение типа INT. Диапазон значений: 0~53. Конкретный диапазон определяется на основе параметра `mode`. Возвращает `NULL`, если значение `date` недопустимо или входное значение пустое.

Примеры

Возвращает год и неделю для даты. Значение аргумента mode по умолчанию равно 0

```
mysql> SELECT YEARWEEK('2007-01-01', 0);
+---------------------------+
| yearweek('2007-01-01', 0) |
+---------------------------+
|                    200653 |
+---------------------------+
```

```
mysql> SELECT YEARWEEK('2007-01-01', 1);
+---------------------------+
| yearweek('2007-01-01', 1) |
+---------------------------+
|                    200701 |
+---------------------------+
```

```
mysql> SELECT YEARWEEK('2007-01-01', 2);
+---------------------------+
| yearweek('2007-01-01', 2) |
+---------------------------+
|                    200653 |
+---------------------------+
1 row in set (0.01 sec)
```

2.15 Dictionary

Dict_mapping

Возвращает значение, сопоставленное с указанным ключом в таблице словаря.

Эта функция в основном используется для упрощения применения глобальной таблицы словаря. При загрузке данных в целевую таблицу Selena автоматически получает значение, сопоставленное с указанным ключом из таблицы словаря, используя входные параметры этой функции, а затем загружает значение в целевую таблицу.

Начиная с версии 3.2.5, Selena поддерживает эту функцию. Также обратите внимание, что в настоящее время режим shared-data в Selena не поддерживает эту функцию.

Синтаксис

```
dict_mapping("[<db_name>.]<dict_table>", key_column_expr_list [,
<value_column> ] [, <null_if_not_exist>] )

key_column_expr_list ::= key_column_expr [, key_column_expr ... ]

key_column_expr ::= <column_name> | <expr>
```

Параметры

- Обязательные параметры:
  - `[<db_name>.]<dict_table>`: имя таблицы словаря, которая должна быть таблицей с первичным ключом. Поддерживаемый тип данных - VARCHAR.
  - `key_column_expr_list`: список выражений для ключевых столбцов в таблице словаря, включающий одно или несколько `key_column_expr`. `key_column_expr` может быть именем ключевого столбца в таблице словаря или конкретным ключом или выражением ключа.

    Этот список выражений должен включать все столбцы первичного ключа таблицы словаря, то есть общее количество выражений должно соответствовать общему количеству столбцов первичного ключа в таблице словаря. Поэтому когда таблица словаря использует составной первичный ключ, выражения в этом списке должны соответствовать столбцам первичного ключа, определённым в

схеме таблицы, по порядку. Несколько выражений в этом списке разделяются запятыми (`,`). Если `key_column_expr` является конкретным ключом или выражением ключа, его тип должен соответствовать типу соответствующего столбца первичного ключа в таблице словаря.

- Необязательные параметры:
  - `<value_column>`: имя столбца значения, который также является столбцом сопоставления. Если столбец значения не указан, столбцом значения по умолчанию является столбец AUTO_INCREMENT таблицы словаря. Столбец значения также может быть определён как любой столбец в таблице словаря, исключая автоинкрементные столбцы и первичные ключи. Тип данных столбца не имеет ограничений.
  - `<null_if_not_exist>` (необязательный): возвращать ли NULL, если ключ не существует в таблице словаря. Допустимые значения:
    - `true`: возвращает NULL, если ключ не существует.
    - `false` (по умолчанию): генерирует исключение, если ключ не существует.

Возвращаемое значение

Тип данных возвращаемых значений соответствует типу данных столбца значения. Если столбец значения является автоинкрементным столбцом таблицы словаря, тип данных возвращаемых значений - BIGINT.

Однако когда значение, сопоставленное с указанным ключом, не найдено, если параметр `<null_if_not_exist>` установлен в `true`, возвращается `NULL`. Если параметр установлен в `false` (по умолчанию), возвращается ошибка `query failed if record not exist in dict table`.

Примеры

Пример 1: Прямой запрос значения, сопоставленного с ключом, из таблицы словаря.

1. Создайте таблицу словаря и загрузите тестовые данные.

```
MySQL [test]> CREATE TABLE dict (
    order_uuid STRING,
    order_id_int BIGINT AUTO_INCREMENT
)
PRIMARY KEY (order_uuid)
DISTRIBUTED BY HASH (order_uuid);
Query OK, 0 rows affected (0.02 sec)

MySQL [test]> INSERT INTO dict (order_uuid) VALUES ('a1'), ('a2'),
('a3');
Query OK, 3 rows affected (0.12 sec)
{'label':'insert_9e60b0e4-89fa-11ee-a41f-b22a2c00f66b',
'status':'VISIBLE', 'txnId':'15029'}

MySQL [test]> SELECT * FROM dict;
+------------+--------------+
| order_uuid | order_id_int |
+------------+--------------+
| a1         |            1 |
| a3         |            3 |
| a2         |            2 |
+------------+--------------+
3 rows in set (0.01 sec)
```

> **ПРИМЕЧАНИЕ**
>
> В настоящее время оператор `INSERT INTO` не поддерживает частичные обновления. Поэтому убедитесь, что значения, вставляемые в ключевой столбец `dict`, не дублируются. В противном случае вставка одного и того же значения ключевого столбца в таблицу словаря несколько раз приведёт к изменению сопоставленного значения в столбце значений.

2. Запросите значение, сопоставленное с ключом `a1`, в таблице словаря.

```
MySQL [test]> SELECT dict_mapping('dict', 'a1');
+----------------------------+
| dict_mapping('dict', 'a1') |
+----------------------------+
|                          1 |
+----------------------------+
1 row in set (0.01 sec)
```

Пример 2: Столбец сопоставления в таблице настроен как генерируемый столбец с использованием функции `dict_mapping`. Таким образом, Selena может автоматически получать значения, сопоставленные с ключами, при загрузке данных в эту таблицу.

1. Создайте таблицу данных и настройте столбец сопоставления как генерируемый столбец с использованием `dict_mapping('dict', order_uuid)`.

```
CREATE TABLE dest_table1 (
    id BIGINT,
    -- Этот столбец записывает номер заказа типа STRING,
соответствующий столбцу order_uuid в таблице dict из Примера 1.
    order_uuid STRING,
    batch int comment 'используется для различения разных пакетных
загрузок',
    -- Этот столбец записывает номер заказа типа BIGINT, сопоставленный
со столбцом order_uuid.
    -- Поскольку этот столбец является генерируемым столбцом,
настроенным с dict_mapping, значения в этом столбце автоматически
получаются из таблицы dict из Примера 1 во время загрузки данных.
    -- Впоследствии этот столбец можно напрямую использовать для
дедупликации и JOIN-запросов.
    order_id_int BIGINT AS dict_mapping('dict', order_uuid)
)
DUPLICATE KEY (id, order_uuid)
DISTRIBUTED BY HASH(id);
```

2. При загрузке тестовых данных в эту таблицу, где столбец `order_id_int` настроен как `dict_mapping('dict', 'order_uuid')`, Selena автоматически загружает значения в столбец `order_id_int` на основе сопоставления между ключами и значениями в таблице `dict`.

```
MySQL [test]> INSERT INTO dest_table1(id, order_uuid, batch) VALUES (1,
'a1', 1), (2, 'a1', 1), (3, 'a3', 1), (4, 'a3', 1);
Query OK, 4 rows affected (0.05 sec)
{'label':'insert_e191b9e4-8a98-11ee-b29c-00163e03897d',
'status':'VISIBLE', 'txnId':'72'}

MySQL [test]> SELECT * FROM dest_table1;
+------+------------+-------+--------------+
| id   | order_uuid | batch | order_id_int |
+------+------------+-------+--------------+
|    1 | a1         |     1 |            1 |
|    4 | a3         |     1 |            3 |
|    2 | a1         |     1 |            1 |
|    3 | a3         |     1 |            3 |
+------+------------+-------+--------------+
4 rows in set (0.02 sec)
```

Использование `dict_mapping` в этом примере может ускорить вычисления дедупликации и JOIN-запросы. По сравнению с предыдущими решениями для построения глобального словаря для ускорения точной дедупликации, решение с использованием `dict_mapping` более гибкое и удобное для пользователя. Поскольку сопоставленные значения напрямую получаются из таблицы словаря на этапе "загрузки сопоставлений между ключами и значениями в таблицу". Вам не нужно писать операторы для соединения с таблицей словаря для получения сопоставленных значений. Кроме того, это решение поддерживает различные методы загрузки данных.

Пример 3: Если столбец сопоставления в таблице не настроен как генерируемый столбец, необходимо явно настроить функцию `dict_mapping` для столбца сопоставления при загрузке данных в таблицу, чтобы получить значения, сопоставленные с ключами.

> **ПРИМЕЧАНИЕ**
>
> Разница между Примером 3 и Примером 2 заключается в том, что при импорте в таблицу данных необходимо изменить команду импорта, чтобы явно настроить выражение `dict_mapping` для столбца сопоставления.

1. Создайте таблицу.

```
CREATE TABLE dest_table2 (
    id BIGINT,
    order_uuid STRING,
    order_id_int BIGINT NULL,
    batch int comment 'используется для различения разных пакетных
загрузок'
)
DUPLICATE KEY (id, order_uuid, order_id_int)
DISTRIBUTED BY HASH(id);
```

2. При загрузке тестовых данных в эту таблицу вы получаете сопоставленные значения из таблицы словаря, настроив `dict_mapping`.

```
MySQL [test]> INSERT INTO dest_table2 VALUES (1, 'a1',
dict_mapping('dict', 'a1'), 1);
Query OK, 1 row affected (0.35 sec)
{'label':'insert_19872ab6-8a96-11ee-b29c-00163e03897d',
'status':'VISIBLE', 'txnId':'42'}

MySQL [test]> SELECT * FROM dest_table2;
+------+------------+--------------+-------+
| id   | order_uuid | order_id_int | batch |
+------+------------+--------------+-------+
|    1 | a1         |            1 |     1 |
+------+------------+--------------+-------+
1 row in set (0.02 sec)
```

Пример 4: Включение режима null_if_not_exist

Когда режим `<null_if_not_exist>` отключён и запрашивается значение, сопоставленное с ключом, который не существует в таблице словаря, возвращается ошибка вместо `NULL`. Это гарантирует, что ключ строки данных сначала загружается в таблицу словаря и генерируется его сопоставленное значение (ID словаря) перед загрузкой этой строки данных в целевую таблицу.

```
MySQL [test]>  SELECT dict_mapping('dict', 'b1', true);
ERROR 1064 (HY000): Query failed if record not exist in dict table.
```

Пример 5: Если таблица словаря использует составные первичные ключи, при запросе должны быть указаны все первичные ключи.

1. Создайте таблицу словаря с составными первичными ключами и загрузите в неё тестовые данные.

```
MySQL [test]> CREATE TABLE dict2 (
    order_uuid STRING,
    order_date DATE,
    order_id_int BIGINT AUTO_INCREMENT
)
PRIMARY KEY (order_uuid,order_date)  -- Составной первичный ключ
DISTRIBUTED BY HASH (order_uuid,order_date)
;
Query OK, 0 rows affected (0.02 sec)

MySQL [test]> INSERT INTO dict2 VALUES ('a1','2023-11-22',default),
('a2','2023-11-22',default), ('a3','2023-11-22',default);
Query OK, 3 rows affected (0.12 sec)
{'label':'insert_9e60b0e4-89fa-11ee-a41f-b22a2c00f66b',
'status':'VISIBLE', 'txnId':'15029'}


MySQL [test]> select * from dict2;
+------------+------------+--------------+
| order_uuid | order_date | order_id_int |
+------------+------------+--------------+
| a1         | 2023-11-22 |            1 |
| a3         | 2023-11-22 |            3 |
| a2         | 2023-11-22 |            2 |
+------------+------------+--------------+
3 rows in set (0.01 sec)
```

2. Запросите значение, сопоставленное с ключом, в таблице словаря. Поскольку таблица словаря имеет составные первичные ключи, в `dict_mapping` должны быть указаны все первичные ключи.

```
SELECT dict_mapping('dict2', 'a1', cast('2023-11-22' as DATE));
```

Обратите внимание, что ошибка возникает, если указан только один первичный ключ.

```
MySQL [test]> SELECT dict_mapping('dict2', 'a1');
ERROR 1064 (HY000): Getting analyzing error. Detail message:
dict_mapping function param size should be 3 - 5.
```

Dictionary_get

Запрашивает значение, сопоставленное с ключом, в объекте словаря.

Синтаксис

```
dictionary_get('dictionary_object_name', key_expression_list,
[NULL_IF_NOT_EXIST])

key_expression_list ::=
    key_expression [, ...]

key_expression ::=
    column_name | const_value
```

Параметры

- `dictionary_name`: имя объекта словаря.
- `key_expression_list`: список выражений для всех ключевых столбцов. Это может быть список имён столбцов или список значений.
- `NULL_IF_NOT_EXIST` (необязательный): возвращать ли NULL, если ключ не существует в кэше словаря. Допустимые значения:
  - `true`: возвращает NULL, если ключ не существует.
  - `false` (по умолчанию): генерирует исключение, если ключ не существует.

Возвращаемое значение

Возвращает значения столбцов значений как тип STRUCT. Поэтому вы можете использовать `[N]` или `.<column_name>` для указания значения конкретного столбца. `N` представляет позицию столбца, начиная с 1.

Примеры

Следующие примеры используют набор данных из примеров dict_mapping.

- Пример 1: Запрос значений столбца значений, сопоставленного с ключевым столбцом `order_uuid`, в объекте словаря `dict_obj`.

```
MySQL > SELECT dictionary_get('dict_obj', order_uuid) FROM dict;
+--------------------+
| DICTIONARY_GET     |
+--------------------+
| {"order_id_int":1} |
| {"order_id_int":3} |
| {"order_id_int":2} |
+--------------------+
3 rows in set (0.02 sec)
```

- Пример 2: Запрос значения столбца значений, сопоставленного с ключом `a1`, в объекте словаря `dict_obj`.

```
MySQL > SELECT dictionary_get("dict_obj", "a1");
+--------------------+
| DICTIONARY_GET     |
+--------------------+
| {"order_id_int":1} |
+--------------------+
1 row in set (0.01 sec)
```

- Пример 3: Запрос значений столбцов значений, сопоставленных с ключом `1`, в объекте словаря `dimension_obj`.

```
MySQL > SELECT dictionary_get("dimension_obj", 1);
+-----------------------------------------------------------------------
---------------------+
| DICTIONARY_GET
|
+-----------------------------------------------------------------------
---------------------+
| {"ProductName":"T-
Shirt","Category":"Apparel","SubCategory":"Shirts","Brand":"BrandA","Colo
|
+-----------------------------------------------------------------------
---------------------+
1 row in set (0.01 sec)
```

- Пример 4: Запрос значения первого столбца значений, сопоставленного с ключом `1`, в объекте словаря `dimension_obj`.

```
MySQL > SELECT dictionary_get("dimension_obj", 1)[1];
+-------------------+
| DICTIONARY_GET[1] |
+-------------------+
| T-Shirt           |
+-------------------+
1 row in set (0.01 sec)
```

- Пример 5: Запрос значения второго столбца значений, сопоставленного с ключом `1`, в объекте словаря `dimension_obj`.

```
MySQL > SELECT dictionary_get("dimension_obj", 1)[2];
+-------------------+
| DICTIONARY_GET[2] |
+-------------------+
| Apparel           |
+-------------------+
1 row in set (0.01 sec)
```

- Пример 6: Запрос значения столбца значений `ProductName`, сопоставленного с ключом `1`, в объекте словаря `dimension_obj`.

```
MySQL > SELECT dictionary_get("dimension_obj", 1).ProductName;
+----------------------------+
| DICTIONARY_GET.ProductName |
+----------------------------+
| T-Shirt                    |
+----------------------------+
1 row in set (0.01 sec)
```

2.16 Hash

Murmur_hash3_32

Возвращает 32-битное значение хеша murmur3 для входной строки.

Синтаксис

```
INT MURMUR_HASH3_32(VARCHAR input, ...)
```

Примеры

```
MySQL > select murmur_hash3_32(null);
+------------------------+
| murmur_hash3_32(NULL) |
+------------------------+
|                   NULL |
+------------------------+

MySQL > select murmur_hash3_32("hello");
+---------------------------+
| murmur_hash3_32('hello') |
+---------------------------+
|                1321743225 |
+---------------------------+

MySQL > select murmur_hash3_32("hello", "world");
+-----------------------------------+
| murmur_hash3_32('hello', 'world') |
+-----------------------------------+
|                          984713481 |
+-----------------------------------+
```

Ключевые слова

MURMUR_HASH3_32,HASH

Xx_hash3_128

Возвращает 128-битное значение хеша xxhash3 для входной строки.

Синтаксис

```
LARGEINT XX_HASH3_128(VARCHAR input, ...)
```

Примеры

```
MySQL > select xx_hash3_128(null);
+--------------------+
| xx_hash3_128(NULL) |
+--------------------+
|               NULL |
+--------------------+

MySQL > select xx_hash3_128("hello");
+-----------------------------------------+
| xx_hash3_128('hello')                   |
+-----------------------------------------+
| -98478366302105124680504504609445627880 |
+-----------------------------------------+

MySQL > select xx_hash3_128("hello", "world");
+-----------------------------------------+
| xx_hash3_128('hello', 'world')          |
+-----------------------------------------+
| -45235302294609689438180196264627189905 |
+-----------------------------------------+
```

Ключевые слова

XX_HASH3_128,HASH,xxHash3

Xx_hash3_64

Возвращает 64-битное значение хеша xxhash3 для входной строки. xx_hash3_64 имеет лучшую производительность, чем murmur_hash3_32, благодаря использованию инструкции AVX2 и обеспечивает качество хеширования на уровне современных стандартов, что широко интегрировано во многие программные продукты.

Эта функция поддерживается начиная с версии 3.2.0.

Синтаксис

```
BIGINT XX_HASH3_64(VARCHAR input, ...)
```

Примеры

```
MySQL > select xx_hash3_64(null);
+-------------------+
| xx_hash3_64(NULL) |
+-------------------+
|              NULL |
+-------------------+

MySQL > select xx_hash3_64("hello");
+----------------------+
| xx_hash3_64('hello') |
+----------------------+
| -7685981735718036227 |
+----------------------+

MySQL > select xx_hash3_64("hello", "world");
+------------------------------+
| xx_hash3_64('hello', 'world') |
+------------------------------+
|           7001965798170371843 |
+------------------------------+
```

Ключевые слова

XX_HASH3_64,HASH,xxHash3

2.17 JSON

JSON Constructor

Json_array

Преобразует каждый элемент SQL-массива в JSON-значение и возвращает JSON-массив, состоящий из этих JSON-значений.

> **ПОДСКАЗКА**
>
> Все JSON-функции и операторы перечислены в навигации и на обзорной странице
>
> Ускорьте ваши запросы с помощью генерируемых столбцов

Синтаксис

```
json_array(value, ...)
```

Параметры

`value`: элемент в SQL-массиве. Поддерживаются только значения `NULL` и следующие типы данных: STRING, VARCHAR, CHAR, JSON, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, DOUBLE, FLOAT и BOOLEAN.

Возвращаемое значение

Возвращает JSON-массив.

Примеры

Пример 1: Создание JSON-массива, состоящего из значений разных типов данных.

```
mysql> SELECT json_array(1, true, 'selena', 1.1);

       -> [1, true, "selena", 1.1]
```

Пример 2: Создание пустого JSON-массива.

```
mysql> SELECT json_array();

       -> []
```

Json_object

Преобразует одну или несколько пар ключ-значение в JSON-объект, состоящий из этих пар ключ-значение. Пары ключ-значение сортируются по ключу в алфавитном порядке.

> **ПОДСКАЗКА**
>
> Все JSON-функции и операторы перечислены в навигации и на обзорной странице
>
> Ускорьте ваши запросы с помощью генерируемых столбцов

Синтаксис

```
json_object(key, value, ...)
```

Параметры

- `key`: ключ в JSON-объекте. Поддерживается только тип данных VARCHAR.
- `value`: значение в JSON-объекте. Поддерживаются только значения `NULL` и следующие типы данных: STRING, VARCHAR, CHAR, JSON, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, DOUBLE, FLOAT и BOOLEAN.

Возвращаемое значение

Возвращает JSON-объект.

> Если общее количество ключей и значений нечётное, функция JSON_OBJECT заполняет `NULL` в последнем поле.

Примеры

Пример 1: Создание JSON-объекта, состоящего из значений разных типов данных.

```
mysql> SELECT json_object('name', 'selena', 'active', true, 'published',
2020);

       -> {"active": true, "name": "selena", "published": 2020}
```

Пример 2: Создание JSON-объекта с использованием вложенных функций JSON_OBJECT.

```
mysql> SELECT json_object('k1', 1, 'k2', json_object('k2', 2), 'k3',
json_array(4, 5));

       -> {"k1": 1, "k2": {"k2": 2}, "k3": [4, 5]}
```

Пример 3: Создание пустого JSON-объекта.

```
mysql> SELECT json_object();

       -> {}
```

Parse_json

Преобразует строку в JSON-значение.

ПОДСКАЗКА

Все JSON-функции и операторы перечислены в навигации и на обзорной странице

Ускорьте ваши запросы с помощью генерируемых столбцов

Синтаксис

```
parse_json(string_expr)
```

Параметры

`string_expr`: выражение, представляющее строку. Поддерживаются только типы данных STRING, VARCHAR и CHAR.

Возвращаемое значение

Возвращает JSON-значение.

> Примечание: Если строку невозможно разобрать в стандартное JSON-значение, функция PARSE_JSON возвращает `NULL` (см. Пример 5). Информацию о спецификациях JSON см. в RFC 7159.

Примеры

Пример 1: Преобразование STRING-значения `1` в JSON-значение `1`.

```
mysql> SELECT parse_json('1');
+-----------------+
| parse_json('1') |
+-----------------+
| "1"             |
+-----------------+
```

Пример 2: Преобразование массива типа STRING в JSON-массив.

```
mysql> SELECT parse_json('[1,2,3]');
+-----------------------+
| parse_json('[1,2,3]') |
+-----------------------+
| [1, 2, 3]             |
+-----------------------+
```

Пример 3: Преобразование объекта типа STRING в JSON-объект.

```
mysql> SELECT parse_json('{"star": "rocks"}');
+---------------------------------+
| parse_json('{"star": "rocks"}') |
+---------------------------------+
| {"star": "rocks"}               |
+---------------------------------+
```

Пример 4: Создание JSON-значения `NULL`.

```
mysql> SELECT parse_json('null');
+--------------------+
| parse_json('null') |
+--------------------+
| "null"             |
+--------------------+
```

Пример 5: Если строку невозможно разобрать в стандартное JSON-значение, функция PARSE_JSON возвращает `NULL`. В этом примере `star` не заключён в двойные кавычки ("). Поэтому функция PARSE_JSON возвращает `NULL`.

```
mysql> SELECT parse_json('{star: "rocks"}');
+-------------------------------+
| parse_json('{star: "rocks"}') |
+-------------------------------+
| NULL                          |
+-------------------------------+
```

Пример 6: Если JSON-ключ содержит '.', например, 'a.1', его нужно экранировать с помощью '\' или заключить весь ключ вместе с двойными кавычками в одинарные кавычки.

```
mysql> select parse_json('{"b":4, "a.1": "1"}')->"a\\.1";
+---------------------------------------------+
| parse_json('{"b":4, "a.1": "1"}')->'a\\.1' |
+---------------------------------------------+
| "1"                                         |
+---------------------------------------------+
mysql> select parse_json('{"b":4, "a.1": "1"}')->'"a.1"';
+---------------------------------------------+
| parse_json('{"b":4, "a.1": "1"}')->'"a.1"' |
+---------------------------------------------+
| "1"                                         |
+---------------------------------------------+
```

Ключевые слова

parse_json, parse json

JSON query and processing

Функция стрелки

Запрашивает элемент, который может быть найден по выражению `json_path` в JSON-объекте, и возвращает JSON-значение. Функция стрелки `->` более компактна и проще в использовании, чем функция json_query.

> **ПОДСКАЗКА**
>
> Все JSON-функции и операторы перечислены в навигации и на обзорной странице
>
> Ускорьте ваши запросы с помощью генерируемых столбцов

Синтаксис

```
json_object_expr -> json_path
```

Параметры

- `json_object_expr`: выражение, представляющее JSON-объект. Объект может быть JSON-столбцом или JSON-объектом, созданным JSON-функцией-конструктором, такой как PARSE_JSON.

- `json_path`: выражение, представляющее путь к элементу в JSON-объекте. Значение этого параметра - строка. Информацию о синтаксисе JSON-пути, поддерживаемом Selena, см. в Обзоре JSON-функций и операторов.

Возвращаемое значение

Возвращает JSON-значение.

> Если элемент не существует, функция стрелки возвращает SQL-значение `NULL`.

Примеры

Пример 1: Запрос элемента, который может быть найден по выражению `'$.a.b'` в указанном JSON-объекте.

```
mysql> SELECT parse_json('{"a": {"b": 1}}') -> '$.a.b';

       -> 1
```

Пример 2: Использование вложенных функций стрелки для запроса элемента. Функция стрелки, в которую вложена другая функция стрелки, запрашивает элемент на основе результата, возвращённого вложенной функцией стрелки.

> В этом примере корневой элемент $ опущен в выражении `json_path`.

```
mysql> SELECT parse_json('{"a": {"b": 1}}')->'a'->'b';

       -> 1
```

Пример 3: Запрос элемента, который может быть найден по выражению `'a'` в указанном JSON-объекте.

> В этом примере корневой элемент $ опущен в выражении `json_path`.

```
mysql> SELECT parse_json('{"a": "b"}') -> 'a';

       -> "b"
```

Cast

Преобразует значение между типом JSON и типом SQL.

> **ПОДСКАЗКА**
>
> Все JSON-функции и операторы перечислены в навигации и на обзорной странице
>
> Ускорьте ваши запросы с помощью генерируемых столбцов

Синтаксис

- Преобразование из JSON в SQL

```
cast(json_expr AS sql_data_type)
```

- Преобразование из SQL в JSON

```
cast(sql_expr AS JSON)
```

Параметры

- `json_expr`: выражение, представляющее JSON-значение, которое вы хотите преобразовать в SQL-значение.
- `sql_data_type`: тип данных SQL, в который вы хотите преобразовать JSON-значение. Поддерживаются только типы данных STRING, VARCHAR, CHAR, BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, DOUBLE и FLOAT.
- `sql_expr`: выражение, представляющее SQL-значение, которое вы хотите преобразовать в JSON-значение. Этот параметр поддерживает все типы данных SQL, поддерживаемые параметром `sql_data_type`.

Возвращаемое значение

- Если вы используете синтаксис `cast(json_expr AS sql_data_type)`, функция cast возвращает значение типа данных SQL, указанного параметром `sql_data_type`.
- Если вы используете синтаксис `cast(sql_expr AS JSON)`, функция cast возвращает JSON-значение.

Примечания по использованию

- Преобразование из SQL в JSON
  - Если SQL-значение превышает точность, поддерживаемую JSON, функция cast возвращает `NULL` для предотвращения арифметического переполнения.
  - Если SQL-значение равно `NULL`, функция cast не преобразует SQL-значение `NULL` в JSON-значение `NULL`. Возвращаемое значение по-прежнему является SQL-значением `NULL`.
- Преобразование из JSON в SQL
  - Функция cast поддерживает только преобразования между совместимыми типами данных JSON и SQL. Например, вы можете преобразовать JSON-строку в SQL-

строку.

- Функция cast не поддерживает преобразования между несовместимыми типами данных JSON и SQL. Например, если вы преобразуете JSON-число в SQL-строку, функция возвращает `NULL`.
- Если происходит арифметическое переполнение, функция cast возвращает SQL-значение `NULL`.
- Если вы преобразуете JSON-значение `NULL` в SQL-значение, функция возвращает SQL-значение `NULL`.
- Если вы преобразуете JSON-строку в значение VARCHAR, функция возвращает значение VARCHAR, не заключённое в двойные кавычки (").

Примеры

Пример 1: Преобразование JSON-значения в SQL-значение.

```
-- Преобразование JSON-значения в значение INT.
mysql> select cast(parse_json('{"a": 1}') -> 'a' as int);
+---------------------------------------------+
| CAST((parse_json('{"a": 1}')->'a') AS INT) |
+---------------------------------------------+
|                                           1 |
+---------------------------------------------+

-- Преобразование JSON-строки в значение VARCHAR.
mysql> select cast(parse_json('"star"') as varchar);
+---------------------------------------+
| cast(parse_json('"star"') AS VARCHAR) |
+---------------------------------------+
| star                                  |
+---------------------------------------+

-- Преобразование JSON-объекта в значение VARCHAR.
mysql> select cast(parse_json('{"star": 1}') as varchar);
+----------------------------------------------+
| cast(parse_json('{"star": 1}') AS VARCHAR) |
+----------------------------------------------+
| {"star": 1}                                  |
+----------------------------------------------+

-- Преобразование JSON-массива в значение VARCHAR.

mysql> select cast(parse_json('[1,2,3]') as varchar);
+----------------------------------------+
| cast(parse_json('[1,2,3]') AS VARCHAR) |
+----------------------------------------+
| [1, 2, 3]                              |
+----------------------------------------+
```

Пример 2: Преобразование SQL-значения в JSON-значение.

```
-- Преобразование значения INT в JSON-значение.
mysql> select cast(1 as json);
+-----------------+
| cast(1 AS JSON) |
+-----------------+
| 1               |
+-----------------+

-- Преобразование значения VARCHAR в JSON-значение.
mysql> select cast("star" as json);
+----------------------+
| cast('star' AS JSON) |
+----------------------+
| "star"               |
+----------------------+

-- Преобразование значения BOOLEAN в JSON-значение.
mysql> select cast(true as json);
+--------------------+
| cast(TRUE AS JSON) |
+--------------------+
| true               |
+--------------------+
```

Get_json_bool

Разбирает и получает булево значение из указанного пути в JSON-строке.

> **ПОДСКАЗКА**
>
> Все JSON-функции и операторы перечислены в навигации и на обзорной странице
>
> Ускорьте ваши запросы с помощью генерируемых столбцов

Если формат `json_str` или `json_path` недопустим, или если соответствующее содержимое не может быть найдено, эта функция вернёт NULL.

Эта функция поддерживается начиная с версии 3.3.

Синтаксис

```
BOOLEAN get_json_bool(VARCHAR json_str, VARCHAR json_path)
```

Параметры

- `json_str`: JSON-строка. Поддерживаемый тип данных - VARCHAR.
- `json_path`: JSON-путь. Поддерживаемый тип данных - VARCHAR.
  - `json_path` должен начинаться с `$` и использовать `.` в качестве разделителя пути. Если путь включает `.`, его можно заключить в пару `"`.
  - `[ ]` используется как индекс массива, который начинается с 0.

Примеры

1. Получение значения, ключ которого равен "k1". Значение равно `true`, и возвращается `1`.

```
MySQL > SELECT get_json_bool('{"k1":true, "k2":"false"}', "$.k1");
+-----------------------------------------------------+
| get_json_bool('{"k1":true, "k2":"false"}', '$.k1') |
+-----------------------------------------------------+
|                                                   1 |
+-----------------------------------------------------+
```

2. Получение второго элемента в массиве, ключ которого равен "my.key". Второй элемент равен `false`, и возвращается `0`.

```
SELECT get_json_bool('{"k1":"v1", "my.key":[true, false, 3]}',
'$."my.key"[1]');
+-----------------------------------------------------------------------
----+
| get_json_bool('{"k1":"v1", "my.key":[true, false, 3]}', '$."my.key"
[1]') |
+-----------------------------------------------------------------------
----+
|
0 |
+-----------------------------------------------------------------------
----+
```

3. Получение первого элемента в массиве, путь которого `k1.key -> k2`. Первый элемент равен `false`, и возвращается `0`.

```
MYSQL > SELECT get_json_bool('{"k1.key":{"k2":[false, true]}}',
'$."k1.key".k2[0]');
+-----------------------------------------------------------------------
+
| get_json_bool('{"k1.key":{"k2":[false, true]}}', '$."k1.key".k2[0]')
|
+-----------------------------------------------------------------------
+
|                                                                     0
|
+-----------------------------------------------------------------------
+
```

Ключевые слова

GET_JSON_BOOL,GET,JSON,BOOL

Get_json_double

Эта функция анализирует и получает значение с плавающей точкой из указанного пути в JSON-строке. json_path должен начинаться с `$` и использовать `.` в качестве разделителя пути.

ПОДСКАЗКА

Все JSON-функции и операторы перечислены в навигации и на обзорной странице

Ускорьте ваши запросы с помощью генерируемых столбцов

Если путь включает `.`, его можно заключить в `"` и `"`.

`[ ]` используется как индекс массива, который начинается с 0.

Содержимое пути не должно содержать `"`, `[` и `]`.

Если формат json_string или json_path неправильный, эта функция вернёт NULL.

Синтаксис

```
DOUBLE get_json_double(VARCHAR json_str, VARCHAR json_path)
```

Примеры

1. Получение значения, ключ которого равен "k1"

```
MySQL > SELECT get_json_double('{"k1":1.3, "k2":"2"}', "$.k1");
+-------------------------------------------------+
| get_json_double('{"k1":1.3, "k2":"2"}', '$.k1') |
+-------------------------------------------------+
|                                             1.3 |
+-------------------------------------------------+
```

2. Получение второго элемента в массиве, ключ которого равен "my.key"

```
MySQL > SELECT get_json_double('{"k1":"v1", "my.key":[1.1, 2.2, 3.3]}',
'$."my.key"[1]');
+-----------------------------------------------------------------------
-----+
| get_json_double('{"k1":"v1", "my.key":[1.1, 2.2, 3.3]}', '$."my.key"
[1]') |
+-----------------------------------------------------------------------
-----+
|
2.2 |
+-----------------------------------------------------------------------
-----+
```

3. Получение первого элемента в массиве, путь которого k1.key -> k2.

```
MySQL > SELECT get_json_double('{"k1.key":{"k2":[1.1, 2.2]}}',
'$."k1.key".k2[0]');
+-----------------------------------------------------------------------+
| get_json_double('{"k1.key":{"k2":[1.1, 2.2]}}', '$."k1.key".k2[0]') |
+-----------------------------------------------------------------------+
|                                                                   1.1 |
+-----------------------------------------------------------------------+
```

Ключевые слова

GET_JSON_DOUBLE,GET,JSON,DOUBLE

Get_json_int

Эта функция анализирует и получает целочисленное значение из указанного пути в JSON-строке.

> **ПОДСКАЗКА**
>
> Все JSON-функции и операторы перечислены в навигации и на обзорной странице
>
> Ускорьте ваши запросы с помощью генерируемых столбцов

json_path должен начинаться с `$` и использовать `.` в качестве разделителя пути.

Если путь включает `.`, его можно заключить в `"` и `"`.

`[ ]` используется как индекс массива, который начинается с 0.

Содержимое пути не должно содержать `"`, `[` и `]`.

Если формат json_string или json_path неправильный, эта функция вернёт NULL.

Синтаксис

```
BIGINT get_json_int(VARCHAR json_str, VARCHAR json_path)
```

Примеры

1. Получение значения, ключ которого равен "k1"

```
MySQL > SELECT get_json_int('{"k1":1, "k2":"2"}', "$.k1");
+---------------------------------------------+
| get_json_int('{"k1":1, "k2":"2"}', '$.k1') |
+---------------------------------------------+
|                                           1 |
+---------------------------------------------+
```

2. Получение второго элемента в массиве, ключ которого равен "my.key"

```
MySQL > SELECT get_json_int('{"k1":"v1", "my.key":[1, 2, 3]}',
'$."my.key"[1]');
+-------------------------------------------------------------------+
| get_json_int('{"k1":"v1", "my.key":[1, 2, 3]}', '$."my.key"[1]') |
+-------------------------------------------------------------------+
|                                                                 2 |
+-------------------------------------------------------------------+
```

3. Получение первого элемента в массиве, путь которого k1.key -> k2.

```
MySQL > SELECT get_json_int('{"k1.key":{"k2":[1, 2]}}',
'$."k1.key".k2[0]');
+-----------------------------------------------------------------+
| get_json_int('{"k1.key":{"k2":[1, 2]}}', '$."k1.key".k2[0]') |
+-----------------------------------------------------------------+
|                                                               1 |
+-----------------------------------------------------------------+
```

Ключевые слова

GET_JSON_INT,GET,JSON,INT

Get_json_string,get_json_object

Анализирует и извлекает строки из указанного пути (`json_path`) в JSON-строке. Если формат `json_string` или `json_path` неверен, или если соответствующее значение не найдено, функция возвращает NULL.

> **ПОДСКАЗКА**
>
> Все JSON-функции и операторы перечислены в навигации и на обзорной странице
>
> Ускорьте ваши запросы с помощью генерируемых столбцов

Псевдоним — get_json_object.

Синтаксис

```
VARCHAR get_json_string(VARCHAR json_str, VARCHAR json_path)
```

Параметры

- `json_str`: JSON-строка. Поддерживаемый тип данных — VARCHAR.
- `json_path`: JSON-путь. Поддерживаемый тип данных — VARCHAR. `json_path` начинается с `$` и использует `.` в качестве разделителя пути. `[ ]` используется для индексов массива, которые начинаются с 0. Например, `$."my.key"[1]` означает получение второго значения из элемента `my.key`.

Возвращаемое значение

Возвращает значение типа VARCHAR. Если соответствующий объект не найден, возвращается NULL.

Примеры

Пример 1: Получить значение с ключом `k1`.

```
SELECT get_json_string('{"k1":"v1", "k2":"v2"}', "$.k1");
```

```
+-----------------------------------------------------+
| get_json_string('{"k1":"v1", "k2":"v2"}', '$.k1') |
+-----------------------------------------------------+
| v1                                                  |
+-----------------------------------------------------+
```

Пример 2: Получить значение с ключом `a` из первого элемента.

```
SELECT get_json_object('[{"a":"123", "b": "456"},{"a":"23", "b": "56"}]',
'$[0].a');
```

```
+-----------------------------------------------------------------------------
----+
| get_json_object('[{"a":"123", "b": "456"},{"a":"23", "b": "56"}]',
'$[0].a') |
+-----------------------------------------------------------------------------
----+
| 123
|
+-----------------------------------------------------------------------------
----+
```

Пример 3: Получить второй элемент массива с ключом `my.key`

```
SELECT get_json_string('{"k1":"v1", "my.key":["e1", "e2", "e3"]}',
'$."my.key"[1]');
```

```
+-----------------------------------------------------------------------------
----+
| get_json_string('{"k1":"v1", "my.key":["e1", "e2", "e3"]}', '$."my.key"
[1]') |
+-----------------------------------------------------------------------------
----+
| e2
|
+-----------------------------------------------------------------------------
----+
```

Пример 4: Получить первый элемент массива по пути `k1.key -> k2`.

```
SELECT get_json_string('{"k1.key":{"k2":["v1", "v2"]}}',
'$."k1.key".k2[0]');
```

```
+-----------------------------------------------------------------------+
| get_json_string('{"k1.key":{"k2":["v1", "v2"]}}', '$."k1.key".k2[0]') |
+-----------------------------------------------------------------------+
| v1                                                                    |
+-----------------------------------------------------------------------+
```

Пример 5: Получить все значения с ключом `k1` из массива.

```
SELECT get_json_string('[{"k1":"v1"}, {"k2":"v2"}, {"k1":"v3"},
{"k1":"v4"}]', '$.[*].k1');
```

```
+-----------------------------------------------------------------------------
-----------+
| get_json_string('[{"k1":"v1"}, {"k2":"v2"}, {"k1":"v3"}, {"k1":"v4"}]',
'$.[*].k1') |
+-----------------------------------------------------------------------------
-----------+
| ["v1", "v3", "v4"]
|
+-----------------------------------------------------------------------------
-----------+
1 row in set (0.01 sec)
```

Keyword

GET_JSON_STRING,GET,JSON,STRING

Json_contains

Проверяет, содержит ли JSON-документ определённое значение или поддокумент. Если целевой JSON-документ содержит искомое JSON-значение, функция JSON_CONTAINS возвращает `1`. В противном случае функция JSON_CONTAINS возвращает `0`.

> **ПОДСКАЗКА**
>
> Все JSON-функции и операторы перечислены в навигации и на обзорной странице

Синтаксис

```
json_contains(json_target, json_candidate)
```

Параметры

- `json_target`: выражение, представляющее целевой JSON-документ. Документ может быть JSON-столбцом или JSON-объектом, созданным функцией-конструктором JSON, такой как PARSE_JSON.

- `json_candidate`: выражение, представляющее искомое JSON-значение или поддокумент для поиска в целевом документе. Значение может быть JSON-столбцом или JSON-объектом, созданным функцией-конструктором JSON, такой как PARSE_JSON.

Возвращаемое значение

Возвращает значение типа BOOLEAN.

Примечания по использованию

- Для скалярных значений (строки, числа, логические значения, null) функция возвращает true, если значения равны.
- Для JSON-объектов функция возвращает true, если целевой объект содержит все пары ключ-значение из искомого объекта.

- Для JSON-массивов функция возвращает true, если целевой массив содержит все элементы из искомого массива, или если искомое значение является отдельным значением, содержащимся в целевом массиве.
- Функция выполняет глубокую проверку вложенности для вложенных структур.

Примеры

Пример 1: Проверить, содержит ли JSON-объект определённую пару ключ-значение.

```
mysql> SELECT json_contains(PARSE_JSON('{"a": 1, "b": 2}'),
PARSE_JSON('{"a": 1}'));

       -> 1
```

Пример 2: Проверить, содержит ли JSON-объект несуществующую пару ключ-значение.

```
mysql> SELECT json_contains(PARSE_JSON('{"a": 1, "b": 2}'),
PARSE_JSON('{"c": 3}'));

       -> 0
```

Пример 3: Проверить, содержит ли JSON-массив определённые элементы.

```
mysql> SELECT json_contains(PARSE_JSON('[1, 2, 3, 4]'), PARSE_JSON('[2,
3]'));

       -> 1
```

Пример 4: Проверить, содержит ли JSON-массив одиночное скалярное значение.

```
mysql> SELECT json_contains(PARSE_JSON('[1, 2, 3, 4]'), PARSE_JSON('2'));

       -> 1
```

Пример 5: Проверить, содержит ли JSON-массив несуществующие элементы.

```
mysql> SELECT json_contains(PARSE_JSON('[1, 2, 3, 4]'), PARSE_JSON('[5,
6]'));

       -> 0
```

Пример 6: Проверка вложенности для вложенных JSON-структур.

```
mysql> SELECT json_contains(PARSE_JSON('{"users": [{"id": 1, "name":
"John"}, {"id": 2, "name": "Jane"}]}'),
                            PARSE_JSON('{"users": [{"id": 1}]}'));

       -> 0
```

Примечание: В последнем примере результат равен 0, потому что проверка вложенности массивов требует точного соответствия элементов, а не частичного соответствия объектов внутри массивов.

Json_each

Раскрывает внешние элементы JSON-объекта в набор пар ключ-значение, хранящихся в двух столбцах, и возвращает таблицу, состоящую из одной строки для каждого элемента.

> **ПОДСКАЗКА**
>
> Все JSON-функции и операторы перечислены в навигации и на обзорной странице
>
> Ускорьте ваши запросы с помощью генерируемых столбцов

Синтаксис

```
json_each(json_object_expr)
```

Параметры

`json_object_expr`: выражение, представляющее JSON-объект. Объект может быть JSON-столбцом или JSON-объектом, созданным функцией-конструктором JSON, такой как PARSE_JSON.

Возвращаемое значение

Возвращает два столбца: один с именем key и один с именем value. Столбец key хранит значения VARCHAR, а столбец value хранит значения JSON.

Примечания по использованию

Функция json_each является табличной функцией, которая возвращает таблицу. Возвращаемая таблица представляет собой набор результатов, состоящий из нескольких строк. Поэтому в предложении FROM необходимо использовать lateral join для соединения возвращаемой таблицы с исходной таблицей. Lateral join является обязательным, но ключевое слово LATERAL необязательно. Функцию json_each нельзя использовать в предложении SELECT.

Примеры

```
-- В качестве примера используется таблица tj. В таблице tj столбец j
является JSON-объектом.
mysql> SELECT * FROM tj;
+------+------------------+
| id   | j                |
+------+------------------+
|    1 | {"a": 1, "b": 2} |
|    3 | {"a": 3}         |
+------+------------------+

-- Раскрыть столбец j таблицы tj в два столбца по ключу и значению для
получения набора результатов, состоящего из нескольких строк. В этом
примере ключевое слово LATERAL используется для соединения набора
результатов с таблицей tj.

mysql> SELECT * FROM tj, LATERAL json_each(j);
+------+------------------+------+-------+
| id   | j                | key  | value |
+------+------------------+------+-------+
|    1 | {"a": 1, "b": 2} | a    | 1     |
|    1 | {"a": 1, "b": 2} | b    | 2     |
|    3 | {"a": 3}         | a    | 3     |
+------+------------------+------+-------+
```

Json_exists

Проверяет, содержит ли JSON-объект элемент, который можно найти по выражению `json_path`. Если элемент существует, функция JSON_EXISTS возвращает `1`. В противном случае функция JSON_EXISTS возвращает `0`.

> **ПОДСКАЗКА**
>
> Все JSON-функции и операторы перечислены в навигации и на обзорной странице
>
> Ускорьте ваши запросы с помощью генерируемых столбцов

Синтаксис

```
json_exists(json_object_expr, json_path)
```

Параметры

- `json_object_expr`: выражение, представляющее JSON-объект. Объект может быть JSON-столбцом или JSON-объектом, созданным функцией-конструктором JSON, такой как PARSE_JSON.
- `json_path`: выражение, представляющее путь к элементу в JSON-объекте. Значение этого параметра — строка. Дополнительную информацию о синтаксисе JSON-путей, поддерживаемом Selena, см. в разделе Обзор JSON-функций и операторов.

Возвращаемое значение

Возвращает значение типа BOOLEAN.

Примеры

Пример 1: Проверить, содержит ли указанный JSON-объект элемент, который можно найти по выражению `'$.a.b'`. В этом примере элемент существует в JSON-объекте. Поэтому функция json_exists возвращает `1`.

```
mysql> SELECT json_exists(PARSE_JSON('{"a": {"b": 1}}'), '$.a.b') ;

       -> 1
```

Пример 2: Проверить, содержит ли указанный JSON-объект элемент, который можно найти по выражению `'$.a.c'`. В этом примере элемент не существует в JSON-объекте. Поэтому функция json_exists возвращает `0`.

```
mysql> SELECT json_exists(PARSE_JSON('{"a": {"b": 1}}'), '$.a.c') ;

       -> 0
```

Пример 3: Проверить, содержит ли указанный JSON-объект элемент, который можно найти по выражению `'$.a[2]'`. В этом примере JSON-объект, который является массивом с именем a, содержит элемент с индексом 2. Поэтому функция json_exists возвращает `1`.

```
mysql> SELECT json_exists(PARSE_JSON('{"a": [1,2,3]}'), '$.a[2]') ;

       -> 1
```

Пример 4: Проверить, содержит ли указанный JSON-объект элемент, который можно найти по выражению `'$.a[3]'`. В этом примере JSON-объект, который является массивом с именем a, не содержит элемент с индексом 3. Поэтому функция json_exists возвращает `0`.

```
mysql> SELECT json_exists(PARSE_JSON('{"a": [1,2,3]}'), '$.a[3]') ;

       -> 0
```

Json_keys

Возвращает ключи верхнего уровня из JSON-объекта в виде JSON-массива или, если указан `path`, ключи верхнего уровня по указанному пути.

ПОДСКАЗКА

Все JSON-функции и операторы перечислены в навигации и на обзорной странице

Ускорьте ваши запросы с помощью генерируемых столбцов

Синтаксис

```
json_keys(json_doc[, path])
```

Параметры

`json_doc`: обязательный. JSON-документ, из которого нужно вернуть ключи. Должен быть JSON-объектом.

`path`: необязательный. Путь обычно начинается с `$` и использует `.` в качестве разделителя пути. `[]` используется для индексов массива, которые начинаются с 0.

Возвращаемое значение

Возвращает JSON-массив.

Возвращается пустой массив, если JSON-объект пуст.

Возвращается `NULL`, если JSON-документ не является JSON-объектом или путь не указывает на значение в документе.

Если JSON-документ является массивом, вложенным с JSON-объектом, можно использовать параметр `path` для получения ключей из этого объекта.

Примеры

Пример 1: Возвращает пустой массив, потому что входной JSON-объект пуст.

```
select json_keys('{}');
+-----------------+
| json_keys('{}') |
+-----------------+
| []              |
+-----------------+
```

Пример 2: Возвращает ключи JSON-объекта.

```
select json_keys('{"a": 1, "b": 2, "c": 3}');
+----------------+
| json_keys('1') |
+----------------+
|["a", "b", "c"] |
+----------------+
```

Пример 3: Возвращает ключи JSON-объекта, соответствующего указанному пути.

```
select json_keys('{"a": 1, "b": 2, "c": {"d": 3, "e": 4, "f": 5}}', '$.c');
+-----------------------------------------------------------------------+
| json_keys('{"a": 1, "b": 2, "c": {"d": 3, "e": 4, "f": 5}}', '$.c') |
+-----------------------------------------------------------------------+
| ["d", "e", "f"]                                                       |
+-----------------------------------------------------------------------+
```

Пример 4: Путь не существует.

```
select json_keys('{"a": 1, "b": 2, "c": {"d": 3, "e": 4, "f": 5}}', '$.e');
+-----------------------------------------------------------------------+
| json_keys('{"a": 1, "b": 2, "c": {"d": 3, "e": 4, "f": 5}}', '$.e') |
+-----------------------------------------------------------------------+
| NULL                                                                  |
+-----------------------------------------------------------------------+
```

Пример 5: JSON-документ не является JSON-объектом.

```
select json_keys('[1, 2, {"a": 1, "b": 2}]');
+---------------------------------------+
| json_keys('[1, 2, {"a": 1, "b": 2}]') |
+---------------------------------------+
| NULL                                  |
+---------------------------------------+
```

Пример 6: JSON-документ является массивом, вложенным с JSON-объектом. Указан путь для получения ключей из этого объекта.

```
select json_keys('[0, 1, {"a": 1, "b": 2}]', '$[2]');
+-----------------------------------------------+
| json_keys('[0, 1, {"a": 1, "b": 2}]', '$[2]') |
+-----------------------------------------------+
| ["a", "b"]                                    |
+-----------------------------------------------+
```

Json_length

Возвращает длину JSON-документа. Если указан путь, функция возвращает длину значения, найденного по этому пути.

> **ПОДСКАЗКА**
>
> Все JSON-функции и операторы перечислены в навигации и на обзорной странице
>
> Ускорьте ваши запросы с помощью генерируемых столбцов

Длина документа определяется по следующим правилам:

- Длина скалярного значения равна 1. Например, длина `1`, `"a"`, `true`, `false` и `null` равна 1.
- Длина массива равна количеству элементов массива. Например, длина `[1, 2]` равна 2.
- Длина объекта равна количеству членов объекта. Например, длина `{"a": 1}` равна 1.
- Длина вложенных массивов или объектов не учитывается. Например, длина `{"a": [1, 2]}` равна 1, потому что вложенный массив `[1, 2]` не включается в подсчёт длины.

Синтаксис

```
json_length(json_doc[, path])
```

Параметры

`json_doc`: обязательный, JSON-документ, для которого нужно вернуть длину.

`path`: необязательный. Используется для возврата длины значения внутри документа. Путь обычно начинается с `$` и использует `.` в качестве разделителя пути. `[]` используется для индексов массива, которые начинаются с 0.

Возвращаемое значение

Возвращает значение типа INT.

Возвращается ошибка, если JSON-документ не является допустимым документом.

Возвращается 0 в любом из следующих случаев:

- `path` не указывает на значение в документе.
- Путь не является допустимым выражением пути.
- Путь содержит подстановочные знаки `*` или `**`.

Примеры

Пример 1: Возвращает длину скалярного значения.

```
select json_length('1');
+------------------+
| json_length('1') |
+------------------+
|                1 |
+------------------+
```

Пример 2: Возвращает длину пустого объекта.

```
select json_length('{}');
+-------------------+
| json_length('{}') |
+-------------------+
|                 0 |
+-------------------+
```

Пример 3: Возвращает длину объекта с данными.

```
select json_length('{"Name": "Homer"}');
+-----------------------------------+
| json_length('{"Name": "Homer"}') |
+-----------------------------------+
|                                 1 |
+-----------------------------------+
```

Пример 4: Возвращает длину JSON-массива.

```
select json_length('[1, 2, 3]');
+---------------------------+
| json_length('[1, 2, 3]') |
+---------------------------+
|                         3 |
+---------------------------+
```

Пример 5: Возвращает длину JSON-массива, в котором один элемент имеет вложенный массив.

Вложенный массив `[3, 4]` не включается в подсчёт длины.

```
select json_length('[1, 2, [3, 4]]');
+-------------------------------+
| json_length('[1, 2, [3, 4]]') |
+-------------------------------+
|                             3 |
+-------------------------------+
```

Пример 6: Возвращает длину объекта, указанного путём `$.Person`.

```
SET @file = '{
    "Person": {
       "Name": "Homer",
       "Age": 39,
       "Hobbies": ["Eating", "Sleeping"]
    }
 }';
select json_length(@file, '$.Person') 'Result';
```

Пример 7: Возвращает длину значения, указанного путём `$.y`.

```
select json_length('{"x": 1, "y": [1, 2]}', '$.y');
+----------------------------------------------+
| json_length('{"x": 1, "y": [1, 2]}', '$.y') |
+----------------------------------------------+
|                                            2 |
+----------------------------------------------+
```

Json_query

Запрашивает значение элемента, который можно найти по выражению `json_path` в JSON-объекте, и возвращает JSON-значение.

> **ПОДСКАЗКА**
>
> Все JSON-функции и операторы перечислены в навигации и на обзорной странице
>
> Ускорьте ваши запросы с помощью генерируемых столбцов

Синтаксис

```
json_query(json_object_expr, json_path)
```

Параметры

- `json_object_expr`: выражение, представляющее JSON-объект. Объект может быть JSON-столбцом или JSON-объектом, созданным функцией-конструктором JSON, такой как PARSE_JSON.

- `json_path`: выражение, представляющее путь к элементу в JSON-объекте. Значение этого параметра — строка. Информацию о синтаксисе JSON-путей, поддерживаемом Selena, см. в разделе Обзор JSON-функций и операторов.

Возвращаемое значение

Возвращает JSON-значение.

> Если элемент не существует, функция json_query возвращает SQL-значение `NULL`.

Примеры

Пример 1: Запросить значение элемента, который можно найти по выражению `'$.a.b'` в указанном JSON-объекте. В этом примере функция json_query возвращает JSON-значение `1`.

```
mysql> SELECT json_query(PARSE_JSON('{"a": {"b": 1}}'), '$.a.b') ;

       -> 1
```

Пример 2: Запросить значение элемента, который можно найти по выражению `'$.a.c'` в указанном JSON-объекте. В этом примере элемент не существует. Поэтому функция json_query возвращает SQL-значение `NULL`.

```
mysql> SELECT json_query(PARSE_JSON('{"a": {"b": 1}}'), '$.a.c') ;

       -> NULL
```

Пример 3: Запросить значение элемента, который можно найти по выражению `'$.a[2]'` в указанном JSON-объекте. В этом примере JSON-объект, который является массивом с именем a, содержит элемент с индексом 2, и значение элемента равно 3. Поэтому функция JSON_QUERY возвращает JSON-значение `3`.

```
mysql> SELECT json_query(PARSE_JSON('{"a": [1,2,3]}'), '$.a[2]') ;

       -> 3
```

Пример 4: Запросить элемент, который можно найти по выражению `'$.a[3]'` в указанном JSON-объекте. В этом примере JSON-объект, который является массивом с именем a, не содержит элемент с индексом 3. Поэтому функция json_query возвращает SQL-значение `NULL`.

```
mysql> SELECT json_query(PARSE_JSON('{"a": [1,2,3]}'), '$.a[3]') ;

       -> NULL
```

Json_remove

Удаляет данные из JSON-документа по одному или нескольким указанным JSON-путям и возвращает изменённый JSON-документ.

> **ПОДСКАЗКА**
>
> Все JSON-функции и операторы перечислены в навигации и на обзорной странице

Синтаксис

```
json_remove(json_object_expr, json_path[, json_path] ...)
```

Параметры

- `json_object_expr`: выражение, представляющее JSON-объект. Объект может быть JSON-столбцом или JSON-объектом, созданным функцией-конструктором JSON, такой как PARSE_JSON.

- `json_path`: один или несколько выражений, представляющих пути к элементам в JSON-объекте, которые должны быть удалены. Значение каждого параметра — строка. Информацию о синтаксисе JSON-путей, поддерживаемом Selena, см. в разделе Обзор JSON-функций и операторов.

Возвращаемое значение

Возвращает JSON-документ с удалёнными указанными путями.

> - Если путь не существует в JSON-документе, он игнорируется.
> - Если указан недопустимый путь, он игнорируется.

> - Если все пути недопустимы или не существуют, возвращается исходный JSON-документ без изменений.

Примеры

Пример 1: Удалить один ключ из JSON-объекта.

```
mysql> SELECT json_remove('{"a": 1, "b": [10, 20, 30]}', '$.a');

       -> {"b": [10, 20, 30]}
```

Пример 2: Удалить несколько ключей из JSON-объекта.

```
mysql> SELECT json_remove('{"a": 1, "b": [10, 20, 30], "c": "test"}',
'$.a', '$.c');

       -> {"b": [10, 20, 30]}
```

Пример 3: Удалить элементы массива из JSON-объекта.

```
mysql> SELECT json_remove('{"a": 1, "b": [10, 20, 30]}', '$.b[1]');

       -> {"a": 1, "b": [10, 30]}
```

Пример 4: Удалить свойства вложенного объекта.

```
mysql> SELECT json_remove('{"a": {"x": 1, "y": 2}, "b": 3}', '$.a.x');

       -> {"a": {"y": 2}, "b": 3}
```

Пример 5: Попытка удалить несуществующие пути (игнорируются).

```
mysql> SELECT json_remove('{"a": 1, "b": 2}', '$.c', '$.d');

       -> {"a": 1, "b": 2}
```

Пример 6: Удалить несколько путей, включая несуществующие.

```
mysql> SELECT json_remove('{"a": 1, "b": 2, "c": 3}', '$.a',
'$.nonexistent', '$.c');

       -> {"b": 2}
```

Примечания по использованию

- Функция `json_remove` соответствует поведению, совместимому с MySQL.
- Недопустимые JSON-пути молча игнорируются, а не вызывают ошибки.
- Функция поддерживает удаление нескольких путей за одну операцию, что более эффективно, чем несколько отдельных операций.
- В настоящее время функция поддерживает простое удаление ключей объекта (например, `$.key`). Поддержка сложных вложенных путей и удаления элементов массива может быть ограничена в текущей реализации.

Json_string

Преобразует JSON-объект в JSON-строку

> **ПОДСКАЗКА**
>
> Все JSON-функции и операторы перечислены в навигации и на обзорной странице
>
> Ускорьте ваши запросы с помощью генерируемых столбцов

Синтаксис

```
json_string(json_object_expr)
```

Параметры

- `json_object_expr`: выражение, представляющее JSON-объект. Объект может быть JSON-столбцом или JSON-объектом, созданным функцией-конструктором JSON, такой как PARSE_JSON.

Возвращаемое значение

Возвращает значение типа VARCHAR.

Примеры

Пример 1: Преобразование JSON-объекта в JSON-строку

```
select json_string('{"Name": "Alice"}');
+-----------------------------------+
| json_string('{"Name": "Alice"}') |
+-----------------------------------+
| {"Name": "Alice"}                 |
+-----------------------------------+
```

Пример 2: Преобразование результата PARSE_JSON в JSON-строку

```
select json_string(parse_json('{"Name": "Alice"}'));
+-----------------------------------+
| json_string('{"Name": "Alice"}') |
+-----------------------------------+
| {"Name": "Alice"}                 |
+-----------------------------------+
```

To_json

Преобразует значение типа Map или Struct в JSON-строку. Если входное значение равно NULL, возвращается NULL.

> **ПОДСКАЗКА**
>
> Все JSON-функции и операторы перечислены в навигации и на обзорной странице
>
> Ускорьте ваши запросы с помощью генерируемых столбцов

Если вы хотите преобразовать значения других типов данных, см. cast.

Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
to_json(any_value)
```

Параметры

`any_value`: выражение типа Map или Struct, которое вы хотите преобразовать. Если входное значение недопустимо, возвращается ошибка. Значение в каждой паре ключ-значение Map или Struct может быть nullable. См. последний пример.

Возвращаемое значение

Возвращает JSON-значение.

Примеры

```
select to_json(map{1:'a',2:'b'});
+---------------------------+
| to_json(map{1:'a',2:'b'}) |
+---------------------------+
| {"1": "a", "2": "b"}      |
+---------------------------+

select to_json(row('asia','eu'));
+--------------------------------+
| to_json(row('asia', 'eu'))     |
+--------------------------------+
| {"col1": "asia", "col2": "eu"} |
+--------------------------------+

select to_json(map('a', named_struct('b', 1)));
+----------------------------------------+
| to_json(map{'a':named_struct('b', 1)}) |
+----------------------------------------+
| {"a": {"b": 1}}                        |
+----------------------------------------+

select to_json(named_struct("k1", cast(null as string), "k2", "v2"));
+------------------------------------------------------------------------+
| to_json(named_struct('k1', CAST(NULL AS VARCHAR(65533)), 'k2', 'v2')) |
+------------------------------------------------------------------------+
| {"k1": null, "k2": "v2"}                                               |
+------------------------------------------------------------------------+
```

См. также

- Тип данных Map
- Тип данных Struct

- Функции для работы с Map
- Функции для работы со Struct

JSON-операторы

Selena поддерживает следующие JSON-операторы сравнения: `<`, `<=`, `>`, `>=`, `=` и `!=`. Вы можете использовать эти операторы для запроса JSON-данных. Однако Selena не позволяет использовать `IN` для запроса JSON-данных.

- > Операнды оператора должны быть JSON-значениями.
- > Если один операнд оператора является JSON-значением, а другой - нет, операнд, не являющийся JSON-значением, преобразуется в JSON-значение во время арифметической операции. Дополнительную информацию о правилах преобразования см. в CAST.

ПОДСКАЗКА

Все JSON-функции и операторы перечислены в навигации и на обзорной странице

Ускорьте ваши запросы с помощью генерируемых столбцов

Арифметические правила

JSON-операторы соответствуют следующим арифметическим правилам:

- Когда операнды оператора являются JSON-значениями одного и того же типа данных:
  - Если оба операнда являются JSON-значениями базового типа данных, такого как NUMBER, STRING или BOOLEAN, оператор выполняет арифметическую операцию в соответствии с арифметическими правилами для базового типа данных.

> Примечание: Если оба операнда являются числами, но один является значением DOUBLE, а другой - значением INT, оператор преобразует значение INT в значение DOUBLE.

- Если оба операнда являются JSON-значениями составного типа данных, такого как OBJECT или ARRAY, оператор сортирует ключи в операндах в алфавитном порядке на основе последовательности ключей в первом операнде, а затем сравнивает значения ключей между операндами.

Пример 1:

Первый операнд - `{"a": 1, "c": 2}`, второй операнд - `{"b": 1, "a": 2}`. В этом примере оператор сравнивает значения ключей `a` между операндами. Значение ключа `a` в первом операнде равно `1`, тогда как значение ключа `a` во втором операнде равно `2`. Значение `1` меньше значения `2`. Поэтому оператор делает вывод, что первый операнд `{"a": 1, "c": 2}` меньше второго операнда `{"b": 1, "a": 2}`.

```
mysql> SELECT PARSE_JSON('{"a": 1, "c": 2}') < PARSE_JSON('{"b": 1, "a": 2}
');

       -> 1
```

Пример 2:

Первый операнд - `{"a": 1, "c": 2}`, второй операнд - `{"b": 1, "a": 1}`. В этом примере оператор сначала сравнивает значения ключей `a` между операндами. Значения ключей `a` в операндах оба равны `1`. Затем оператор сравнивает значения ключей `c` в операндах. Второй операнд не содержит ключ `c`. Поэтому оператор делает вывод, что первый операнд `{"a": 1, "c": 2}` больше второго операнда `{"b": 1, "a": 1}`.

```
mysql> SELECT PARSE_JSON('{"a": 1, "c": 2}') < PARSE_JSON('{"b": 1, "a":
1}');

       -> 0
```

- Когда операнды оператора являются JSON-значениями двух различных типов данных, оператор сравнивает операнды в соответствии со следующими арифметическими правилами: NULL < BOOLEAN < ARRAY < OBJECT < DOUBLE < INT < STRING.

```
mysql> SELECT PARSE_JSON('"a"') < PARSE_JSON('{"a": 1, "c": 2}');

       -> 0
```

Обзор JSON-функций и операторов

В этом разделе представлен обзор функций-конструкторов JSON, функций запросов и обработки, операторов и выражений путей, поддерживаемых Selena.

> **ПОДСКАЗКА**
>
> Ускорьте ваши запросы с помощью генерируемых столбцов

Функции-конструкторы JSON

Функции-конструкторы JSON используются для создания JSON-данных, таких как JSON-объекты и JSON-массивы.

Функция	Описание	Пример	Возвращаемое значение
json_object	Преобразует одну или несколько пар ключ-значение в JSON-объект, состоящий из пар ключ-значение, отсортированных по ключу в словарном порядке.	`SELECT JSON_OBJECT('Daniel Smith', 26, 'Lily Smith', 25);`	`{"Daniel Smith": 26, "Lily Smith": 25}`
json_array	Преобразует каждый элемент SQL-массива в JSON-значение и возвращает JSON-массив, состоящий из этих JSON-значений.	`SELECT JSON_ARRAY(1, 2, 3);`	`[1,2,3]`
parse_json	Преобразует строку в JSON-значение.	`SELECT PARSE_JSON('{"a": 1}');`	`{"a": 1}`

Функции запросов и обработки JSON

Функции запросов и обработки JSON используются для запроса и обработки JSON-данных. Например, можно использовать выражение пути для поиска элемента в JSON-объекте.

Функция	Описание	Пример	Возвращаемое значение
функция стрелки	Запрашивает элемент, который можно найти по выражению пути в JSON-объекте.	`SELECT parse_json('{"a": {"b": 1}}') -> '$.a.b';`	`1`
cast	Преобразует данные между типом JSON и типом SQL.	`SELECT CAST(1 AS JSON);`	`1`
get_json_doub	Анализирует и извлекает значение с плавающей точкой из указанного пути в JSON-строке.	`SELECT get_json_double('{"k1":1.3, "k2":"2"}',`	`1.3`

Функция	Описание	Пример	Возвращаемое значение
le		`"$.k1");`	
get_json_int	Анализирует и извлекает целочисленное значение из указанного пути в JSON-строке.	`SELECT get_json_int('{"k1":1, "k2":"2"}', "$.k1");`	`1`
get_json_string	Анализирует и извлекает строки из указанного пути в JSON-строке.	`SELECT get_json_string('{"k1":"v1", "k2":"v2"}', "$.k1");`	`v1`
json_query	Запрашивает значение элемента, который можно найти по выражению пути в JSON-объекте.	`SELECT JSON_QUERY('{"a": 1}', '$.a');`	`1`
json_remove	Удаляет данные из JSON-документа по одному или нескольким указанным JSON-путям.	`SELECT JSON_REMOVE('{"a": 1, "b": [10, 20, 30]}', '$.a', '$.b[1]');`	`{"b": [10, 30]}`
json_each	Раскрывает элементы верхнего уровня JSON-объекта в пары ключ-значение.	`SELECT * FROM tj_test, LATERAL JSON_EACH(j);`	`!` json_each
json_exists	Проверяет, содержит ли JSON-объект элемент, который можно найти по выражению пути. Если элемент существует, возвращает 1. Если элемент не существует, возвращает 0.	`SELECT JSON_EXISTS('{"a": 1}', '$.a');`	`1`
json_keys	Возвращает ключи верхнего уровня из JSON-объекта в виде JSON-массива или, если указан путь, ключи верхнего уровня по этому пути.	`SELECT JSON_KEYS('{"a": 1, "b": 2, "c": 3}');`	`["a", "b", "c"]`
json_length	Возвращает длину JSON-документа.	`SELECT json_length('{"Name": "Alice"}');`	`1`
json_string	Преобразует JSON-объект в JSON-строку	`SELECT json_string(parse_json('{"Name": "Alice"}'));`	`{"Name": "Alice"}`

Операторы JSON

Selena поддерживает следующие операторы сравнения JSON: `<`, `<=`, `>`, `>=`, `=` и `!=`. Вы можете использовать эти операторы для запроса JSON-данных. Однако использование `IN` для запроса JSON-данных не допускается. Дополнительную информацию об операторах JSON см. в разделе Операторы JSON.

Выражения путей JSON

Вы можете использовать выражение пути JSON для запроса элемента в JSON-объекте. Выражения путей JSON имеют тип данных STRING. В большинстве случаев они используются с различными JSON-функциями, такими как JSON_QUERY. В Selena

выражения путей JSON не полностью соответствуют спецификациям путей SQL/JSON. Информацию о синтаксисе путей JSON, поддерживаемом в Selena, см. в следующей таблице, в которой используется следующий JSON-объект в качестве примера.

```
{
    "people": [{
        "name": "Daniel",
        "surname": "Smith"
    }, {
        "name": "Lily",
        "surname": "Smith",
        "active": true
    }]
}
```

Символ пути JSON	Описание	Пример пути JSON	Возвращаемое значение
`$`	Обозначает корневой JSON-объект.	`'$'`	`{ "people": [ { "name": "Daniel", "surname": "Smith" }, { "name": "Lily", "surname": Smith, "active": true } ] }`
`.`	Обозначает дочерний JSON-объект.	`' $.people'`	`[ { "name": "Daniel", "surname": "Smith" }, { "name": "Lily", "surname": Smith, "active": true } ]`
`[]`	Обозначает один или несколько индексов массива. `[n]` обозначает n-й элемент в массиве. Индексы начинаются с 0. Selena 2.5 поддерживает запросы к многомерным массивам, например, `["Lucy", "Daniel"], ["James", "Smith"]`. Для запроса элемента "Lucy" можно использовать `$.people[0][0]`.	`'$.people [0]'`	`{ "name": "Daniel", "surname": "Smith" }`
`[*]`	Обозначает все элементы в массиве.	`'$.people[*].name'`	`["Daniel", "Lily"]`
`[start: end]`	Обозначает подмножество элементов из массива. Подмножество задаётся интервалом `[start, end]`, который исключает элемент, обозначенный конечным индексом.	`'$.people[0: 1].name'`	`["Daniel"]`

2.18 Pattern matching

Like

Проверяет, соответствует ли заданное выражение указанному шаблону нечёткого поиска. Если да, возвращается 1. В противном случае возвращается 0. Возвращается NULL, если любой из входных параметров равен NULL.

LIKE обычно используется вместе с символами, такими как знак процента (%) и подчёркивание (_). `%` соответствует 0, 1 или более символам. `_` соответствует любому одному символу.

Синтаксис

```
BOOLEAN like(VARCHAR expr, VARCHAR pattern);
```

Параметры

- `expr`: строковое выражение. Поддерживаемый тип данных — VARCHAR.
- `pattern`: шаблон для сопоставления. Поддерживаемый тип данных — VARCHAR.

Возвращаемое значение

Возвращает значение типа BOOLEAN.

Примеры

```
mysql> select like("star","star");
+----------------------+
| like('star', 'star') |
+----------------------+
|                    1 |
+----------------------+

mysql> select like("selena","star%");
+----------------------+
| like('star', 'star') |
+----------------------+
|                    1 |
+----------------------+

mysql> select like("selena","star_");
+----------------------------+
| like('selena', 'star_') |
+----------------------------+
|                          0 |
+----------------------------+
```

Regexp

Проверяет, соответствует ли заданное выражение регулярному выражению, указанному в `pattern`. Если да, возвращается 1. В противном случае возвращается 0. Возвращается NULL, если любой из входных параметров равен NULL.

regexp() поддерживает более сложные условия сопоставления, чем like().

Синтаксис

```
BOOLEAN regexp(VARCHAR expr, VARCHAR pattern);
```

Параметры

- `expr`: строковое выражение. Поддерживаемый тип данных — VARCHAR.
- `pattern`: шаблон для сопоставления. Поддерживаемый тип данных — VARCHAR.

Возвращаемое значение

Возвращает значение типа BOOLEAN.

Примеры

```
mysql> select regexp("abc123","abc*");
+--------------------------+
| regexp('abc123', 'abc*') |
+--------------------------+
|                        1 |
+--------------------------+
1 row in set (0.06 sec)

select regexp("abc123","xyz*");
+--------------------------+
| regexp('abc123', 'xyz*') |
+--------------------------+
|                        0 |
+--------------------------+
```

Ключевые слова

regexp, regular

Regexp_count

Подсчитывает количество вхождений шаблона в строку. Возвращает количество вхождений шаблона регулярного выражения в целевую строку.

Синтаксис

```
INT regexp_count(VARCHAR str, VARCHAR pattern)
```

Параметры

- `str`: строка для поиска. Поддерживаемый тип данных — VARCHAR. Если входное значение равно NULL, возвращается NULL.
- `pattern`: шаблон регулярного выражения для поиска. Поддерживаемый тип данных — VARCHAR. Если шаблон равен NULL, возвращается NULL.

Возвращаемое значение

Возвращает INT, представляющий количество вхождений. Возвращает 0, если совпадения не найдены или если входная строка пуста.

Примеры

ПОДСКАЗКА

Для быстрого старта с Selena см. раздел Быстрый старт.

```
-- Подсчёт вхождений цифр
SELECT regexp_count('abc123def456', '[0-9]');
```

```
+---------------------------------------+
| regexp_count('abc123def456', '[0-9]') |
+---------------------------------------+
|                                     6 |
+---------------------------------------+
```

```
-- Подсчёт вхождений точек
SELECT regexp_count('test.com test.net test.org', '\\.');
```

```
+----------------------------------------------------+
| regexp_count('test.com test.net test.org', '\\.') |
+----------------------------------------------------+
|                                                  3 |
+----------------------------------------------------+
```

```
-- Подсчёт вхождений последовательностей пробельных символов
SELECT regexp_count('a b  c   d', '\\s+');
```

```
+----------------------------------------+
| regexp_count('a b  c   d', '\\s+')     |
+----------------------------------------+
|                                      3 |
+----------------------------------------+
```

```
-- Подсчёт вхождений повторяющегося шаблона
SELECT regexp_count('abababababab', 'ab');
```

```
+-------------------------------------+
| regexp_count('abababababab', 'ab')  |
+-------------------------------------+
|                                   5 |
+-------------------------------------+
```

```
-- Использование с NULL и пустыми значениями
SELECT
  regexp_count('', '.') AS empty_str,
  regexp_count(NULL, '.') AS null_str,
  regexp_count('abc', NULL) AS null_pattern;
```

```
+------------+----------+--------------+
| empty_str  | null_str | null_pattern |
+------------+----------+--------------+
|          0 |     NULL |         NULL |
+------------+----------+--------------+
```

```
-- Подсчёт вхождений Unicode/китайских символов
SELECT regexp_count('abc中文def', '[\\p{Han}]+');
```

```
+-----------------------------------------------+
| regexp_count('abc中文def', '[\\p{Han}]+')     |
+-----------------------------------------------+
|                                             1 |
+-----------------------------------------------+
```

С данными таблицы

```
CREATE TABLE sample_text (
  str VARCHAR(65533) NULL,
  regex VARCHAR(65533) NULL
);

INSERT INTO sample_text VALUES
  ('abc123def456', '[0-9]'),
  ('test.com test.net test.org', '\\.'),
  ('a b  c   d', '\\s+'),
  ('abababababab', 'ab');

SELECT str, regex, regexp_count(str, regex) AS count
FROM sample_text
ORDER BY str;
```

```
+-----------------------------+------+-------+
| str                         | regex| count |
+-----------------------------+------+-------+
| a b  c   d                  | \s+  |     3 |
| abc123def456                | [0-9]|     6 |
| abababababab                | ab   |     5 |
| test.com test.net test.org  | \.   |     3 |
+-----------------------------+------+-------+
```

Keyword

REGEXP_COUNT,REGEXP,COUNT

Regexp_extract

Эта функция возвращает первую совпадающую подстроку в целевом значении, которая соответствует шаблону регулярного выражения. Она извлекает элемент в позиции pos, соответствующий шаблону. Шаблон должен полностью соответствовать некоторым частям str, чтобы функция могла вернуть части, которые необходимо сопоставить в шаблоне. Если совпадения не найдены, возвращается пустая строка.

Синтаксис

```
VARCHAR regexp_extract(VARCHAR str, VARCHAR pattern, int pos)
```

Примеры

> **ПОДСКАЗКА**
>
> Для быстрого старта с Selena см. раздел Быстрый старт.

Исходные данные:

```
SELECT HourlySkyConditions FROM quickstart.weatherdata
WHERE HourlySkyConditions LIKE '%OVC%'
LIMIT 10;
```

```
+---------------------+
| HourlySkyConditions |
+---------------------+
| OVC:08 110          |
| OVC:08 120          |
| OVC:08 120          |
| OVC:08 30           |
| OVC:08 29           |
| OVC:08 27           |
| OVC:08 26           |
| OVC:08 22           |
| OVC:08 23           |
| OVC:08 22           |
+---------------------+
10 rows in set (0.03 sec)
```

Вернуть два набора цифр, следующих за строкой `OVC:`

```
SELECT regexp_extract(HourlySkyConditions, 'OVC:(\\d+ \\d+)', 1) FROM
quickstart.weatherdata
WHERE HourlySkyConditions LIKE '%OVC%'
LIMIT 10;
```

```
+-----------------------------------------------------------+
| regexp_extract(HourlySkyConditions, 'OVC:(\\d+ \\d+)', 1) |
+-----------------------------------------------------------+
| 08 110                                                    |
| 08 120                                                    |
| 08 120                                                    |
| 08 30                                                     |
| 08 29                                                     |
| 08 27                                                     |
| 08 26                                                     |
| 08 22                                                     |
| 08 23                                                     |
| 08 22                                                     |
+-----------------------------------------------------------+
10 rows in set (0.01 sec)
```

Вернуть только второй набор цифр, следующих за строкой `OVC:`

```
SELECT regexp_extract(HourlySkyConditions, 'OVC:(\\d+) (\\d+)', 2) FROM
quickstart.weatherdata WHERE HourlySkyConditions LIKE '%OVC%' LIMIT 10;
```

```
+-------------------------------------------------------------+
| regexp_extract(HourlySkyConditions, 'OVC:(\\d+) (\\d+)', 2) |
+-------------------------------------------------------------+
| 110                                                         |
| 120                                                         |
| 120                                                         |
| 30                                                          |
| 29                                                          |
| 27                                                          |
| 26                                                          |
| 22                                                          |
| 23                                                          |
| 22                                                          |
+-------------------------------------------------------------+
10 rows in set (0.01 sec)
```

ПОДСКАЗКА

Тот же результат можно получить, не заключая первый набор цифр в группу захвата `()` и возвращая первую группу:

```
regexp_extract(HourlySkyConditions, 'OVC:\\d+ (\\d+)', 1)
```

Keyword

REGEXP_EXTRACT,REGEXP,EXTRACT

Regexp_extract_all

Извлекает все подстроки из целевой строки (`str`), которые соответствуют шаблону регулярного выражения (`pattern`) и соответствуют индексу группы регулярного выражения, указанному в `pos`. Эта функция возвращает массив.

В регулярных выражениях группы заключаются в круглые скобки () и нумеруются путём подсчёта их открывающих скобок слева направо, начиная с 1. Например, `([[:lower:]]+)C([[:lower:]]+)` соответствует строчным буквам слева или справа от заглавной буквы `C`. Этот шаблон содержит две группы: `([[:lower:]]+)` слева от `C` — первая группа, а `([[:lower:]]+)` справа от `C` — вторая группа.

Шаблон должен полностью соответствовать некоторым частям `str`. Если совпадения не найдены, возвращается пустая строка.

Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
ARRAY<VARCHAR> regexp_extract_all(VARCHAR str, VARCHAR pattern, BIGINT pos)
```

Параметры

- `str`: строка для сопоставления.
- `pattern`: шаблон регулярного выражения, используемый для сопоставления подстрок.

- `pos`: `pattern` может содержать несколько групп. `pos` указывает, какую группу регулярного выражения извлекать.

Возвращаемое значение

Возвращает ARRAY, состоящий из элементов VARCHAR.

Примеры

```
-- Вернуть все буквы, соответствующие группе 1 в шаблоне.
MySQL > SELECT regexp_extract_all('AbCdE', '([[:lower:]]+)C([[:lower:]]+)',
1);
+-----------------------------------------------------------------+
| regexp_extract_all('AbCdE', '([[:lower:]]+)C([[:lower:]]+)', 1)   |
+-----------------------------------------------------------------+
| ['b']                                                             |
+-----------------------------------------------------------------+

-- Вернуть все буквы, соответствующие группе 2 в шаблоне.
MySQL > SELECT regexp_extract_all('AbCdExCeF',
'([[:lower:]]+)C([[:lower:]]+)', 2);
+-------------------------------------------------------------------+
| regexp_extract_all('AbCdExCeF', '([[:lower:]]+)C([[:lower:]]+)', 2) |
+-------------------------------------------------------------------+
| ['d','e']                                                           |
+-------------------------------------------------------------------+
```

Ключевые слова

REGEXP_EXTRACT_ALL,REGEXP,EXTRACT

Regexp_replace

Эта функция использует repl для замены последовательности символов в str, которая соответствует шаблону регулярного выражения.

Синтаксис

```
VARCHAR regexp_replace(VARCHAR str, VARCHAR pattern, VARCHAR repl)
```

Примеры

```
MySQL > SELECT regexp_replace('a b c', " ", "-");
+-----------------------------------+
| regexp_replace('a b c', ' ', '-') |
+-----------------------------------+
| a-b-c                             |
+-----------------------------------+

MySQL > SELECT regexp_replace('a b c','(b)','<\\1>');
+----------------------------------------+
| regexp_replace('a b c', '(b)', '<\1>') |
+----------------------------------------+
| a <b> c                                |
+----------------------------------------+
```

Keyword

REGEXP_REPLACE,REGEXP,REPLACE

2.19 Map

Cardinality

Возвращает количество элементов в значении MAP. MAP — это неупорядоченная коллекция пар ключ-значение, например, `{"a":1, "b":2}`. Одна пара ключ-значение составляет один элемент. `{"a":1, "b":2}` содержит два элемента.

Эта функция поддерживается начиная с версии v1.5.2. Она является псевдонимом map_size().

Синтаксис

```
INT cardinality(any_map)
```

Параметры

`any_map`: значение MAP, из которого вы хотите получить количество элементов.

Возвращаемое значение

Возвращает значение типа INT.

Если входное значение равно NULL, возвращается NULL.

Если ключ или значение в MAP равно NULL, NULL обрабатывается как обычное значение.

Примеры

Запрос данных MAP из нативной таблицы Selena

Начиная с версии v1.5.2, Selena поддерживает определение столбцов MAP при создании таблицы. В этом примере используется таблица `test_map`, которая содержит следующие данные:

```
CREATE TABLE test_map(
    col_int INT,
    col_map MAP<VARCHAR(50),INT>
  )
DUPLICATE KEY(col_int);

INSERT INTO test_map VALUES
(1,map{"a":1,"b":2}),
(2,map{"c":3}),
(3,map{"d":4,"e":5});

SELECT * FROM test_map ORDER BY col_int;
+---------+---------------+
| col_int | col_map       |
+---------+---------------+
|       1 | {"a":1,"b":2} |
|       2 | {"c":3}       |
|       3 | {"d":4,"e":5} |
+---------+---------------+
3 rows in set (0.05 sec)
```

Получить количество элементов в каждой строке столбца `col_map`.

```
select cardinality(col_map) from test_map order by col_int;
+----------------------+
| cardinality(col_map) |
+----------------------+
|                    2 |
|                    1 |
|                    2 |
+----------------------+
3 rows in set (0.05 sec)
```

Запрос данных MAP из озера данных

В этом примере используется таблица Hive `hive_map`, которая содержит следующие данные:

```
SELECT * FROM hive_map ORDER BY col_int;
+---------+---------------+
| col_int | col_map       |
+---------+---------------+
|       1 | {"a":1,"b":2} |
|       2 | {"c":3}       |
|       3 | {"d":4,"e":5} |
+---------+---------------+
```

После создания каталога Hive в вашем cluster вы можете использовать этот каталог и функцию cardinality() для получения количества элементов в каждой строке столбца `col_map`.

```
SELECT cardinality(col_map) FROM hive_map ORDER BY col_int;
+----------------------+
| cardinality(col_map) |
+----------------------+
|                    2 |
|                    1 |
|                    2 |
+----------------------+
3 rows in set (0.05 sec)
```

Distinct_map_keys

Удаляет дублирующиеся ключи из map, поскольку ключи в map должны быть уникальными по семантике. Эта функция сохраняет только последнее значение для идентичных ключей (принцип LAST WIN). Эта функция используется при запросе данных MAP из внешних таблиц, если в map есть дублирующиеся ключи. Внутренние таблицы Selena изначально удаляют дублирующиеся ключи в map.

Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
distinct_map_keys(any_map)
```

Параметры

`any_map`: значение MAP, из которого вы хотите удалить дублирующиеся ключи.

Возвращаемое значение

Возвращает новый map без дублирующихся ключей в каждом map.

Если входное значение равно NULL, возвращается NULL.

Примеры

Пример 1: Простое использование.

```
select distinct_map_keys(map{"a":1,"a":2});
+-------------------------------------+
| distinct_map_keys(map{'a':1,'a':2}) |
+-------------------------------------+
| {"a":2}                             |
+-------------------------------------+
```

Пример 2: Запрос данных MAP из внешних таблиц и удаление дублирующихся ключей из столбца `col_map`.

```
select distinct_map_keys(col_map) as unique, col_map from external_table;
+---------------+---------------+
|      unique   | col_map       |
+---------------+---------------+
|       {"c":2} | {"c":1,"c":2} |
|           NULL|          NULL |
| {"e":4,"d":5} | {"e":4,"d":5} |
+---------------+---------------+
3 rows in set (0.05 sec)
```

Element_at

Возвращает значение для указанного ключа из пары ключ-значение map. Если любой входной параметр равен NULL или если ключ не существует в map, результат — NULL.

Если вы хотите получить элемент из массива, см. element_at.

Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
element_at(any_map, any_key)
```

Параметры

- `any_map`: выражение MAP, из которого нужно получить значения.
- `any_key`: ключ в map.

Возвращаемое значение

Если `any_key` существует в `any_map`, возвращается значение, соответствующее ключу. В противном случае возвращается NULL.

Примеры

```
mysql> select element_at(map{1:3,2:4},1);
+-----------------+
| map{1:3,2:4}[1] |
+-----------------+
|               3 |
+-----------------+

mysql> select element_at(map{1:3,2:4},3);
+-----------------+
| map{1:3,2:4}[3] |
+-----------------+
|            NULL |
+-----------------+

mysql> select element_at(map{'a':1,'b':2},'a');
+-----------------------+
| map{'a':1,'b':2}['a'] |
+-----------------------+
|                     1 |
+-----------------------+
```

Keywords

ELEMENT_AT, MAP

Map_apply

Применяет Lambda-выражение к ключам и значениям исходного Map и создаёт новый Map. Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
MAP map_apply(lambda_func, any_map)
```

Параметры

- `lambda_func`: Lambda-выражение.
- `any_map`: map, к которому применяется Lambda-выражение.

Возвращаемое значение

Возвращает значение map. Типы данных ключей и значений в результирующем map определяются результатом Lambda-выражения.

Если любой входной параметр равен NULL, возвращается NULL.

Если ключ или значение в исходном map равно NULL, NULL обрабатывается как обычное значение.

Lambda-выражение должно иметь два параметра. Первый параметр представляет ключ. Второй параметр представляет значение.

Примеры

В следующем примере используется map_from_arrays() для создания значения map `{1:"ab",3:"cd"}`. Затем Lambda-выражение увеличивает каждый ключ на 1 и вычисляет длину каждого значения. Например, длина "ab" равна 2.

```
mysql> select map_apply((k,v)->(k+1,length(v)), col_map)
from (select map_from_arrays([1,3],["ab","cd"]) as col_map)A;
+-----------------------------------------------+
| map_apply((k, v) -> (k + 1, length(v)), col_map) |
+-----------------------------------------------+
| {2:2,4:2}                                     |
+-----------------------------------------------+
1 row in set (0.01 sec)

mysql> select map_apply((k,v)->(k+1,length(v)), col_map)
from (select map_from_arrays(null,null) as col_map)A;
+-----------------------------------------------+
| map_apply((k, v) -> (k + 1, length(v)), col_map) |
+-----------------------------------------------+
| NULL                                          |
+-----------------------------------------------+
1 row in set (0.02 sec)

mysql> select map_apply((k,v)->(k+1,length(v)), col_map)
from (select map_from_arrays([1,null],["ab","cd"]) as col_map)A;
+-----------------------------------------------+
| map_apply((k, v) -> (k + 1, length(v)), col_map) |
+-----------------------------------------------+
| NULL                                          |
+-----------------------------------------------+
```

Map_concat

Возвращает объединение входных map. Если ключ найден в нескольких map, эта функция сохраняет только последнее значение среди этих map (принцип LAST WIN). Например, `SELECT map_concat(map{1:3},map{1:'4'});` возвращает `{1:"4"}`.

Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
ANY_MAP map_concat(any_map0, any_map1...)
```

Параметры

`any_mapN`: значения map, которые вы хотите объединить. Все map должны иметь общий тип. Если типы данных входных map не совпадают, возвращаемый тип является общим

супертипом входных map.

Возвращаемое значение

Возвращает MAP общего супертипа входных map.

Примеры

```
mysql> SELECT map_concat(map('a',1, 'b',2), map('c',3));
+-----------------------------------------+
| map_concat(map{'a':1,'b':2}, map{'c':3}) |
+-----------------------------------------+
| {"c":3,"a":1,"b":2}                     |
+-----------------------------------------+

mysql> select map_concat(map{1:3},map{'3.323':3});
+--------------------------------------+
| map_concat(map{1:3}, map{'3.323':3}) |
+--------------------------------------+
| {"3.323":3,"1":3}                    |
+--------------------------------------+


mysql> select map_concat(map{1:3},map{1:'4', 3:'5',null:null}, null);
+-----------------------------------------------------------+
| map_concat(map{1:3}, map{1:'4',3:'5',NULL:NULL}, NULL) |
+-----------------------------------------------------------+
| {null:null,1:"4",3:"5"}                                   |
+-----------------------------------------------------------+
```

Map_filter

Фильтрует пары ключ-значение в map, применяя к каждой паре Boolean-массив или Lambda-выражение. Возвращаются пары, которые оцениваются как `true`.

Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
MAP map_filter(any_map, array<boolean>)
MAP map_filter(lambda_func, any_map)
```

- `map_filter(any_map, array<boolean>)`

  Оценивает пары ключ-значение в `any_map` одну за другой в соответствии с `array<boolean>` и возвращает пары ключ-значение, которые оцениваются как `true`.

- `map_filter(lambda_func, any_map)`

  Применяет `lambda_func` к парам ключ-значение в `any_map` одну за другой и возвращает пары ключ-значение, результат которых равен `true`.

Параметры

- `any_map`: значение map.
- `array<boolean>`: Boolean-массив, используемый для оценки значения map.
- `lambda_func`: Lambda-выражение, используемое для оценки значения map.

Возвращаемое значение

Возвращает map, тип данных которого совпадает с `any_map`.

Если `any_map` равен NULL, возвращается NULL. Если `array<boolean>` равен null, возвращается пустой map.

Если ключ или значение в map равен NULL, NULL обрабатывается как обычное значение.

Lambda-выражение должно иметь два параметра. Первый параметр представляет ключ. Второй параметр представляет значение.

Примеры

Использование `array<boolean>`

Следующий пример использует map_from_arrays() для генерации значения map `{1:"ab",3:"cdd",2:null,null:"abc"}`. Затем каждая пара ключ-значение оценивается в соответствии с `array<boolean>`, и возвращается пара, результат которой равен `true`.

```
mysql> select map_filter(col_map, array<boolean>[0,0,0,1,1]) from (select
map_from_arrays([1,3,null,2,null],['ab','cdd',null,null,'abc']) as
col_map)A;
+-----------------------------------------------------+
| map_filter(col_map, ARRAY<BOOLEAN>[0, 0, 0, 1, 1]) |
+-----------------------------------------------------+
| {null:"abc"}                                        |
+-----------------------------------------------------+
1 row in set (0.02 sec)

mysql> select map_filter(null, array<boolean>[0,0,0,1,1]);
+-----------------------------------------------------+
| map_filter(NULL, ARRAY<BOOLEAN>[0, 0, 0, 1, 1]) |
+-----------------------------------------------------+
| NULL                                                |
+-----------------------------------------------------+
1 row in set (0.02 sec)

mysql> select map_filter(col_map, null) from (select
map_from_arrays([1,3,null,2,null],['ab','cdd',null,null,'abc']) as
col_map)A;
+---------------------------+
| map_filter(col_map, NULL) |
+---------------------------+
| {}                        |
+---------------------------+
1 row in set (0.01 sec)
```

Использование Lambda-выражения

Следующий пример использует map_from_arrays() для генерации значения map `{1:"ab",3:"cdd",2:null,null:"abc"}`. Затем каждая пара ключ-значение оценивается в соответствии с Lambda-выражением, и возвращается пара ключ-значение, значение которой не равно null.

```
mysql> select map_filter((k,v) -> v is not null,col_map) from (select
map_from_arrays([1,3,null,2,null],['ab','cdd',null,null,'abc']) as
col_map)A;
+-----------------------------------------------+
| map_filter((k,v) -> v is not null, col_map)    |
+-----------------------------------------------+
| {1:"ab",3:"cdd",null:'abc'}                      |
+-----------------------------------------------+
1 row in set (0.02 sec)
```

Map_from_arrays

Создает значение MAP из заданной пары массивов элементов ключей и элементов значений.

Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
MAP map_from_arrays(ARRAY keys, ARRAY values)
```

Параметры

- `keys`: используется для построения ключей результирующего MAP. Убедитесь, что элементы ключей уникальны.
- `values`: используется для построения значений результирующего MAP.

Возвращаемое значение

Возвращает MAP, который построен из входных ключей и значений.

- Ключи и значения должны иметь одинаковую длину, в противном случае будет возвращена ошибка.
- Если ключ или значение равны NULL, эта функция возвращает NULL.
- Возвращаемое значение MAP имеет уникальные ключи.

Примеры

```
select map_from_arrays([1, 2], ['Star', 'Rocks']);
+---------------------------------------------+
| map_from_arrays([1, 2], ['Star', 'Rocks']) |
+---------------------------------------------+
| {1:"Star",2:"Rocks"}                        |
+---------------------------------------------+
```

```
select map_from_arrays([1, 2], NULL);
+-------------------------------+
| map_from_arrays([1, 2], NULL) |
+-------------------------------+
| NULL                          |
+-------------------------------+

select map_from_arrays([1,3,null,2,null],['ab','cdd',null,null,'abc']);
+-------------------------------------------------------------------------------
+
| map_from_arrays([1, 3, NULL, 2, NULL], ['ab', 'cdd', NULL, NULL, 'abc'])
|
+-------------------------------------------------------------------------------
+
| {1:"ab",3:"cdd",2:null,null:"abc"}
|
+-------------------------------------------------------------------------------
+
```

Map_keys

Возвращает массив всех ключей в указанном map.

Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
map_keys(any_map)
```

Параметры

`any_map`: значение MAP, из которого вы хотите получить ключи.

Возвращаемое значение

Возвращаемое значение имеет формат `array<keyType>`. Тип элемента в массиве соответствует типу ключа в map.

Если входное значение равно NULL, возвращается NULL. Если ключ или значение в MAP равны NULL, NULL обрабатывается как обычное значение и включается в результат.

Примеры

Запрос MAP-данных из нативной таблицы Selena

Начиная с версии v1.5.2, Selena поддерживает определение MAP-столбцов при создании таблицы. В этом примере используется таблица `test_map`, которая содержит следующие данные:

```
CREATE TABLE test_map(
    col_int INT,
    col_map MAP<VARCHAR(50),INT>
  )
DUPLICATE KEY(col_int);

INSERT INTO test_map VALUES
(1,map{"a":1,"b":2}),
(2,map{"c":3}),
(3,map{"d":4,"e":5});

SELECT * FROM test_map ORDER BY col_int;
+---------+---------------+
| col_int | col_map       |
+---------+---------------+
|       1 | {"a":1,"b":2} |
|       2 | {"c":3}       |
|       3 | {"d":4,"e":5} |
+---------+---------------+
3 rows in set (0.05 sec)
```

Получение всех ключей из каждой строки столбца `col_map`.

```
select map_keys(col_map) from test_map order by col_int;
+-------------------+
| map_keys(col_map) |
+-------------------+
| ["a","b"]         |
| ["c"]             |
| ["d","e"]         |
+-------------------+
3 rows in set (0.05 sec)
```

Запрос MAP-данных из data lake

В этом примере используется Hive-таблица `hive_map`, которая содержит следующие данные:

```
SELECT * FROM hive_map ORDER BY col_int;
+---------+---------------+
| col_int | col_map       |
+---------+---------------+
|       1 | {"a":1,"b":2} |
|       2 | {"c":3}       |
|       3 | {"d":4,"e":5} |
+---------+---------------+
```

После создания Hive catalog в вашем кластере вы можете использовать этот catalog и функцию map_keys() для получения всех ключей из каждой строки столбца `col_map`.

```
select map_keys(col_map) from hive_map order by col_int;
+-------------------+
| map_keys(col_map) |
+-------------------+
| ["a","b"]         |
| ["c"]             |
| ["d","e"]         |
+-------------------+
3 rows in set (0.05 sec)
```

Map_size

Возвращает количество элементов в значении MAP. MAP представляет собой неупорядоченную коллекцию пар ключ-значение, например, `{"a":1, "b":2}`. Одна пара ключ-значение составляет один элемент, например, `{"a":1, "b":2}` содержит два элемента.

Эта функция имеет псевдоним cardinality(). Поддерживается начиная с версии v1.5.2.

Синтаксис

```
INT map_size(any_map)
```

Параметры

`any_map`: значение MAP, из которого вы хотите получить количество элементов.

Возвращаемое значение

Возвращает значение типа INT.

Если входное значение равно NULL, возвращается NULL.

Если ключ или значение в MAP равны NULL, NULL обрабатывается как обычное значение.

Примеры

Запрос MAP-данных из нативной таблицы Selena

Начиная с версии v1.5.2, Selena поддерживает определение MAP-столбцов при создании таблицы. В этом примере используется таблица `test_map`, которая содержит следующие данные:

```
CREATE TABLE test_map(
    col_int INT,
    col_map MAP<VARCHAR(50),INT>
  )
DUPLICATE KEY(col_int);

INSERT INTO test_map VALUES
(1,map{"a":1,"b":2}),
(2,map{"c":3}),
(3,map{"d":4,"e":5});

SELECT * FROM test_map ORDER BY col_int;
+---------+---------------+
| col_int | col_map       |
+---------+---------------+
|       1 | {"a":1,"b":2} |
|       2 | {"c":3}       |
|       3 | {"d":4,"e":5} |
+---------+---------------+
3 rows in set (0.05 sec)
```

Получение количества элементов в каждой строке столбца `col_map`.

```
select map_size(col_map) from test_map order by col_int;
+-------------------+
| map_size(col_map) |
+-------------------+
|                 2 |
|                 1 |
|                 2 |
+-------------------+
3 rows in set (0.05 sec)
```

Запрос MAP-данных из data lake

В этом примере используется Hive-таблица `hive_map`, которая содержит следующие данные:

```
SELECT * FROM hive_map ORDER BY col_int;
+---------+---------------+
| col_int | col_map       |
+---------+---------------+
|       1 | {"a":1,"b":2} |
|       2 | {"c":3}       |
|       3 | {"d":4,"e":5} |
+---------+---------------+
```

После создания Hive catalog в вашем кластере вы можете использовать этот catalog и функцию map_size() для получения количества элементов в каждой строке столбца `col_map`.

```
select map_size(col_map) from hive_map order by col_int;
+-------------------+
| map_size(col_map) |
+-------------------+
|                 2 |
|                 1 |
|                 2 |
+-------------------+
3 rows in set (0.05 sec)
```

Map_values

Возвращает массив всех значений в указанном map.

Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
map_values(any_map)
```

Параметры

`any_map`: значение MAP, из которого вы хотите получить значения.

Возвращаемое значение

Возвращаемое значение имеет формат `array<valueType>`. Тип элемента в массиве соответствует типу значения в map.

Если входное значение равно NULL, возвращается NULL. Если ключ или значение в MAP равны NULL, NULL обрабатывается как обычное значение и включается в результат.

Примеры

Запрос MAP-данных из нативной таблицы Selena

Начиная с версии v1.5.2, Selena поддерживает определение MAP-столбцов при создании таблицы. В этом примере используется таблица `test_map`, которая содержит следующие данные:

```
CREATE TABLE test_map(
    col_int INT,
    col_map MAP<VARCHAR(50),INT>
  )
DUPLICATE KEY(col_int);

INSERT INTO test_map VALUES
(1,map{"a":1,"b":2}),
(2,map{"c":3}),
(3,map{"d":4,"e":5});

SELECT * FROM test_map ORDER BY col_int;
+---------+---------------+
| col_int | col_map       |
+---------+---------------+
|       1 | {"a":1,"b":2} |
|       2 | {"c":3}       |
|       3 | {"d":4,"e":5} |
+---------+---------------+
3 rows in set (0.05 sec)
```

Получение всех значений из каждой строки столбца `col_map`.

```
select map_values(col_map) from test_map order by col_int;
+---------------------+
| map_values(col_map) |
+---------------------+
| [1,2]               |
| [3]                 |
| [4,5]               |
+---------------------+
3 rows in set (0.04 sec)
```

Запрос MAP-данных из data lake

В этом примере используется Hive-таблица `hive_map`, которая содержит следующие данные:

```
SELECT * FROM hive_map ORDER BY col_int;
+---------+---------------+
| col_int | col_map       |
+---------+---------------+
|       1 | {"a":1,"b":2} |
|       2 | {"c":3}       |
|       3 | {"d":4,"e":5} |
+---------+---------------+
```

После создания Hive catalog в вашем кластере вы можете использовать этот catalog и функцию map_values() для получения всех значений из каждой строки столбца `col_map`.

```
select map_values(col_map) from hive_map order by col_int;
+---------------------+
| map_values(col_map) |
+---------------------+
| [1,2]               |
| [3]                 |
| [4,5]               |
+---------------------+
3 rows in set (0.04 sec)
```

Transform_keys

Преобразует ключи в map с использованием Lambda-выражения и создает новый ключ для каждой записи в map.

Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
MAP transform_keys(lambda_func, any_map)
```

`lambda_func` также может быть размещена после `any_map`:

```
MAP transform_keys(any_map, lambda_func)
```

Параметры

- `any_map`: Map.
- `lambda_func`: Lambda-выражение, которое вы хотите применить к `any_map`.

Возвращаемое значение

Возвращает значение map, где типы данных ключей определяются результатом Lambda-выражения, а типы данных значений совпадают со значениями в `any_map`.

Если любой входной параметр равен NULL, возвращается NULL.

Если ключ или значение в исходном map равны NULL, NULL обрабатывается как обычное значение.

Lambda-выражение должно иметь два параметра. Первый параметр представляет ключ. Второй параметр представляет значение.

Примеры

Следующий пример использует map_from_arrays для генерации значения map `{1:"ab",3:"cdd",2:null,null:"abc"}`. Затем Lambda-выражение применяется к каждому ключу для увеличения ключа на 1.

```
mysql> select transform_keys((k,v)->(k+1), col_map) from (select
map_from_arrays([1,3,null,2,null],['ab','cdd',null,null,'abc']) as
col_map)A;
+-----------------------------------------+
| transform_keys((k, v) -> k + 1, col_map) |
+-----------------------------------------+
| {2:"ab",4:"cdd",3:null,null:"abc"}       |
+-----------------------------------------+
```

Transform_values

Преобразует значения в map с использованием Lambda-выражения и создает новое значение для каждой записи в map.

Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
MAP transform_values(lambda_func, any_map)
```

`lambda_func` также может быть размещена после `any_map`:

```
MAP transform_values(any_map, lambda_func)
```

Параметры

- `any_map`: Map.
- `lambda_func`: Lambda-выражение, которое вы хотите применить к `any_map`.

Возвращаемое значение

Возвращает значение map, где типы данных значений определяются результатом Lambda-выражения, а типы данных ключей совпадают с ключами в `any_map`.

Если любой входной параметр равен NULL, возвращается NULL.

Если ключ или значение в исходном map равны NULL, NULL обрабатывается как обычное значение.

Lambda-выражение должно иметь два параметра. Первый параметр представляет ключ. Второй параметр представляет значение.

Примеры

Следующий пример использует map_from_arrays для генерации значения map `{1:"ab",3:"cdd",2:null,null:"abc"}`. Затем Lambda-выражение применяется к каждому значению map. Первый пример изменяет значение каждой пары ключ-значение на 1. Второй пример изменяет значение каждой пары ключ-значение на null.

```
mysql> select transform_values((k,v)->1, col_map) from (select
map_from_arrays([1,3,null,2,null],['ab','cdd',null,null,'abc']) as
col_map)A;
+----------------------------------------+
| transform_values((k, v) -> 1, col_map) |
+----------------------------------------+
| {1:1,3:1,2:1,null:1}                   |
+----------------------------------------+
1 row in set (0.02 sec)

mysql> select transform_values((k,v)->null, col_map) from (select
map_from_arrays([1,3,null,2,null],['ab','cdd',null,null,'abc']) as
col_map)A;
+---------------------------------------------+
| transform_values((k, v) -> NULL, col_map)   |
+---------------------------------------------+
| {1:null,3:null,2:null,null:null} |
+---------------------------------------------+
1 row in set (0.01 sec)
```

2.20 Math

Abs

Возвращает абсолютное значение числового значения `x`. Если входное значение NULL, возвращается NULL.

Синтаксис

```
ABS(x);
```

Параметры

`x`: числовое значение или выражение.

Поддерживаемые типы данных: DOUBLE, FLOAT, LARGEINT, BIGINT, INT, SMALLINT, TINYINT, DECIMALV2, DECIMAL32, DECIMAL64, DECIMAL128.

Возвращаемое значение

Тип данных возвращаемого значения совпадает с типом `x`.

Примеры

```
mysql> select abs(-1);
+---------+
| abs(-1) |
+---------+
|       1 |
+---------+
1 row in set (0.00 sec)
```

Ключевые слова

abs, absolute

Acos

Вычисляет арккосинус аргумента.

Синтаксис

```
DOUBLE acos(DOUBLE arg)
```

Параметры

`arg`: Можно указать только числовое значение. Эта функция преобразует числовое значение в значение DOUBLE перед вычислением арккосинуса значения.

Возвращаемое значение

Возвращает значение типа данных DOUBLE.

Примечания по использованию

Если указано нечисловое значение, эта функция возвращает `NULL`.

Примеры

```
mysql> select acos(-1);
+-------------------+
| acos(-1)          |
+-------------------+
| 3.141592653589793 |
+-------------------+

mysql> select acos(0);
+--------------------+
| acos(0)            |
+--------------------+
| 1.5707963267948966 |
+--------------------+

mysql> select acos(1);
+---------+
| acos(1) |
+---------+
|       0 |
+---------+

mysql> select acos("");
+----------+
| acos('') |
+----------+
|     NULL |
+----------+
```

Ключевое слово

ACOS

Asin

Вычисляет арксинус аргумента.

Синтаксис

```
DOUBLE asin(DOUBLE arg)
```

Параметры

`arg`: Можно указать только числовое значение. Эта функция преобразует числовое значение в значение DOUBLE перед вычислением арксинуса значения.

Возвращаемое значение

Возвращает значение типа данных DOUBLE.

Примечания по использованию

Если указано нечисловое значение, эта функция возвращает `NULL`.

Примеры

```
mysql> select asin(1);
+--------------------+
| asin(1)            |
+--------------------+
| 1.5707963267948966 |
+--------------------+

mysql> select asin(-0.5);
+----------------------+
| asin(-0.5)           |
+----------------------+
| -0.5235987755982989 |
+----------------------+

mysql> select asin(0);
+---------+
| asin(0) |
+---------+
|       0 |
+---------+

mysql> select asin("");
+----------+
| asin('') |
+----------+
|     NULL |
+----------+
```

Ключевое слово

ASIN

Atan

Вычисляет арктангенс аргумента.

Синтаксис

```
DOUBLE atan(DOUBLE arg)
```

Параметры

`arg`: Можно указать только числовое значение. Эта функция преобразует числовое значение в значение DOUBLE перед вычислением арктангенса значения.

Возвращаемое значение

Возвращает значение типа данных DOUBLE.

Примечания по использованию

Если указано нечисловое значение, эта функция возвращает `NULL`.

Примеры

```
mysql> select atan(1);
+--------------------+
| atan(1)            |
+--------------------+
| 0.7853981633974483 |
+--------------------+

mysql> select atan(0);
+---------+
| atan(0) |
+---------+
|       0 |
+---------+

mysql> select atan(-1);
+----------------------+
| atan(-1)             |
+----------------------+
| -0.7853981633974483 |
+----------------------+

mysql> select atan("");
+----------+
| atan('') |
+----------+
|     NULL |
+----------+
```

Ключевое слово

ATAN

Atan2

Возвращает арктангенс `x`, деленного на `y`, то есть арктангенс `x/y`. Знаки двух параметров используются для определения квадранта результата.

Возвращаемое значение находится в диапазоне [-π, π].

Синтаксис

```
ATAN2(x,y);
```

Параметры

`x`: Поддерживаемый тип данных - DOUBLE.

`y`: Поддерживаемый тип данных - DOUBLE.

Возвращаемое значение

Возвращает значение типа DOUBLE. Возвращает NULL, если `x` или `y` равно NULL.

Примеры

```
mysql> select atan2(-0.8,2);
+---------------------+
| atan2(-0.8, 2)      |
+---------------------+
| -0.3805063771123649 |
+---------------------+
1 row in set (0.01 sec)
```

Bin

Преобразует входное значение `arg` в двоичное представление.

Синтаксис

```
bin(arg)
```

Параметр

`arg`: входное значение, которое необходимо преобразовать в двоичное представление. Поддерживается тип данных BIGINT.

Возвращаемое значение

Возвращает значение типа данных VARCHAR.

Примеры

```
mysql> select bin(3);
+--------+
| bin(3) |
+--------+
| 11     |
+--------+
1 row in set (0.02 sec)
```

Cbrt

Вычисляет кубический корень из аргумента.

Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
DOUBLE cbrt(DOUBLE arg)
```

Параметры

`arg`: Можно указать только числовое значение. Функция преобразует числовое значение в значение типа DOUBLE перед вычислением кубического корня.

Возвращаемое значение

Возвращает значение типа данных DOUBLE. Если указано нечисловое значение, функция возвращает `NULL`.

Примеры

```
mysql> select cbrt(8);
+---------+
| cbrt(8) |
+---------+
|       2 |
+---------+

mysql> select cbrt(-8);
+----------+
| cbrt(-8) |
+----------+
|       -2 |
+----------+

mysql> select cbrt(0);
+---------+
| cbrt(0) |
+---------+
|       0 |
+---------+

mysql> select cbrt("");
+----------+
| cbrt('') |
+----------+
|     NULL |
+----------+
```

Keywords

cbrt, cube root

Ceil, dceil

Возвращает значения из входного параметра `arg`, округленные до ближайшего равного или большего целого числа.

Синтаксис

```
ceil(arg)
```

Параметр

`arg` поддерживает тип данных DOUBLE.

Возвращаемое значение

Возвращает значение типа данных BIGINT.

Примеры

```
mysql> select ceil(3.14);
+------------+
| ceil(3.14) |
+------------+
|          4 |
+------------+
1 row in set (0.15 sec)
```

Ceiling

Возвращает значения из входного параметра `arg`, округленные до ближайшего равного или большего целого числа.

Синтаксис

```
ceiling(arg)
```

Параметр

`arg` поддерживает тип данных DOUBLE.

Возвращаемое значение

Возвращает значение типа данных BIGINT.

Примеры

```
mysql> select ceiling(3.14);
+---------------+
| ceiling(3.14) |
+---------------+
|             4 |
+---------------+
1 row in set (0.00 sec)
```

Conv

Преобразует число `x` из одной системы счисления в другую и возвращает результат в виде строкового значения.

Синтаксис

```
CONV(x,y,z);
```

Параметры

- `x`: число для преобразования. Поддерживаются типы VARCHAR и BIGINT.
- `y`: исходная система счисления. Поддерживается тип TINYINT.
- `z`: целевая система счисления. Поддерживается тип TINYINT.

Возвращаемое значение

Возвращает значение типа данных VARCHAR.

Примеры

Преобразовать число 8 из системы счисления 10 в систему счисления 2:

```
mysql> select conv(8,10,2);
+----------------+
| conv(8, 10, 2) |
+----------------+
| 1000           |
+----------------+
1 row in set (0.00 sec)
```

Cos

Вычисляет косинус аргумента.

Синтаксис

```
DOUBLE cos(DOUBLE arg)
```

Параметры

`arg`: Можно указать только числовое значение. Функция преобразует числовое значение в значение типа DOUBLE перед вычислением косинуса.

Возвращаемое значение

Возвращает значение типа данных DOUBLE.

Примечания по использованию

Если указано нечисловое значение, функция возвращает `NULL`.

© ООО «Платформа Данных Селена», 2026

Примеры

```
mysql> select cos(-1);
+--------------------+
| cos(-1)            |
+--------------------+
| 0.5403023058681398 |
+--------------------+

mysql> select cos(0);
+--------+
| cos(0) |
+--------+
|      1 |
+--------+

mysql> select cos(1);
+--------------------+
| cos(1)             |
+--------------------+
| 0.5403023058681398 |
+--------------------+

mysql> select cos("");
+---------+
| cos('') |
+---------+
|    NULL |
+---------+
```

Keyword

COS

Cos_similarity

Измеряет схожесть двух векторов путём вычисления косинуса угла между ними. Угол формируется направлением векторов, при этом разница в их величине игнорируется.

Схожесть находится в диапазоне от -1 до 1. Меньшие углы между векторами указывают на большую косинусную схожесть.

- Если два вектора имеют одинаковое направление, они образуют угол 0 градусов и косинусная схожесть равна 1.
- Перпендикулярные векторы образуют угол 90 градусов и косинусная схожесть равна 0.
- Противоположные векторы образуют угол 180 градусов и косинусная схожесть равна -1.

`cosine_similarity` обычно используется для оценки схожести текста и видео.

Эта функция нормализует векторы перед измерением косинусной схожести. Если входные векторы уже были нормализованы, вы можете использовать cosine_similarity_norm.

Синтаксис

```
cosine_similarity(a, b)
```

Параметры

`a` и `b` - это векторы для сравнения. Они должны иметь одинаковую размерность. Поддерживаемый тип данных - `Array<float>`. Два массива должны иметь одинаковое количество элементов. В противном случае возвращается ошибка.

Возвращаемое значение

Возвращает значение FLOAT в диапазооне [-1, 1]. Если любой входной параметр равен null или недействителен, возвращается ошибка.

Примеры

1. Создайте таблицу для хранения векторов и вставьте данные в эту таблицу.

```
CREATE TABLE t1_similarity
(id int, data array<float>)
DISTRIBUTED BY HASH(id);

INSERT INTO t1_similarity VALUES
(1, array<float>[0.1, 0.2, 0.3]),
(2, array<float>[0.2, 0.1, 0.3]),
(3, array<float>[0.3, 0.2, 0.1]);
```

2. Вычислите схожесть каждой строки в столбце `data` по сравнению с массивом `[0.1, 0.2, 0.3]` и выведите результат в порядке убывания.

```
SELECT id, data, cosine_similarity([0.1, 0.2, 0.3], data) as dist
FROM t1_similarity
ORDER BY dist DESC;

+------+---------------+-----------+
| id   | data          | dist      |
+------+---------------+-----------+
|    1 | [0.1,0.2,0.3] | 0.9999999 |
|    2 | [0.2,0.1,0.3] | 0.9285713 |
|    3 | [0.3,0.2,0.1] | 0.7142856 |
+------+---------------+-----------+
```

Cos_similarity_norm

Измеряет схожесть двух нормализованных векторов путём вычисления косинуса угла между ними. Угол формируется направлением векторов, при этом разница в их величине игнорируется. Эта функция предполагает, что входные векторы уже были нормализованы. Если вам нужно нормализовать векторы перед вычислением косинусной схожести, используйте cosine_similarity.

Схожесть находится в диапазоне от -1 до 1. Меньшие углы между векторами указывают на большую косинусную схожесть.

- Если два вектора имеют одинаковое направление, они образуют угол 0 градусов и косинусная схожесть равна 1.
- Перпендикулярные векторы образуют угол 90 градусов и косинусная схожесть равна 0.
- Противоположные векторы образуют угол 180 градусов и косинусная схожесть равна -1.

Синтаксис

```
cosine_similarity_norm(a, b)
```

Параметры

`a` и `b` - это векторы для сравнения. Они должны иметь одинаковую размерность. Поддерживаемый тип данных - `Array<float>`. Два массива должны иметь одинаковое количество элементов. В противном случае возвращается ошибка.

Возвращаемое значение

Возвращает значение FLOAT в диапазоне [-1, 1]. Если любой входной параметр равен null или недействителен, возвращается ошибка.

Примеры

1. Создайте таблицу для хранения векторов и вставьте данные в эту таблицу.

```
CREATE TABLE t1_similarity
(id int, data array<float>)
DISTRIBUTED BY HASH(id);

INSERT INTO t1_similarity VALUES
(1, array<float>[0.1, 0.2, 0.3]),
(2, array<float>[0.2, 0.1, 0.3]),
(3, array<float>[0.3, 0.2, 0.1]);
```

2. Вычислите схожесть каждой строки в столбце `data` по сравнению с массивом `[0.1, 0.2, 0.3]` и выведите результат в порядке убывания.

```
SELECT id, data, cosine_similarity_norm([0.1, 0.2, 0.3], data) as dist
FROM t1_similarity
ORDER BY dist DESC;

+------+---------------+------------+
| id   | data          | dist       |
+------+---------------+------------+
|    1 | [0.1,0.2,0.3] | 0.14000002 |
|    2 | [0.2,0.1,0.3] | 0.13000001 |
|    3 | [0.3,0.2,0.1] | 0.10000001 |
+------+---------------+------------+
```

Cosh

Вычисляет гиперболический косинус аргумента.

Эта функция поддерживается начиная с версии v1.5.2.

Синтаксис

```
DOUBLE cosh(DOUBLE arg)
```

Параметры

`arg`: Вы можете указать только числовое значение. Эта функция преобразует числовое значение в значение DOUBLE перед вычислением гиперболического косинуса этого значения.

Возвращаемое значение

Возвращает значение типа данных DOUBLE.

Примечания по использованию

Если вы указываете нечисловое значение, эта функция возвращает `NULL`.

Примеры

```
mysql> select cosh(-1);
+--------------------+
| cosh(-1)           |
+--------------------+
| 1.5430806348152437 |
+--------------------+

mysql> select cosh(0);
+---------+
| cosh(0) |
+---------+
|       1 |
+---------+

mysql> select cosh(1);
+--------------------+
| cosh(1)            |
+--------------------+
| 1.5430806348152437 |
+--------------------+

mysql> select cosh("");
+----------+
| cosh('') |
+----------+
|     NULL |
+----------+
```

Keyword

COSH

Cot

Возвращает котангенс угла в радианах `x`. Если угол в радианах `x` равен `0` или `NULL`, возвращается `NULL`.

Синтаксис

```
COT(x);
```

Параметры

`x`: угол в радианах. Поддерживается тип DOUBLE.

Возвращаемое значение

Возвращает значение типа данных DOUBLE.

Примеры

```
mysql> select cot(3.1415926/2);
+------------------------+
| cot(3.1415926 / 2)     |
+------------------------+
| 2.6794896585028646e-08 |
+------------------------+
1 row in set (0.25 sec)

mysql> select cot(0);
+--------+
| cot(0) |
+--------+
|   NULL |
+--------+
1 row in set (0.03 sec)
```

Degrees

Преобразует угол `x` из радиан в градусы.

Синтаксис

```
DEGREES(x);
```

Параметры

`x`: угол в радианах. Поддерживается тип DOUBLE.

Возвращаемое значение

Возвращает значение типа DOUBLE.

Примеры

```
mysql> select degrees(3.1415926535898);
+---------------------------+
| degrees(3.1415926535898) |
+---------------------------+
|        180.0000000000004 |
+---------------------------+
1 row in set (0.07 sec)
```

Divide

Возвращает частное от деления x на y. Если y равен 0, возвращает null.

Синтаксис

```
divide(x, y)
```

Параметры

- `x`: поддерживаемые типы: DOUBLE, FLOAT, LARGEINT, BIGINT, INT, SMALLINT, TINYINT, DECIMALV2, DECIMAL32, DECIMAL64, DECIMAL128.
- `y`: поддерживаемые типы такие же, как и для `x`.

Возвращаемое значение

Возвращает значение типа DOUBLE.

Примечания

Если указано нечисловое значение, функция возвращает `NULL`.

Примеры

```
mysql> select divide(3, 2);
+--------------+
| divide(3, 2) |
+--------------+
|          1.5 |
+--------------+
1 row in set (0.00 sec)
```

E

Возвращает константу e (основание натурального логарифма).

Синтаксис

```
E();
```

Параметры

Эта функция не принимает параметров.

Возвращаемое значение

Возвращает константу e как значение типа DOUBLE.

Примеры

```
mysql> select e();
+-------------------+
| e()               |
+-------------------+
| 2.718281828459045 |
+-------------------+
1 row in set (0.01 sec)
```

Exp,dexp

Возвращает значение e в степени `x`. Эта функция называется экспонентой (функцией натурального логарифма).

Синтаксис

```
EXP(x);
```

Параметры

`x`: показатель степени. Поддерживается тип DOUBLE.

Возвращаемое значение

Возвращает значение типа DOUBLE.

Примеры

Возвращает e в степени 3.14:

```
mysql> select exp(3.14);
+--------------------+
| exp(3.14)          |
+--------------------+
| 23.103866858722185 |
+--------------------+
1 row in set (0.01 sec)
```

Floor, dfloor

Возвращает наибольшее целое число, не превышающее `x`.

Синтаксис

```
FLOOR(x);
```

Параметры

`x`: поддерживается тип DOUBLE.

Возвращаемое значение

Возвращает значение типа BIGINT.

Примеры

```
mysql> select floor(3.14);
+-------------+
| floor(3.14) |
+-------------+
|           3 |
+-------------+
1 row in set (0.01 sec)
```

Fmod

Возвращает остаток от деления с плавающей точкой (`dividend`/`divisor`). Это функция взятия остатка от деления (модуль).

Синтаксис

```
fmod(dividend,devisor);
```

Параметры

- `dividend`: поддерживается тип DOUBLE или FLOAT.
- `devisor`: поддерживается тип DOUBLE или FLOAT.

> **Примечание**
>
> Тип данных `devisor` должен совпадать с типом данных `dividend`. В противном случае Selena выполняет неявное преобразование типа.

Возвращаемое значение

Тип данных и знак результата совпадают с типом данных и знаком `dividend`. Если `divisor` равен `0`, возвращается `NULL`.

Примеры

```
mysql> select fmod(3.14,3.14);
+------------------+
| fmod(3.14, 3.14) |
+------------------+
|                0 |
+------------------+

mysql> select fmod(11.5,3);
+---------------+
| fmod(11.5, 3) |
+---------------+
|           2.5 |
+---------------+

mysql> select fmod(3,6);
+------------+
| fmod(3, 6) |
+------------+
|          3 |
+------------+

mysql> select fmod(3,0);
+------------+
| fmod(3, 0) |
+------------+
|       NULL |
+------------+
```

Greatest

Возвращает наибольшее значение из списка одного или нескольких параметров.

Как правило, возвращаемое значение имеет тот же тип данных, что и входные параметры.

Параметры сравниваются по следующим правилам:

- Если любой из входных параметров равен NULL, возвращается NULL.
- Если хотя бы один параметр имеет тип DOUBLE, все параметры сравниваются как значения DOUBLE. Это же правило применяется к типам данных DECIMAL и FLOAT.

- Если параметры являются смесью чисел и строк, но строки могут быть преобразованы в числа, параметры сравниваются как числа. Если строки не могут быть преобразованы в числа, параметры сравниваются как строки.
- Если параметры являются символами, они сравниваются по алфавитному порядку первой буквы.

Синтаксис

```
GREATEST(expr1,...);
```

Параметры

`expr1`: выражение для сравнения. Поддерживаются следующие типы данных:

- SMALLINT
- TINYINT
- INT
- BIGINT
- LARGEINT
- FLOAT
- DOUBLE
- DECIMALV2
- DECIMAL32
- DECIMAL64
- DECIMAL128
- DATETIME
- VARCHAR

Примеры

Пример 1: возврат наибольшего значения для одного входного параметра.

```
mysql> select greatest(3);
+-------------+
| greatest(3) |
+-------------+
|           3 |
+-------------+
1 row in set (0.01 sec)
```

Пример 2: возврат наибольшего значения из списка значений.

```
mysql> select greatest(3,4,5,5,6);
+-------------------------+
| greatest(3, 4, 5, 5, 6) |
+-------------------------+
|                       6 |
+-------------------------+
1 row in set (0.00 sec)
```

Пример 3: один параметр имеет тип DOUBLE, возвращается значение DOUBLE.

```
mysql> select greatest(7,4.5);
+------------------+
| greatest(7, 4.5) |
+------------------+
|              7.0 |
+------------------+
1 row in set (0.05 sec)
```

Пример 4: входные параметры являются смесью числа и строки, но строка может быть преобразована в число. Параметры сравниваются как числа.

```
mysql> select greatest(7,'9');
+------------------+
| greatest(7, '9') |
+------------------+
| 9                |
+------------------+
1 row in set (0.04 sec)
```

Пример 5: входные параметры являются смесью числа и строки, но строка не может быть преобразована в число. Параметры сравниваются как строки. Строка `'1'` меньше `'at'`.

```
select greatest(1,'at');
+-------------------+
| greatest(1, 'at') |
+-------------------+
| at                |
+-------------------+
```

Пример 6: входные параметры являются символами. Буква `Z` имеет наибольшее значение.

```
mysql> select greatest('A','B','Z');
+-------------------------+
| greatest('A', 'B', 'Z') |
+-------------------------+
| Z                       |
+-------------------------+
1 row in set (0.00 sec)
```

Ключевые слова

GREATEST, greatest

Least

Возвращает наименьшее значение из списка одного или нескольких параметров.

Как правило, возвращаемое значение имеет тот же тип данных, что и входные параметры.

Правила сравнения такие же, как и для функции greatest.

Синтаксис

```
LEAST(expr1,...);
```

Параметры

`expr1`: выражение для сравнения. Поддерживаются следующие типы данных:

- SMALLINT
- TINYINT
- INT
- BIGINT
- LARGEINT
- FLOAT
- DOUBLE
- DECIMALV2
- DECIMAL32
- DECIMAL64
- DECIMAL128
- DATETIME
- VARCHAR

Примеры

Пример 1: возврат наименьшего значения для одного входного параметра.

```
select least(3);
+----------+
| least(3) |
+----------+
|        3 |
+----------+
1 row in set (0.00 sec)
```

Пример 2: возврат наименьшего значения из списка значений.

```
select least(3,4,5,5,6);
+----------------------+
| least(3, 4, 5, 5, 6) |
+----------------------+
|                    3 |
+----------------------+
1 row in set (0.01 sec)
```

Пример 3: один параметр имеет тип DOUBLE, возвращается значение DOUBLE.

```
select least(4,4.5,5.5);
+--------------------+
| least(4, 4.5, 5.5) |
+--------------------+
|                4.0 |
+--------------------+
```

Пример 4: входные параметры являются смесью числа и строки, но строка может быть преобразована в число. Параметры сравниваются как числа.

```
select least(7,'5');
+---------------+
| least(7, '5') |
+---------------+
| 5             |
+---------------+
1 row in set (0.01 sec)
```

Пример 5: входные параметры являются смесью числа и строки, но строка не может быть преобразована в число. Параметры сравниваются как строки. Строка `'1'` меньше `'at'`.

```
select least(1,'at');
+----------------+
| least(1, 'at') |
+----------------+
| 1              |
+----------------+
```

Пример 6: входные параметры являются символами.

```
mysql> select least('A','B','Z');
+----------------------+
| least('A', 'B', 'Z') |
+----------------------+
| A                    |
+----------------------+
1 row in set (0.00 sec)
```

Ключевые слова

LEAST, least

Ln, dlog1, log

Вычисляет натуральный логарифм (по основанию `e`) числа.

Синтаксис

```
ln(arg)
log(arg)
dlog1(arg)
```

Параметры

- `arg`: значение, логарифм которого нужно вычислить. Поддерживается только тип данных DOUBLE.

> **ПРИМЕЧАНИЕ**

> Selena возвращает NULL, если `arg` является отрицательным числом или равен 0.

Возвращаемое значение

Возвращает значение типа DOUBLE.

Примеры

Пример 1: вычисление натурального логарифма числа 3.

```
mysql> select ln(3);
+--------------------+
| ln(3)              |
+--------------------+
| 1.0986122886681098 |
+--------------------+
1 row in set (0.00 sec)

mysql> select dlog1(3);
+--------------------+
| dlog1(3)              |
+--------------------+
| 1.0986122886681098 |
+--------------------+
1 row in set (0.00 sec)

mysql> select log(3);
+--------------------+
| log(3)              |
+--------------------+
| 1.0986122886681098 |
+--------------------+
1 row in set (0.00 sec)
```

Log

Вычисляет логарифм числа по указанному основанию. Если основание не указано, эта функция эквивалентна ln.

Синтаксис

```
log([base,] arg)
```

Параметры

- `base`: необязательный параметр. Основание логарифма. Поддерживается только тип данных DOUBLE. Если этот параметр не указан, функция эквивалентна ln.

> **ПРИМЕЧАНИЕ**
>
> Selena возвращает NULL, если `base` является отрицательным числом, равен 0 или 1.

- `arg`: значение, логарифм которого нужно вычислить. Поддерживается только тип данных DOUBLE.

> **ПРИМЕЧАНИЕ**
>
> Selena возвращает NULL, если `arg` является отрицательным числом или равен 0.

Возвращаемое значение

Возвращает значение типа DOUBLE.

Примеры

Пример 1: вычисление логарифма 8 по основанию 2.

```
mysql> select log(2,8);
+-----------+
| log(2, 8) |
+-----------+
|         3 |
+-----------+
1 row in set (0.01 sec)
```

Пример 2: вычисление логарифма 10 по основанию *e* (без указания основания).

```
mysql> select log(10);
+-------------------+
| log(10)           |
+-------------------+
| 2.302585092994046 |
+-------------------+
1 row in set (0.09 sec)
```

Log10, dlog10

Вычисляет десятичный логарифм числа (логарифм по основанию 10).

Синтаксис

```
log10(arg)
```

Параметры

- `arg`: значение, логарифм которого нужно вычислить. Поддерживается только тип данных DOUBLE.

> **ПРИМЕЧАНИЕ**
>
> Selena возвращает NULL, если `arg` является отрицательным числом или равен 0.

Возвращаемое значение

Возвращает значение типа DOUBLE.

Примеры

Пример 1: вычисление десятичного логарифма числа 100.

```
select log10(100);
+------------+
| log10(100) |
+------------+
|          2 |
+------------+
1 row in set (0.02 sec)
```

Log2

Вычисляет двоичный логарифм числа (логарифм по основанию 2).

Синтаксис

```
log2(arg)
```

Параметры

- `arg`: значение, логарифм которого нужно вычислить. Поддерживается только тип данных DOUBLE.

> **ПРИМЕЧАНИЕ**
>
> Selena возвращает NULL, если `arg` является отрицательным числом или равен 0.

Возвращаемое значение

Возвращает значение типа DOUBLE.

Примеры

Пример 1: вычисление двоичного логарифма числа 8.

```
mysql> select log2(8);
+---------+
| log2(8) |
+---------+
|       3 |
+---------+
1 row in set (0.00 sec)
```

Mod

Функция взятия остатка от деления, возвращает остаток от деления `dividend` на `divisor`.

Синтаксис

```
mod(dividend, divisor)
```

Параметры

- `dividend`: делимое.
- `divisor`: делитель.

Оба параметра `dividend` и `divisor` поддерживают следующие типы данных:

- TINYINT
- SMALLINT
- INT
- BIGINT
- LARGEINT
- FLOAT
- DOUBLE
- DECIMALV2
- DECIMAL32
- DECIMAL64
- DECIMAL128

> **ПРИМЕЧАНИЕ**
>
> Типы данных `dividend` и `divisor` должны совпадать. Если они не совпадают, Selena выполняет неявное преобразование типа.

Возвращаемое значение

Возвращает значение того же типа данных, что и `dividend`. Selena возвращает NULL, если `divisor` равен 0.

Примеры

```
mysql> select mod(3.14,3.14);
+-----------------+
| mod(3.14, 3.14) |
+-----------------+
|               0 |
+-----------------+

mysql> select mod(3.14, 3);
+--------------+
| mod(3.14, 3) |
+--------------+
|         0.14 |
+--------------+

select mod(11,-5);
+------------+
| mod(11, -5)|
+------------+
|          1 |
+------------+

select mod(-11,5);
+-------------+
| mod(-11, 5) |
+-------------+
|          -1 |
+-------------+
```

Multiply

Вычисляет произведение аргументов.

Синтаксис

```
multiply(arg1, arg2)
```

Параметры

`arg1`: числовой столбец-источник или литерал. `arg2`: числовой столбец-источник или литерал.

Возвращаемое значение

Возвращает произведение двух аргументов. Тип возвращаемого значения зависит от аргументов.

Примечания

Если указано нечисловое значение, функция завершится с ошибкой.

Примеры

```
MySQL [test]> select multiply(10,2);
+-----------------+
| multiply(10, 2) |
+-----------------+
|              20 |
+-----------------+
1 row in set (0.01 sec)

MySQL [test]> select multiply(1,2.1);
+------------------+
| multiply(1, 2.1) |
+------------------+
|              2.1 |
+------------------+
1 row in set (0.01 sec)

MySQL [test]> select * from t;
+------+------+------+------+
| id   | name | job1 | job2 |
+------+------+------+------+
|    2 |    2 |    2 |    2 |
+------+------+------+------+
1 row in set (0.08 sec)

MySQL [test]> select multiply(1.0,id) from t;
+-------------------+
| multiply(1.0, id) |
+-------------------+
|                 2 |
+-------------------+
1 row in set (0.01 sec)
```

Ключевые слова

multiply

Negative

Возвращает отрицательное значение входного аргумента `arg`.

Синтаксис

```
negative(arg)
```

Параметры

`arg` поддерживает следующие типы данных:

- BIGINT
- DOUBLE
- DECIMALV2
- DECIMAL32
- DECIMAL64
- DECIMAL128

Возвращаемое значение

Возвращает значение того же типа данных, что и входной аргумент.

© ООО «Платформа Данных Селена», 2026

Примеры

```
mysql> select negative(3);
+-------------+
| negative(3) |
+-------------+
|          -3 |
+-------------+
1 row in set (0.00 sec)

mysql> select negative(cast(3.14 as decimalv2));
+--------------------------------------+
| negative(CAST(3.14 AS DECIMAL(9,0))) |
+--------------------------------------+
|                                -3.14 |
+--------------------------------------+
1 row in set (0.01 sec)
```

Pi

Возвращает значение числа Пи.

Синтаксис

```
pi()
```

Параметры

Нет

Возвращаемое значение

Возвращает значение числа Пи как значение типа DOUBLE.

Примеры

```
mysql> select pi();
+-------------------+
| pi()              |
+-------------------+
| 3.141592653589793 |
+-------------------+
1 row in set (0.00 sec)
```

Pmod

Возвращает положительный остаток от деления `dividend` на `divisor`.

Синтаксис

```
pmod(dividend, divisor)
```

Параметры

- `dividend`: делимое число.
- `divisor`: делитель.

Оба параметра `arg1` и `arg2` поддерживают следующие типы данных:

- BIGINT
- DOUBLE

> **ПРИМЕЧАНИЕ**
>
> `dividend` и `divisor` должны иметь одинаковый тип данных. Selena выполняет неявное преобразование, если типы данных не совпадают.

Возвращаемое значение

Возвращает значение того же типа данных, что и `dividend`. Selena возвращает NULL, если `divisor` равен 0.

Примеры

```
mysql> select pmod(3.14,3.14);
+------------------+
| pmod(3.14, 3.14) |
+------------------+
|                0 |
+------------------+

mysql> select pmod(3,6);
+------------+
| pmod(3, 6) |
+------------+
|          3 |
+------------+

mysql> select pmod(11,5);
+-------------+
| pmod(11, 5) |
+-------------+
|           1 |
+-------------+

mysql> select pmod(-11,5);
+--------------+
| pmod(-11, 5) |
+--------------+
|            4 |
+--------------+

mysql> SELECT pmod(11,-5);
+--------------+
| pmod(11, -5) |
+--------------+
|           -4 |
+--------------+

mysql> SELECT pmod(-11,-5);
+---------------+
| pmod(-11, -5) |
+---------------+
|            -1 |
+---------------+
```

Positive

Возвращает `x` как значение.

Синтаксис

```
POSITIVE(x);
```

Параметры

`x`: поддерживает типы данных BIGINT, DOUBLE, DECIMALV2, DECIMAL32, DECIMAL64 и DECIMAL128.

Возвращаемое значение

Возвращает значение того же типа данных, что и значение `x`.

Примеры

```
mysql> select positive(3);
+-------------+
| positive(3) |
+-------------+
|           3 |
+-------------+
1 row in set (0.01 sec)

mysql> select positive(cast(3.14 as decimalv2));
+--------------------------------------+
| positive(CAST(3.14 AS DECIMAL(9,0))) |
+--------------------------------------+
|                                 3.14 |
+--------------------------------------+
1 row in set (0.01 sec)
```

Pow, power, dpow, fpow

Возвращает результат возведения `x` в степень `y`.

Синтаксис

```
POW(x,y);POWER(x,y);
```

Параметры

`x`: поддерживает тип данных DOUBLE.

`y`: поддерживает тип данных DOUBLE.

Возвращаемое значение

Возвращает значение типа DOUBLE.

Примеры

```
mysql> select pow(2,2);
+-----------+
| pow(2, 2) |
+-----------+
|         4 |
+-----------+
1 row in set (0.00 sec)

mysql> select power(4,3);
+-------------+
| power(4, 3) |
+-------------+
|          64 |
+-------------+
1 row in set (0.00 sec)
```

Radians

Преобразует `x` из градусов в радианы.

Синтаксис

```
REDIANS(x);
```

Параметры

`x`: поддерживает тип данных DOUBLE.

Возвращаемое значение

Возвращает значение типа DOUBLE.

Примеры

```
mysql> select radians(90);
+--------------------+
| radians(90)        |
+--------------------+
| 1.5707963267948966 |
+--------------------+
1 row in set (0.00 sec)
```

Rand, random

Возвращает случайное число с плавающей точкой от 0 (включительно) до 1 (исключительно).

Синтаксис

```
RAND(x);
```

Параметры

`x`: необязательный. Тип данных BIGINT. Независимо от того, указан `x` или нет, функция возвращает полностью случайное число.

Возвращаемое значение

Возвращает значение типа DOUBLE.

Примеры

```
select rand();
+--------------------+
| rand()             |
+--------------------+
| 0.9393535880089522 |
+--------------------+
1 row in set (0.01 sec)
select rand(3);
+--------------------+
| rand(3)            |
+--------------------+
| 0.6659865964511347 |
+--------------------+
1 row in set (0.00 sec)
```

Ключевые слова

RAND, RANDOM

Round, dround

Округляет число до указанного количества знаков.

- Если `n` не указан, `x` округляется до ближайшего целого числа.
- Если `n` указан, `x` округляется до `n` десятичных знаков. Если `n` отрицательное, `x` округляется слева от десятичной точки. При переполнении возвращается ошибка.

Синтаксис

```
ROUND(x [,n]);
```

Параметры

`x`: округляемое число. Поддерживает типы данных DOUBLE и DECIMAL128.

`n`: количество десятичных знаков для округления. Поддерживает тип данных INT. Этот параметр необязателен.

Возвращаемое значение

- Если указан только `x`, возвращаемое значение имеет следующий тип данных:

  ["DECIMAL128"] -> "DECIMAL128"

  ["DOUBLE"] -> "BIGINT"

- Если указаны оба параметра `x` и `n`, возвращаемое значение имеет следующий тип данных:

  ["DECIMAL128", "INT"] -> "DECIMAL128"

  ["DOUBLE", "INT"] -> "DOUBLE"

Примеры

```
mysql> select round(3.14);
+-------------+
| round(3.14) |
+-------------+
|           3 |
+-------------+

mysql> select round(3.14,1);
+----------------+
| round(3.14, 1) |
+----------------+
|            3.1 |
+----------------+

mysql> select round(13.14,-1);
+------------------+
| round(13.14, -1) |
+------------------+
|               10 |
+------------------+

mysql> select round(122.14,-1);
+--------------------+
| round(122.14, -1) |
+--------------------+
|               120 |
+--------------------+

mysql> select round(122.14,-2);
+--------------------+
| round(122.14, -2) |
+--------------------+
|               100 |
+--------------------+

mysql> select round(122.14,-3);
+--------------------+
| round(122.14, -3) |
+--------------------+
|                  0 |
+--------------------+
```

Sign

Возвращает знак числа `x`. Отрицательное число, 0 или положительное число на входе соответствует `-1`, `0` или `1` на выходе соответственно.

Синтаксис

```
SIGN(x);
```

Параметры

`x`: поддерживает тип данных DOUBLE.

Возвращаемое значение

Возвращает значение типа FLOAT.

Примеры

```
mysql> select sign(3.14159);
+---------------+
| sign(3.14159) |
+---------------+
|             1 |
+---------------+
1 row in set (0.02 sec)
```

Sin

Возвращает синус `x`, где `x` задан в радианах.

Синтаксис

```
SIN(x);
```

Параметры

`x`: поддерживает тип данных DOUBLE.

Возвращаемое значение

Возвращает значение типа DOUBLE.

Примеры

```
mysql> select sin(3.14);
+-----------------------+
| sin(3.14)             |
+-----------------------+
| 0.0015926529164868282 |
+-----------------------+
1 row in set (0.21 sec)
```

Sinh

Вычисляет гиперболический синус аргумента.

Эта функция поддерживается начиная с версии 3.0.

Синтаксис

```
DOUBLE sinh(DOUBLE arg)
```

Параметры

`arg`: можно указать только числовое значение. Функция преобразует числовое значение в тип DOUBLE перед вычислением гиперболического синуса.

Возвращаемое значение

Возвращает значение типа DOUBLE.

Примечания по использованию

Если указано нечисловое значение, функция возвращает `NULL`.

Примеры

```
mysql> select sinh(-1);
+---------------------+
| sinh(-1)            |
+---------------------+
| -1.1752011936438014 |
+---------------------+

mysql> select sinh(0);
+---------+
| sinh(0) |
+---------+
|       0 |
+---------+

mysql> select sinh(1);
+--------------------+
| sinh(1)            |
+--------------------+
| 1.1752011936438014 |
+--------------------+

mysql> select sinh("");
+----------+
| sinh('') |
+----------+
|     NULL |
+----------+
```

Ключевое слово

SINH

Sqrt, dsqrt

Вычисляет квадратный корень из значения. dsqrt аналогична sqrt.

Синтаксис

```
DOUBLE SQRT(DOUBLE x);
DOUBLE DSQRT(DOUBLE x);
```

Параметры

`x`: можно указать только числовое значение. Функция преобразует числовое значение в тип DOUBLE перед вычислением.

Возвращаемое значение

Возвращает значение типа DOUBLE.

Примечания по использованию

Если указано нечисловое значение, функция возвращает `NULL`.

Примеры

```
mysql> select sqrt(3.14);
+-------------------+
| sqrt(3.14)        |
+-------------------+
| 1.772004514666935 |
+-------------------+
1 row in set (0.01 sec)


mysql> select dsqrt(3.14);
+-------------------+
| dsqrt(3.14)       |
+-------------------+
| 1.772004514666935 |
+-------------------+
```

Square

Вычисляет квадрат значения.

Синтаксис

```
square(arg)
```

Параметры

`arg`: значение, квадрат которого нужно вычислить. Можно указать только числовое значение. Функция преобразует числовое значение в тип DOUBLE перед вычислением квадрата.

Возвращаемое значение

Возвращает значение типа DOUBLE.

Примечания по использованию

Если указано нечисловое значение, функция возвращает `NULL`.

Примеры

Пример 1: Вычисление квадрата числового значения.

```
mysql>  select square(11);
+------------+
| square(11) |
+------------+
|        121 |
+------------+
```

Пример 2: Вычисление квадрата нечислового значения. Возвращаемое значение - `NULL`.

```
mysql>  select square('2021-01-01');
+----------------------+
| square('2021-01-01') |
+----------------------+
|                 NULL |
+----------------------+
```

Tan

Возвращает тангенс `X`, где `X` задан в радианах.

Синтаксис

```
TAN(x);
```

Параметры

`x`: поддерживает тип данных DOUBLE.

Возвращаемое значение

Возвращает значение типа DOUBLE.

Примеры

```
mysql> select tan(3.14);
+-----------------------+
| tan(3.14)             |
+-----------------------+
| -0.001592654936407223 |
+-----------------------+
1 row in set (0.12 sec)
```

Tanh

Вычисляет гиперболический тангенс аргумента.

Эта функция поддерживается начиная с версии 3.0.

Синтаксис

```
DOUBLE tanh(DOUBLE arg)
```

Параметры

`arg`: можно указать только числовое значение. Функция преобразует числовое значение в тип DOUBLE перед вычислением гиперболического тангенса.

Возвращаемое значение

Возвращает значение типа DOUBLE.

Примечания по использованию

Если указано нечисловое значение, функция возвращает `NULL`.

Примеры

```
mysql> select tanh(-1);
+---------------------+
| tanh(-1)            |
+---------------------+
| -0.7615941559557649 |
+---------------------+

mysql> select tanh(0);
+---------+
| tanh(0) |
+---------+
|       0 |
+---------+

mysql> select tanh(1);
+--------------------+
| tanh(1)            |
+--------------------+
| 0.7615941559557649 |
+--------------------+

mysql> select tanh("");
+----------+
| tanh('') |
+----------+
|     NULL |
+----------+
```

Ключевое слово

TANH

Truncate

Округляет значение вниз до ближайшего равного или меньшего значения с указанным количеством знаков после десятичной точки.

Синтаксис

```
truncate(arg1,arg2);
```

Параметры

- `arg1`: входное значение для округления. Поддерживает следующие типы данных:
  - DOUBLE
  - DECIMAL128
- `arg2`: количество знаков после десятичной точки. Поддерживает тип данных INT.

Возвращаемое значение

Возвращает значение того же типа данных, что и `arg1`.

Примеры

```
mysql> select truncate(3.14,1);
+-------------------+
| truncate(3.14, 1) |
+-------------------+
|               3.1 |
+-------------------+
1 row in set (0.00 sec)
```

2.21 Percentile

Percentile_approx

Percentile_approx

Возвращает приближённое значение p-го процентиля, где значение p находится между 0 и 1.

Параметр сжатия (compression) является необязательным и имеет диапазон настройки [2048, 10000]. Чем больше значение, тем выше точность, больше потребление памяти и дольше время вычисления. Если параметр не указан или выходит за пределы диапазона [2048, 10000], функция выполняется с параметром сжатия по умолчанию 10000.

Эта функция использует фиксированный объём памяти, поэтому для столбцов с высокой кардинальностью можно использовать меньше памяти. Может использоваться для расчёта статистики, такой как tp99.

Синтаксис

```
DOUBLE PERCENTILE_APPROX(expr, DOUBLE p[, DOUBLE compression])
```

Примеры

```
MySQL > select `table`, percentile_approx(cost_time,0.99)
from log_statis
group by `table`;
+----------+-------------------------------------+
| table    | percentile_approx(`cost_time`, 0.99) |
+----------+-------------------------------------+
| test     |                                54.22 |
+----------+-------------------------------------+

MySQL > select `table`, percentile_approx(cost_time,0.99, 4096)
from log_statis
group by `table`;
+----------+-----------------------------------------------+
| table    | percentile_approx(`cost_time`, 0.99, 4096.0) |
+----------+-----------------------------------------------+
| test     |                                         54.21 |
+----------+-----------------------------------------------+
```

Ключевое слово

PERCENTILE_APPROX,PERCENTILE,APPROX

Percentile_approx_raw

Возвращает значение, соответствующее указанному процентилю из `x`.

Если `x` является столбцом, функция сначала сортирует значения в `x` по возрастанию и возвращает значение, соответствующее процентилю `y`.

Синтаксис

```
PERCENTILE_APPROX_RAW(x, y);
```

Параметры

- `x`: может быть столбцом или набором значений. Должен вычисляться в тип PERCENTILE.
- `y`: процентиль. Поддерживаемый тип данных — DOUBLE. Диапазон значений: [0.0,1.0].

Возвращаемое значение

Возвращает значение типа PERCENTILE.

Примеры

Создайте таблицу `aggregate_tbl`, где столбец `percent` является входным параметром для percentile_approx_raw().

```
CREATE TABLE `aggregate_tbl` (
  `site_id` largeint(40) NOT NULL COMMENT "id of site",
  `date` date NOT NULL COMMENT "time of event",
  `city_code` varchar(20) NULL COMMENT "city_code of user",
  `pv` bigint(20) SUM NULL DEFAULT "0" COMMENT "total page views",
  `percent` PERCENTILE PERCENTILE_UNION COMMENT "others"
) ENGINE=OLAP
AGGREGATE KEY(`site_id`, `date`, `city_code`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`site_id`)
PROPERTIES ("replication_num" = "1");
```

Вставьте данные в таблицу.

```
insert into aggregate_tbl values (5, '2020-02-23', 'city_code', 555,
percentile_hash(1));
insert into aggregate_tbl values (5, '2020-02-23', 'city_code', 555,
percentile_hash(2));
insert into aggregate_tbl values (5, '2020-02-23', 'city_code', 555,
percentile_hash(3));
insert into aggregate_tbl values (5, '2020-02-23', 'city_code', 555,
percentile_hash(4));
```

Вычислите значение, соответствующее процентилю 0.5.

```
mysql> select percentile_approx_raw(percent, 0.5) from aggregate_tbl;
+-------------------------------------+
| percentile_approx_raw(percent, 0.5) |
+-------------------------------------+
|                                 2.5 |
+-------------------------------------+
1 row in set (0.03 sec)
```

Percentile_cont

Percentile_cont

Вычисляет процентильное значение `expr` с линейной интерполяцией.

Синтаксис

```
PERCENTILE_CONT (expr, percentile)
```

Параметры

- `expr`: выражение, по которому упорядочиваются значения. Должно быть числовым типом данных, DATE или DATETIME. Например, если вы хотите найти медианный балл по физике, укажите столбец, содержащий баллы по физике.

- `percentile`: процентиль искомого значения. Это константа с плавающей точкой от 0 до 1. Например, если вы хотите найти медианное значение, установите этот параметр равным `0.5`.

Возвращаемое значение

Возвращает значение, которое находится на указанном процентиле. Если ни одно входное значение не находится точно на желаемом процентиле, результат вычисляется с использованием линейной интерполяции двух ближайших входных значений.

Тип данных совпадает с `expr`.

Примечания по использованию

Эта функция игнорирует значения NULL.

Примеры

Предположим, есть таблица с именем `exam` со следующими данными.

```
select * from exam order by Subject;
+-----------+-------+
| Subject   | Score |
+-----------+-------+
| chemistry |    80 |
| chemistry |   100 |
| chemistry |  NULL |
| math      |    60 |
| math      |    70 |
| math      |    85 |
| physics   |    75 |
| physics   |    80 |
| physics   |    85 |
| physics   |    99 |
+-----------+-------+
```

Рассчитайте медианный балл по каждому предмету, игнорируя значения NULL.

Запрос:

```
SELECT Subject, PERCENTILE_CONT (Score, 0.5)  FROM exam group by Subject;
```

Результат:

```
+-----------+----------------------------+
| Subject   | percentile_cont(Score, 0.5) |
+-----------+----------------------------+
| chemistry |                         90 |
| math      |                         70 |
| physics   |                       82.5 |
+-----------+----------------------------+
```

Percentile_disc

Percentile_disc

Возвращает процентильное значение на основе дискретного распределения входного столбца `expr`. Если точное процентильное значение не может быть найдено, эта функция возвращает большее значение из двух ближайших значений.

Эта функция поддерживается начиная с версии 2.5.

Синтаксис

```
PERCENTILE_DISC (expr, percentile)
```

Параметры

- `expr`: столбец, для которого вы хотите рассчитать процентильное значение. Столбец может быть любого типа данных, который поддерживает сортировку.
- `percentile`: процентиль искомого значения. Должно быть константой с плавающей точкой от 0 до 1. Например, если вы хотите найти медианное значение, установите этот параметр равным `0.5`. Если вы хотите найти значение на 70-м процентиле, укажите 0.7.

Возвращаемое значение

Тип данных возвращаемого значения совпадает с `expr`.

Примечания по использованию

Значения NULL игнорируются при расчёте.

Примеры

Создайте таблицу `exam` и вставьте данные в эту таблицу.

```
CREATE TABLE exam (
    subject STRING,
    score INT
)
DISTRIBUTED BY HASH(`subject`);

INSERT INTO exam VALUES
('chemistry',80),
('chemistry',100),
('chemistry',null),
('math',60),
('math',70),
('math',85),
('physics',75),
('physics',80),
('physics',85),
('physics',99);
```

```
select * from exam order by subject;
+-----------+-------+
| subject   | score |
+-----------+-------+
| chemistry |    80 |
| chemistry |   100 |
| chemistry |  NULL |
| math      |    60 |
| math      |    70 |
| math      |    85 |
| physics   |    75 |
| physics   |    80 |
| physics   |    85 |
| physics   |    99 |
+-----------+-------+
```

Рассчитайте медиану по каждому предмету.

```
select subject, percentile_disc(score, 0.5)
from exam group by subject;
```

Вывод

```
+-----------+----------------------------+
| subject   | percentile_disc(score, 0.5) |
+-----------+----------------------------+
| chemistry |                        100 |
| math      |                         70 |
| physics   |                         85 |
+-----------+----------------------------+
```

Percentile_empty

Создаёт значение типа PERCENTILE, которое используется для заполнения пустых значений при загрузке данных с помощью Stream Load или INSERT INTO.

Синтаксис

```
PERCENTILE_EMPTY();
```

Параметры

Нет

Возвращаемое значение

Возвращает значение типа PERCENTILE.

Примеры

Создайте таблицу. Столбец `percent` имеет тип PERCENTILE.

```
CREATE TABLE `aggregate_tbl` (
  `site_id` largeint(40) NOT NULL COMMENT "id of site",
  `date` date NOT NULL COMMENT "time of event",
  `city_code` varchar(20) NULL COMMENT "city_code of user",
  `pv` bigint(20) SUM NULL DEFAULT "0" COMMENT "total page views",
  `percent` PERCENTILE PERCENTILE_UNION COMMENT "others"
) ENGINE=OLAP
AGGREGATE KEY(`site_id`, `date`, `city_code`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`site_id`)
PROPERTIES ("replication_num" = "3");
```

Вставьте данные в таблицу.

```
INSERT INTO aggregate_tbl VALUES
(5, '2020-02-23', 'city_code', 555, percentile_empty());
```

Percentile_hash

Преобразует значения типа DOUBLE в значения типа PERCENTILE.

Синтаксис

```
PERCENTILE_HASH(x);
```

Параметры

`x`: поддерживаемый тип данных — DOUBLE.

Возвращаемое значение

Возвращает значение типа PERCENTILE.

Примеры

```
mysql> select percentile_approx_raw(percentile_hash(234.234), 0.99);
+-------------------------------------------------------+
| percentile_approx_raw(percentile_hash(234.234), 0.99) |
+-------------------------------------------------------+
|                                   234.23399353027344 |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

Percentile_union

Агрегирует данные типа PERCENTILE.

Синтаксис

```
percentile_union(expr);
```

Параметры

`expr`: поддерживаемый тип данных — PERCENTILE.

Возвращаемое значение

Возвращает значение типа PERCENTILE.

Примеры

Пример 1: использование данных percentile в материализованных представлениях.

Создайте таблицу.

```
CREATE TABLE sales_records(
    record_id int,
    seller_id int,
    store_id int,
    sale_date date,
    sale_amt bigint
) distributed BY hash(record_id)
PROPERTIES ("replication_num" = "3");
```

Создайте материализованное представление на основе столбца `sale_amt` таблицы.

```
create materialized view mv as
select store_id, percentile_union(percentile_hash(sale_amt))
from sales_records
group by store_id;
```

Пример 2: загрузка данных типа PERCENTILE.

Создайте таблицу Aggregate, содержащую столбец `sale_amt_per` типа PERCENTILE.

```
CREATE TABLE sales_records(
    record_id int,
    seller_id int,
    store_id int,
    sale_amt_per percentile percentile_union
) ENGINE=OLAP
AGGREGATE KEY(`record_id`, `seller_id`, `store_id`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`record_id`)
PROPERTIES (
    "replication_num" = "3",
    "storage_format" = "DEFAULT"
);
```

Запросите данные из `sale_amt_per`.

```
select percentile_approx_raw(percentile_union(sale_amt_per), 0.99) from
sales_records;
```

Загрузите данные, содержащие данные типа PERCENTILE, в таблицу `sales_records`.

```
curl --location-trusted -u root
    -H "columns: record_id, seller_id, store_id,tmp, sale_amt_per
=percentile_hash(tmp)"
    -H "column_separator:,"
    -T a http://<ip:port>/api/test/sales_records/_stream_load
```

2.22 Scalar

Hll_cardinality

Вычисляет кардинальность одного значения типа HLL.

Синтаксис

```
HLL_CARDINALITY(hll)
```

Примеры

```
MySQL > select HLL_CARDINALITY(uv_set) from test_uv;
+---------------------------+
| hll_cardinality(`uv_set`) |
+---------------------------+
|                         3 |
+---------------------------+
```

Ключевые слова

HLL,HLL_CARDINALITY

Hll_empty

Генерирует пустой столбец HLL для заполнения значений по умолчанию при вставке или загрузке данных.

Синтаксис

```
HLL_EMPTY()
```

Возвращаемое значение

Возвращает пустой HLL.

Примеры

Заполнение значений по умолчанию при вставке данных.

```
insert into hllDemo(k1,v1) values(10,hll_empty());
```

Заполнение значений по умолчанию при загрузке данных.

```
curl --location-trusted -u <username>:<password> \
    -H "columns: temp1, temp2, col1=hll_hash(temp1), col2=hll_empty()" \
    -T example7.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table7/_stream_load
```

Hll_hash

Преобразует значение в тип HLL. Обычно используется при импорте для преобразования значения из исходных данных в столбец типа HLL в таблице Selena.

Синтаксис

```
HLL_HASH(column_name)
```

Параметры

`column_name`: имя генерируемого столбца HLL.

Возвращаемое значение

Возвращает значение типа HLL.

Пример

```
mysql> select hll_cardinality(hll_hash("a"));
+--------------------------------+
| hll_cardinality(hll_hash('a')) |
+--------------------------------+
|                              1 |
+--------------------------------+
```

2.23 Spatial

ST_AsText,ST_AsWKT

Преобразует геометрическую фигуру в формат WKT (Well Known Text).

Синтаксис

```
VARCHAR ST_AsText(GEOMETRY geo)
```

Примеры

```
MySQL > SELECT ST_AsText(ST_Point(24.7, 56.7));
+----------------------------------+
| st_astext(st_point(24.7, 56.7)) |
+----------------------------------+
| POINT (24.7 56.7)                |
+----------------------------------+
```

Ключевые слова

ST_ASTEXT,ST_ASWKT,ST,ASTEXT,ASWKT

ST_Circle

Преобразует WKT (Well Known Text) в круг на сфере Земли.

Синтаксис

```
GEOMETRY ST_Circle(DOUBLE center_lng, DOUBLE center_lat, DOUBLE radius)
```

Параметры

`center_lng` указывает долготу центра круга.

`center_lat` указывает широту центра круга.

`radius` указывает радиус круга в метрах. Поддерживается максимальный радиус 9999999.

Примеры

```
MySQL > SELECT ST_AsText(ST_Circle(111, 64, 10000));
+---------------------------------------------+
| st_astext(st_circle(111.0, 64.0, 10000.0)) |
+---------------------------------------------+
| CIRCLE ((111 64), 10000)                    |
+---------------------------------------------+
```

Ключевые слова

ST_CIRCLE,ST,CIRCLE

ST_Contains

Проверяет, может ли геометрическая фигура shape1 полностью содержать shape2.

Синтаксис

```
BOOL ST_Contains(GEOMETRY shape1, GEOMETRY shape2)
```

Примеры

```
MySQL > SELECT ST_Contains(ST_Polygon("POLYGON ((0 0, 10 0, 10 10, 0 10, 0
0))"), ST_Point(5, 5));
+-----------------------------------------------------------------------------
--------------+
| st_contains(st_polygon('POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))'),
st_point(5.0, 5.0)) |
+-----------------------------------------------------------------------------
--------------+
|
1 |
+-----------------------------------------------------------------------------
--------------+

MySQL > SELECT ST_Contains(ST_Polygon("POLYGON ((0 0, 10 0, 10 10, 0 10, 0
0))"), ST_Point(50, 50));
+-----------------------------------------------------------------------------
----------------+
| st_contains(st_polygon('POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))'),
st_point(50.0, 50.0)) |
+-----------------------------------------------------------------------------
----------------+
|
0 |
+-----------------------------------------------------------------------------
----------------+
```

Ключевые слова

ST_CONTAINS,ST,CONTAINS

ST_Distance_Sphere

Вычисляет сферическое расстояние между двумя точками на Земле в метрах. Входными параметрами являются долгота X, широта X, долгота Y, широта Y.

Синтаксис

```
DOUBLE ST_Distance_Sphere(DOUBLE x_lng, DOUBLE x_lat, DOUBLE y_lng, DOUBLE
x_lat)
```

Примеры

```
MySQL > select st_distance_sphere(116.35620117, 39.939093, 116.4274406433,
39.9020987219);
+-----------------------------------------------------------------------------
--+
| st_distance_sphere(116.35620117, 39.939093, 116.4274406433,
39.9020987219) |
+-----------------------------------------------------------------------------
--+
|
7336.9135549995917 |
+-----------------------------------------------------------------------------
--+
```

Ключевые слова

ST_DISTANCE_SPHERE,ST,DISTANCE,SPHERE

ST_GeometryFromText,ST_GeomFromText

Преобразует WKT (Well Known Text) в соответствующую геометрию в памяти.

Синтаксис

```
GEOMETRY ST_GeometryFromText(VARCHAR wkt)
```

Примеры

```
MySQL > SELECT ST_AsText(ST_GeometryFromText("LINESTRING (1 1, 2 2)"));
+-------------------------------------------------------+
| st_astext(st_geometryfromtext('LINESTRING (1 1, 2 2)')) |
+-------------------------------------------------------+
| LINESTRING (1 1, 2 2)                                 |
+-------------------------------------------------------+
```

Ключевые слова

ST_GEOMETRYFROMTEXT,ST_GEOMFROMTEXT,ST,GEOMETRYFROMTEXT,GEOMFROMTEXT

ST_LineFromText,ST_LineStringFromText

Преобразует WKT (Well Known Text) в представление в памяти в форме линии.

Синтаксис

```
GEOMETRY ST_LineFromText(VARCHAR wkt)
```

Примеры

```
MySQL > SELECT ST_AsText(ST_LineFromText("LINESTRING (1 1, 2 2)"));
+-----------------------------------------------------------+
| st_astext(st_linefromtext('LINESTRING (1 1, 2 2)'))     |
+-----------------------------------------------------------+
| LINESTRING (1 1, 2 2)                                     |
+-----------------------------------------------------------+
```

Ключевые слова

ST_LINEFROMTEXT,ST_LINESTRINGFROMTEXT,ST,LINEFROMTEXT,LINESTRINGFROMTEXT

ST_Point

Возвращает соответствующую точку с заданными координатами X и Y. В настоящее время это значение имеет смысл только на сферическом множестве. X/Y соответствует долготе/широте.

> **Внимание**
>
> При прямом вызове ST_Point() запрос может зависнуть.

Синтаксис

```
POINT ST_Point(DOUBLE x, DOUBLE y)
```

Примеры

```
MySQL > SELECT ST_AsText(ST_Point(24.7, 56.7));
+---------------------------------+
| st_astext(st_point(24.7, 56.7)) |
+---------------------------------+
| POINT (24.7 56.7)               |
+---------------------------------+
```

Ключевые слова

ST_POINT,ST,POINT

ST_Polygon,ST_PolyFromText,ST_PolygonFromText

Преобразует WKT (Well Known Text) в соответствующую форму полигона в памяти.

Синтаксис

```
GEOMETRY ST_Polygon(VARCHAR wkt)
```

Примеры

```
MySQL > SELECT ST_AsText(ST_Polygon("POLYGON ((0 0, 10 0, 10 10, 0 10, 0
0))"));
+-------------------------------------------------------------------+
| st_astext(st_polygon('POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))')) |
+-------------------------------------------------------------------+
| POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))                           |
+-------------------------------------------------------------------+
```

Ключевые слова

ST_POLYGON,ST_POLYFROMTEXT,ST_POLYGONFROMTEXT,ST,POLYGON,POLYFROMTEXT,POLYGONFROMTEXT

ST_X

Если точка имеет допустимый тип Point, возвращает соответствующее значение координаты X.

Синтаксис

```
DOUBLE ST_X(POINT point)
```

Примеры

```
MySQL > SELECT ST_X(ST_Point(24.7, 56.7));
+----------------------------+
| st_x(st_point(24.7, 56.7)) |
+----------------------------+
|                       24.7 |
+----------------------------+
```

Ключевые слова

ST_X,ST,X

ST_Y

Если точка имеет допустимый тип Point, возвращает соответствующее значение координаты Y.

Синтаксис

```
DOUBLE ST_Y(POINT point)
```

Примеры

```
MySQL > SELECT ST_Y(ST_Point(24.7, 56.7));
+----------------------------+
| st_y(st_point(24.7, 56.7)) |
+----------------------------+
|                       56.7 |
+----------------------------+
```

Ключевые слова

ST_Y,ST,Y

2.24 String

Append_trailing_char_if_absent

Если строка str не пуста и не содержит символ trailing_char в конце, добавляет символ trailing_char в конец. trailing_char может содержать только один символ. Если он содержит несколько символов, эта функция вернёт NULL.

Синтаксис

```
VARCHAR append_trailing_char_if_absent(VARCHAR str, VARCHAR trailing_char)
```

Примеры

```
MySQL [test]> select append_trailing_char_if_absent('a','c');
+-----------------------------------------+
|append_trailing_char_if_absent('a', 'c')  |
+-----------------------------------------+
| ac                                      |
+-----------------------------------------+
1 row in set (0.02 sec)

MySQL [test]> select append_trailing_char_if_absent('ac','c');
+------------------------------------------+
|append_trailing_char_if_absent('ac', 'c')  |
+------------------------------------------+
| ac                                       |
+------------------------------------------+
1 row in set (0.00 sec)
```

Ключевые слова

APPEND_TRAILING_CHAR_IF_ABSENT

Ascii

Эта функция возвращает ASCII-код самого левого символа заданной строки.

Синтаксис

```
INT ascii(VARCHAR str)
```

Примеры

```
MySQL > select ascii('1');
+------------+
| ascii('1') |
+------------+
|         49 |
+------------+

MySQL > select ascii('234');
+--------------+
| ascii('234') |
+--------------+
|           50 |
+--------------+
```

Ключевые слова

ASCII

Char

CHAR() возвращает символьное значение заданного целочисленного значения согласно таблице ASCII.

Синтаксис

```
char(n)
```

Параметры

- `n`: целочисленное значение

Возвращаемое значение

Возвращает значение типа VARCHAR.

Примеры

```
> select char(77);
+----------+
| char(77) |
+----------+
| M        |
+----------+
```

Ключевые слова

CHAR

Char_length

Эта функция возвращает длину строки. Для многобайтовых символов возвращает количество символов. В настоящее время поддерживается только кодировка utf8. Примечание: эта функция также называется character_length.

Синтаксис

```
INT char_length(VARCHAR str)
```

Примеры

```
MySQL > select char_length("abc");
+--------------------+
| char_length('abc') |
+--------------------+
|                  3 |
+--------------------+
```

Ключевые слова

CHAR_LENGTH, CHARACTER_LENGTH

Character_length

Эта функция является псевдонимом `char_length`. Подробное описание см. в функции char_length.

Ключевые слова

CHAR_LENGTH,CHARACTER_LENGTH

Concat

Эта функция объединяет несколько строк. Если какое-либо из значений параметров равно NULL, возвращается NULL.

Синтаксис

```
VARCHAR concat(VARCHAR,...)
```

Примеры

```
MySQL > select concat("a", "b");
+------------------+
| concat('a', 'b') |
+------------------+
| ab               |
+------------------+

MySQL > select concat("a", "b", "c");
+-----------------------+
| concat('a', 'b', 'c') |
+-----------------------+
| abc                   |
+-----------------------+

MySQL > select concat("a", null, "c");
+------------------------+
| concat('a', NULL, 'c') |
+------------------------+
| NULL                   |
+------------------------+
```

Ключевые слова

CONCAT

Concat_ws

Эта функция использует первый аргумент sep в качестве разделителя, который объединяет второй и остальные аргументы в строку. Если разделитель равен NULL, результат будет NULL. concat_ws не пропускает пустые строки, но пропускает значения NULL.

Синтаксис

```
VARCHAR concat_ws(VARCHAR sep, VARCHAR str,...)
```

Примеры

```
MySQL > select concat_ws("Rock", "Star", "s");
+--------------------------------+
| concat_ws('Rock', 'Star', 's') |
+--------------------------------+
| Selena                         |
+--------------------------------+

MySQL > select concat_ws(NULL, "Star", "s");
+------------------------------+
| concat_ws(NULL, 'Star', 's') |
+------------------------------+
| NULL                         |
+------------------------------+
1 row in set (0.01 sec)

MySQL > Selena > select concat_ws("Rock", "Star", NULL, "s");
+---------------------------------------+
| concat_ws('Rock', 'Star', NULL, 's') |
+---------------------------------------+
| Selena                                |
+---------------------------------------+
1 row in set (0.04 sec)
```

Ключевые слова

CONCAT_WS,CONCAT,WS

Crc32

Возвращает 32-битное значение циклического избыточного кода (CRC) строки. Если входное значение равно NULL, возвращается NULL.

Эта функция используется для обнаружения ошибок. Она использует алгоритм CRC32 для обнаружения изменений между исходными и целевыми данными.

Эта функция поддерживается начиная с версии 3.3.

Синтаксис

```
BIGINT crc32(VARCHAR str)
```

Параметры

`str`: строка, для которой нужно вычислить значение CRC32.

Примеры

```
mysql > select crc32("selena");
+--------------------+
| crc32('selena') |
+--------------------+
|         2312449062 |
+--------------------+

mysql > select crc32(null);
+-------------+
| crc32(NULL) |
+-------------+
|        NULL |
+-------------+
1 row in set (0.18 sec)
```

Ключевые слова

CRC32

Ends_with

Возвращает `true`, если строка заканчивается указанным суффиксом. В противном случае возвращает `false`. Если аргумент равен NULL, результат будет NULL.

Синтаксис

```
BOOLEAN ENDS_WITH (VARCHAR str, VARCHAR suffix)
```

Примеры

```
MySQL > select ends_with("Hello selena", "selena");
+-----------------------------------+
| ends_with('Hello selena', 'selena') |
+-----------------------------------+
|                                 1 |
+-----------------------------------+

MySQL > select ends_with("Hello selena", "Hello");
+-----------------------------------+
| ends_with('Hello selena', 'Hello') |
+-----------------------------------+
|                                 0 |
+-----------------------------------+
```

Ключевые слова

ENDS_WITH

Field

Возвращает индекс (позицию) значения в списке значений.

Эта функция поддерживается начиная с версии 3.5.

Синтаксис

```
INT field(VARCHAR val, VARCHAR val1, val2, ...);
```

- `val`: значение для поиска в списке.

- `val1`, `val2`, ...: значения в списке.

Примечания по использованию

- Если указанное значение не найдено в списке значений, эта функция вернёт `0`. Если указанное значение равно NULL, эта функция вернёт `0`.
- Если все аргументы функции являются строками, все аргументы сравниваются как строки. Если все аргументы являются числами, они сравниваются как числа. В противном случае аргументы сравниваются как DOUBLE.

Примеры

```
MYSQL > select field('a', 'b', 'a', 'd');
+---------------------------+
| field('a', 'b', 'a', 'd') |
+---------------------------+
|                         2 |
+---------------------------+
```

Ключевые слова

FIELD

Find_in_set

Эта функция возвращает позицию первого вхождения str в strlist (отсчёт начинается с 1). Strlist — это строка, разделённая запятыми. Если str не найдена, возвращает 0. Если аргумент равен NULL, результат будет NULL.

Синтаксис

```
INT find_in_set(VARCHAR str, VARCHAR strlist)
```

Примеры

```
MySQL > select find_in_set("b", "a,b,c");
+---------------------------+
| find_in_set('b', 'a,b,c') |
+---------------------------+
|                         2 |
+---------------------------+
```

Ключевые слова

FIND_IN_SET,FIND,IN,SET

Format_bytes

Преобразует количество байтов в удобочитаемую строку с соответствующими единицами измерения (B, KB, MB, GB, TB, PB, EB).

Эта функция полезна для отображения размеров файлов, таблиц, использования памяти и других метрик, связанных с хранением данных, в удобном для пользователя формате. Она использует вычисления на основе 1024 (двоичные префиксы), но отображает единицы как KB, MB, GB для простоты и привычности.

Эта функция поддерживается начиная с версии 3.4.

Синтаксис

```
VARCHAR format_bytes(BIGINT bytes)
```

Параметры

- `bytes`: количество байтов для форматирования. Поддерживается тип BIGINT. Должно быть неотрицательным целым числом.

Возвращаемое значение

Возвращает значение типа VARCHAR, представляющее отформатированный размер в байтах с соответствующими единицами измерения.

- Для значений меньше 1024: возвращает точное число с "B" (байты)

- Для больших значений: возвращает отформатированную строку с 2 десятичными знаками и соответствующей единицей (KB, MB, GB, TB, PB, EB)
- Возвращает NULL, если входное значение отрицательное или NULL

Примечания по использованию

- Функция использует вычисления на основе 1024 (двоичные) внутри (1 KB = 1024 байта, 1 MB = 1024^2 байта и т.д.)
- Единицы отображаются как KB, MB, GB, TB, PB, EB для привычности пользователя, хотя они представляют двоичные префиксы (KiB, MiB, GiB и т.д.)
- Значения больше или равные 1 KB отображаются ровно с 2 десятичными знаками
- Значения в байтах (меньше 1024) отображаются как целые числа без десятичных знаков
- Отрицательные значения возвращают NULL
- Поддерживает значения до 8 EB (эксабайт), охватывая весь диапазон BIGINT

Примеры

Пример 1: Форматирование различных размеров в байтах.

```
SELECT format_bytes(0);
-- 0 B

SELECT format_bytes(123);
-- 123 B

SELECT format_bytes(1024);
-- 1.00 KB

SELECT format_bytes(4096);
-- 4.00 KB

SELECT format_bytes(123456789);
-- 117.74 MB

SELECT format_bytes(10737418240);
-- 10.00 GB
```

Пример 2: Обработка граничных случаев и значений null.

```
SELECT format_bytes(-1);
-- NULL

SELECT format_bytes(NULL);
-- NULL

SELECT format_bytes(0);
-- 0 B
```

Пример 3: Практическое использование для мониторинга размера таблиц.

```
-- Создание примера таблицы с информацией о размере
CREATE TABLE storage_info (
    table_name VARCHAR(64),
    size_bytes BIGINT
);

INSERT INTO storage_info VALUES
('user_profiles', 1073741824),
('transaction_logs', 5368709120),
('product_catalog', 524288000),
('analytics_data', 2199023255552);

-- Отображение размеров таблиц в удобочитаемом формате
SELECT
    table_name,
    size_bytes,
    format_bytes(size_bytes) as formatted_size
FROM storage_info
ORDER BY size_bytes DESC;

-- Ожидаемый результат:
-- +------------------+---------------+----------------+
-- | table_name       | size_bytes    | formatted_size |
-- +------------------+---------------+----------------+
-- | analytics_data   | 2199023255552 | 2.00 TB        |
-- | transaction_logs | 5368709120    | 5.00 GB        |
-- | user_profiles    | 1073741824    | 1.00 GB        |
-- | product_catalog  | 524288000     | 500.00 MB      |
-- +------------------+---------------+----------------+
```

Пример 4: Точность и поведение округления.

```
SELECT format_bytes(1536);       -- 1.50 KB (ровно 1.5 KB)
SELECT format_bytes(1025);       -- 1.00 KB (округлено до 2 знаков)
SELECT format_bytes(1048576 + 52429);  -- 1.05 MB (округлено до 2 знаков)
```

Пример 5: Полный диапазон поддерживаемых единиц.

```
SELECT format_bytes(512);                    -- 512 B
SELECT format_bytes(2048);                   -- 2.00 KB
SELECT format_bytes(1572864);                -- 1.50 MB
SELECT format_bytes(2147483648);             -- 2.00 GB
SELECT format_bytes(1099511627776);          -- 1.00 TB
SELECT format_bytes(1125899906842624);       -- 1.00 PB
SELECT format_bytes(1152921504606846976);    -- 1.00 EB
```

Ключевые слова

FORMAT_BYTES

Group_concat

Объединяет ненулевые значения из группы в одну строку с аргументом `sep`, который по умолчанию равен `,`, если не указан. Эта функция может использоваться для объединения значений из нескольких строк столбца в одну строку.

group_concat поддерживает DISTINCT и ORDER BY в версиях 3.0 старше 3.0.6 и версиях 3.1 старше 3.1.3.

Синтаксис

```
VARCHAR GROUP_CONCAT([DISTINCT] expr [,expr ...]
             [ORDER BY {unsigned_integer | col_name | expr}
                 [ASC | DESC] [,col_name ...]]
             [SEPARATOR sep])
```

Параметры

- `expr`: значения для объединения, нулевые значения игнорируются. Должен вычисляться в VARCHAR. Можно опционально указать `DISTINCT` для удаления

дублирующихся значений из результирующей строки. Если вы хотите объединить несколько `expr` напрямую, используйте concat или concat_ws для указания формата.

- Элементы в ORDER BY могут быть беззнаковыми целыми числами (начиная с 1), именами столбцов или обычными выражениями. По умолчанию результаты сортируются по возрастанию. Вы также можете явно указать ключевое слово ASC. Если вы хотите сортировать результаты по убыванию, добавьте ключевое слово DESC к имени столбца, по которому выполняется сортировка.
- `sep`: опциональный разделитель, используемый для объединения ненулевых значений из разных строк. Если не указан, по умолчанию используется `,` (запятая). Чтобы убрать разделитель, укажите пустую строку `''`.

> **ПРИМЕЧАНИЕ**
>
> Начиная с версий 3.0.6 и 3.1.3 изменилось поведение при указании разделителя. Вы должны использовать `SEPARATOR` для объявления разделителя, например, `select group_concat(name SEPARATOR '-') as res from ss;`.

Возвращаемое значение

Возвращает строковое значение для каждой группы и возвращает NULL, если нет ненулевых значений.

Вы можете ограничить длину строки, возвращаемой group_concat, установив переменную сессии `group_concat_max_len`, которая по умолчанию равна 1024. Минимальное значение: 4. Единица измерения: символы.

Пример:

```
SET [GLOBAL | SESSION] group_concat_max_len = <value>;
```

Примеры

1. Создайте таблицу `ss`, содержащую оценки по предметам.

```
CREATE TABLE `ss` (
  `id` int(11) NULL COMMENT "",
  `name` varchar(255) NULL COMMENT "",
  `subject` varchar(255) NULL COMMENT "",
  `score` int(11) NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 4
PROPERTIES (
"replication_num" = "1"
);

insert into ss values (1,"Tom","English",90);
insert into ss values (1,"Tom","Math",80);
insert into ss values (2,"Tom","English",NULL);
insert into ss values (2,"Tom",NULL,NULL);
insert into ss values (3,"May",NULL,NULL);
insert into ss values (3,"Ti","English",98);
insert into ss values (4,NULL,NULL,NULL);
insert into ss values (NULL,"Ti","Phy",98);

select * from ss order by id;
+------+------+---------+-------+
| id   | name | subject | score |
+------+------+---------+-------+
| NULL | Ti   | Phy     |    98 |
|    1 | Tom  | English |    90 |
|    1 | Tom  | Math    |    80 |
|    2 | Tom  | English |  NULL |
|    2 | Tom  | NULL    |  NULL |
|    3 | May  | NULL    |  NULL |
|    3 | Ti   | English |    98 |
|    4 | NULL | NULL    |  NULL |
+------+------+---------+-------+
```

2. Использование group_concat.

Пример 1: Объединение имён в строку с разделителем по умолчанию, нулевые значения игнорируются. Дублирующиеся имена сохраняются.

```
select group_concat(name) as res from ss;
+---------------------------+
| res                       |
+---------------------------+
| Tom,Tom,Ti,Tom,Tom,May,Ti |
+---------------------------+
```

Пример 2: Объединение имён в строку с разделителем `-`, нулевые значения игнорируются. Дублирующиеся имена сохраняются.

```
select group_concat(name SEPARATOR '-') as res from ss;
+---------------------------+
| res                       |
+---------------------------+
| Ti-May-Ti-Tom-Tom-Tom-Tom |
+---------------------------+
```

Пример 3: Объединение уникальных имён в строку с разделителем по умолчанию, нулевые значения игнорируются. Дублирующиеся имена удаляются.

```
select group_concat(distinct name) as res from ss;
+---------------------------+
| res                       |
+---------------------------+
| Ti,May,Tom                |
+---------------------------+
```

Пример 4: Объединение строк имя-предмет с одинаковым ID в порядке возрастания `score`. Например, `TomMath` и `TomEnglish` имеют ID 1 и объединяются через запятую в порядке возрастания `score`.

```
 select id, group_concat(distinct name,subject order by score) as res from
ss group by id order by id;
 +------+--------------------+
 | id   | res                |
 +------+--------------------+
 | NULL | TiPhy              |
 |    1 | TomMath,TomEnglish |
 |    2 | TomEnglish         |
 |    3 | TiEnglish          |
 |    4 | NULL               |
 +------+--------------------+
```

Пример 5: group_concat вложен с concat(), который используется для объединения `name`, `-` и `subject` в строку. Строки в одной строке сортируются в порядке возрастания `score`.

```
 select id, group_concat(distinct concat(name, '-',subject) order by score)
as res from ss group by id order by id;
 +------+----------------------+
 | id   | res                  |
 +------+----------------------+
 | NULL | Ti-Phy               |
 |    1 | Tom-Math,Tom-English |
 |    2 | Tom-English          |
 |    3 | Ti-English           |
 |    4 | NULL                 |
 +------+----------------------+
```

Пример 6: Совпадающий результат не найден, возвращается NULL.

```
select group_concat(distinct name) as res from ss where id < 0;
 +------+
 | res  |
 +------+
 | NULL |
 +------+
```

Пример 7: Ограничение длины возвращаемой строки до шести символов.

```
 set group_concat_max_len = 6;

 select id, group_concat(distinct name,subject order by score) as res from
ss group by id order by id;
 +------+--------+
 | id   | res    |
 +------+--------+
 | NULL | TiPhy  |
 |    1 | TomMat |
 |    2 | NULL   |
 |    3 | TiEngl |
 |    4 | NULL   |
 +------+--------+
```

Ключевые слова

GROUP_CONCAT,CONCAT,ARRAY_AGG

Hex

Если `x` является числовым значением, эта функция возвращает шестнадцатеричное строковое представление значения.

Если `x` является строкой, эта функция возвращает шестнадцатеричное строковое представление строки, где каждый символ в строке преобразуется в две шестнадцатеричные цифры.

Синтаксис

```
HEX(x);
```

Параметры

`x`: строка или число для преобразования. Поддерживаемые типы данных: BIGINT, VARCHAR и VARBINARY (версия 3.0 и выше).

Возвращаемое значение

Возвращает значение типа VARCHAR.

Примеры

```
mysql> select hex(3);
+--------+
| hex(3) |
+--------+
| 3      |
+--------+
1 row in set (0.00 sec)

mysql> select hex('3');
+----------+
| hex('3') |
+----------+
| 33       |
+----------+
1 row in set (0.00 sec)

mysql> select hex('apple');
+--------------+
| hex('apple') |
+--------------+
| 6170706C65   |
+--------------+

-- Входное значение является двоичным.

mysql> select hex(x'abab');
+-------------+
| hex('ABAB') |
+-------------+
| ABAB        |
+-------------+
1 row in set (0.01 sec)
```

Ключевые слова

HEX

Hex_decode_binary

Декодирует строку в шестнадцатеричной кодировке в двоичное значение.

Эта функция поддерживается начиная с версии 3.0.

Синтаксис

```
hex_decode_binary(str);
```

Параметры

`str`: строка для преобразования. Поддерживаемый тип данных — VARCHAR.

Пустое двоичное значение возвращается в следующих случаях:

- Длина строки равна 0 или количество символов в строке нечётное.
- Строка содержит символы, отличные от `[0-9]`, `[a-z]` и `[A-Z]`.

Возвращаемое значение

Возвращает значение типа VARBINARY.

Примеры

```
mysql> select hex(hex_decode_binary(hex("Hello Selena")));
+-------------------------------------------------+
| hex(hex_decode_binary(hex('Hello Selena'))) |
+-------------------------------------------------+
| 48656C6C6F2053746172526F636B73                  |
+-------------------------------------------------+

mysql> select hex_decode_binary(NULL);
+-----------------------------------------------------+
| hex_decode_binary(NULL)                             |
+-----------------------------------------------------+
| NULL                                                |
+-----------------------------------------------------+
```

Ключевые слова

HEX_DECODE_BINARY

Hex_decode_string

Эта функция выполняет операцию, обратную hex().

Она интерпретирует каждую пару шестнадцатеричных цифр во входной строке как число и преобразует его в байт, представленный этим числом. Возвращаемое значение — двоичная строка.

Эта функция поддерживается начиная с версии 3.0.

Синтаксис

```
hex_decode_string(str);
```

Параметры

`str`: строка для преобразования. Поддерживаемый тип данных — VARCHAR. Пустая строка возвращается в следующих случаях:

- Длина строки равна 0 или количество символов в строке нечётное.
- Строка содержит символы, отличные от `[0-9]`, `[a-z]` и `[A-Z]`.

Возвращаемое значение

Возвращает значение типа VARCHAR.

Примеры

```
mysql> select hex_decode_string(hex("Hello Selena"));
+-------------------------------------------+
| hex_decode_string(hex('Hello Selena')) |
+-------------------------------------------+
| Hello Selena                              |
+-------------------------------------------+
```

Ключевые слова

HEX_DECODE_STRING

Inet_aton

Принимает строку, содержащую IPv4-адрес в формате A.B.C.D. Возвращает число BIGINT, представляющее соответствующий IPv4-адрес в формате big endian.

Синтаксис

```
BIGINT inet_aton(VARCHAR ipv4)
```

Примеры

```
mysql> select inet_aton('192.168.1.1');
+--------------------------------------+
| inet_aton('192.168.1.1') |
+--------------------------------------+
| 3232235777                           |
+--------------------------------------+

mysql> select stringIp, inet_aton(stringIp) from ipv4;
+-----------------+----------------------------+
|stringIp         | inet_aton(stringIp)        |
+-----------------+----------------------------+
| 0.0.0.0         | 0                          |
| 255.255.255.255 | 4294967295                 |
| invalid         | NULL                       |
+-----------------+----------------------------+
```

Ключевые слова

INET_ATON

Instr

Эта функция возвращает позицию, где str впервые появляется в substr (отсчёт начинается с 1, измеряется в символах). Если str не найдена в substr, функция возвращает 0.

Синтаксис

```
INT instr(VARCHAR str, VARCHAR substr)
```

Примеры

```
MySQL > select instr("abc", "b");
+-------------------+
| instr('abc', 'b') |
+-------------------+
|                 2 |
+-------------------+

MySQL > select instr("abc", "d");
+-------------------+
| instr('abc', 'd') |
+-------------------+
|                 0 |
+-------------------+
```

Ключевые слова

INSTR

Lcase

Эта функция преобразует строку в нижний регистр. Аналогична функции lower.

Синтаксис

```
VARCHAR lcase(VARCHAR str)
```

Примеры

```
mysql> SELECT lcase("AbC123");
+-----------------+
|lcase('AbC123')  |
+-----------------+
|abc123           |
+-----------------+
```

Ключевые слова

LCASE

Left

Эта функция возвращает указанное количество символов с левой стороны заданной строки. Единица измерения длины: символ utf8. Примечание: эта функция также называется strleft.

Синтаксис

```
VARCHAR left(VARCHAR str,INT len)
```

Примеры

```
MySQL > select left("Hello selena",5);
+----------------------------+
| left('Hello selena', 5) |
+----------------------------+
| Hello                      |
+----------------------------+
```

Ключевые слова

LEFT

Length

Эта функция возвращает длину строки (в байтах).

Синтаксис

```
INT length(VARCHAR str)
```

Примеры

```
MySQL > select length("abc");
+---------------+
| length('abc') |
+---------------+
|             3 |
+---------------+
```

Ключевые слова

LENGTH

Locate

Эта функция используется для поиска местоположения подстроки в строке (начиная отсчет с 1 и измеряя в символах). Если указан третий аргумент pos, поиск позиций substr в строке начнется с позиции pos. Если str не найдена, будет возвращено 0.

Синтаксис

```
INT locate(VARCHAR substr, VARCHAR str[, INT pos])
```

Примеры

```
MySQL > SELECT LOCATE('bar', 'foobarbar');
+----------------------------+
| locate('bar', 'foobarbar') |
+----------------------------+
|                          4 |
+----------------------------+

MySQL > SELECT LOCATE('xbar', 'foobar');
+--------------------------+
| locate('xbar', 'foobar') |
+--------------------------+
|                        0 |
+--------------------------+

MySQL > SELECT LOCATE('bar', 'foobarbar', 5);
+------------------------------+
| locate('bar', 'foobarbar', 5) |
+------------------------------+
|                            7 |
+------------------------------+
```

Ключевое слово

LOCATE

Lower

Преобразует все строки в аргументе в нижний регистр.

Синтаксис

```
INT lower(VARCHAR str)
```

Примеры

```
MySQL > SELECT lower("AbC123");
+-----------------+
| lower('AbC123') |
+-----------------+
| abc123          |
+-----------------+
```

Ключевое слово

LOWER

Lpad

Эта функция возвращает строку длиной `len` (отсчет начинается с первого символа) в `str`. Если `len` больше длины `str`, возвращаемое значение удлиняется до `len` символов путем добавления символов pad перед `str`. Если `str` длиннее `len`, возвращаемое значение укорачивается до `len` символов. `len` означает длину в символах, а не в байтах.

Синтаксис

```
VARCHAR lpad(VARCHAR str, INT len[, VARCHAR pad])
```

Параметры

`str`: обязательный, строка для дополнения, которая должна иметь тип VARCHAR.

`len`: обязательный, длина возвращаемого значения, означает длину в символах, а не в байтах, должен иметь тип INT.

`pad`: опциональный, символы для добавления перед str, должны иметь тип VARCHAR. Если этот параметр не указан, по умолчанию добавляются пробелы.

Возвращаемое значение

Возвращает значение типа VARCHAR.

Примеры

```
MySQL > SELECT lpad("hi", 5, "xy");
+---------------------+
| lpad('hi', 5, 'xy') |
+---------------------+
| xyxhi               |
+---------------------+

MySQL > SELECT lpad("hi", 1, "xy");
+---------------------+
| lpad('hi', 1, 'xy') |
+---------------------+
| h                   |
+---------------------+

MySQL > SELECT lpad("hi", 5);
+---------------------+
| lpad('hi', 5, ' ')  |
+---------------------+
|    hi               |
+---------------------+
```

Ключевое слово

LPAD

Ltrim

Удаляет ведущие пробелы или указанные символы с начала (слева) аргумента `str`. Удаление указанных символов поддерживается начиная с Selena 2.5.0.

Синтаксис

```
VARCHAR ltrim(VARCHAR str[, VARCHAR characters])
```

Параметры

`str`: обязательный, строка для обрезки, которая должна иметь тип VARCHAR.

`characters`: опциональный, символы для удаления, должны иметь тип VARCHAR. Если этот параметр не указан, по умолчанию из строки удаляются пробелы. Если этот параметр установлен в пустую строку, возвращается ошибка.

Возвращаемое значение

Возвращает значение типа VARCHAR.

Примеры

Пример 1: Удаление пробелов с начала строки.

```
MySQL > SELECT ltrim('   ab d');
+------------------+
| ltrim('   ab d') |
+------------------+
| ab d             |
+------------------+
```

Пример 2: Удаление указанных символов с начала строки.

```
MySQL > SELECT ltrim("xxabcdxx", "x");
+------------------------+
| ltrim('xxabcdxx', 'x') |
+------------------------+
| abcdxx                 |
+------------------------+
```

Ссылки

- trim
- rtrim

Ключевое слово

LTRIM

Money_format

Эта функция возвращает строку, отформатированную как денежная строка. Целая часть разделяется запятой каждые три цифры, а десятичная часть сохраняется с двумя цифрами.

Синтаксис

```
VARCHAR money_format(Number)
```

Примеры

```
MySQL > select money_format(17014116);
+------------------------+
| money_format(17014116) |
+------------------------+
| 17,014,116.00          |
+------------------------+

MySQL > select money_format(1123.456);
+------------------------+
| money_format(1123.456) |
+------------------------+
| 1,123.46               |
+------------------------+

MySQL > select money_format(1123.4);
+----------------------+
| money_format(1123.4) |
+----------------------+
| 1,123.40             |
+----------------------+
```

Ключевое слово

MONEY_FORMAT,MONEY,FORMAT

Ngram_search

Вычисляет ngram-сходство двух строк.

К СВЕДЕНИЮ

- В настоящее время кодировка символов поддерживает только ASCII и не поддерживает UTF-8.
- Функция `ngram_search` чувствительна к регистру. Другая функция `ngram_search_case_insensitive` не чувствительна к регистру. В остальном эти две функции идентичны.

Синтаксис

```
DOUBLE ngram_search(VARCHAR haystack, VARCHAR needle, INT gram_num)
```

Параметры

- `haystack`: обязательный, первая строка для сравнения. Должна быть значением VARCHAR. Может быть именем столбца или константным значением. Если `haystack` является именем столбца, и для этого столбца в таблице создан N-Gram bloom filter индекс, индекс может ускорить вычисление функции `ngram_search`.

- `needle`: обязательный, вторая строка для сравнения. Должна быть значением VARCHAR. Может быть только константным значением.

  ПОДСКАЗКА

  - Длина значения `needle` не может превышать 2^15. Иначе будет выдана ошибка.
  - Если длина значения `haystack` больше 2^15, функция вернет 0.
  - Если длина значения `haystack` или `needle` меньше `gram_num`, функция вернет 0.

- `gram_num`: обязательный, используется для указания количества грамм. Рекомендуемое значение `4`.

Возвращаемое значение

Возвращает значение, описывающее степень сходства двух строк. Диапазон возвращаемого значения от 0 до 1. Чем больше это значение, тем более похожи строки.

Примеры

```
-- Значения haystack и needle являются константами.
mysql> select ngram_search('English', 'England',4);
+----------------------------------------+
| ngram_search('English', 'England', 4) |
+----------------------------------------+
|                                   0.25 |
+----------------------------------------+

-- Значение haystack является именем столбца, а значение needle -
константой.
mysql> select rowkey,ngram_search(rowkey,"31dc496b-760d-6f1a-4521-
050073a70000",4) as string_similarity from string_table order by
string_similarity desc limit 5;
+--------------------------------------+--------------------+
| rowkey                               | string_similarity |
+--------------------------------------+--------------------+
| 31dc496b-760d-6f1a-4521-050073a70000 |                  1 |
| 31dc496b-760d-6f1a-4521-050073a40000 |          0.8787879 |
| 31dc496b-760d-6f1a-4521-05007fa70000 |          0.8787879 |
| 31dc496b-760d-6f1a-4521-050073a30000 |          0.8787879 |
| 31dc496b-760d-6f1a-4521-0500c3a70000 |          0.8787879 |
+--------------------------------------+--------------------+
```

Null_or_empty

Эта функция возвращает true, когда строка пуста или NULL. В противном случае возвращает false.

Синтаксис

```
BOOLEAN NULL_OR_EMPTY (VARCHAR str)
BOOLEAN NULL_OR_EMPTY (any_array)
```

Примеры

```
MySQL > select null_or_empty(null);
+---------------------+
| null_or_empty(NULL) |
+---------------------+
|                   1 |
+---------------------+

MySQL > select null_or_empty("");
+--------------------+
| null_or_empty('') |
+--------------------+
|                  1 |
+--------------------+

MySQL > select null_or_empty("a");
+---------------------+
| null_or_empty('a') |
+---------------------+
|                   0 |
+---------------------+

MySQL > select null_or_empty([]);
+--------------------+
| null_or_empty('') |
+--------------------+
|                  1 |
+--------------------+

MySQL > select null_or_empty([1]);
+---------------------+
| null_or_empty('a') |
+---------------------+
|                   0 |
+---------------------+
```

Keyword

NULL_OR_EMPTY

Parse_url

Парсит URL и извлекает компонент из этого URL.

Синтаксис

```
parse_url(expr1,expr2);
```

Параметры

`expr1`: URL. Поддерживаемый тип данных VARCHAR.

`expr2`: компонент, который нужно извлечь из URL. Поддерживаемый тип данных VARCHAR. Допустимые значения:

- PROTOCOL
- HOST
- PATH
- REF
- AUTHORITY
- FILE
- USERINFO
- QUERY. Параметры в QUERY не могут быть возвращены. Если вы хотите вернуть конкретные параметры, используйте parse_url с trim для реализации этого. Подробности см. в разделе Примеры.

`expr2` **чувствителен к регистру**.

Возвращаемое значение

Возвращает значение типа VARCHAR. Если URL недействителен, возвращается ошибка. Если запрашиваемая информация не найдена, возвращается NULL.

Примеры

```
select parse_url('http://facebook.com/path/p1.php?query=1', 'HOST');
+-----------------------------------------------------------------+
| parse_url('http://facebook.com/path/p1.php?query=1', 'HOST') |
+-----------------------------------------------------------------+
| facebook.com                                                    |
+-----------------------------------------------------------------+

select parse_url('http://facebook.com/path/p1.php?query=1', 'AUTHORITY');
+-----------------------------------------------------------------------+
| parse_url('http://facebook.com/path/p1.php?query=1', 'AUTHORITY') |
+-----------------------------------------------------------------------+
| facebook.com                                                          |
+-----------------------------------------------------------------------+

select parse_url('http://facebook.com/path/p1.php?query=1', 'QUERY');
+-------------------------------------------------------------------+
| parse_url('http://facebook.com/path/p1.php?query=1', 'QUERY') |
+-------------------------------------------------------------------+
| query=1                                                           |
+-------------------------------------------------------------------+

select trim(parse_url('http://facebook.com/path/p1.php?query=1',
'QUERY'),'query=');
+-------------------------------------------------------------------------------
-----+
| trim(parse_url('http://facebook.com/path/p1.php?query=1', 'QUERY'),
'query=') |
+-------------------------------------------------------------------------------
-----+
| 1
|
+-------------------------------------------------------------------------------
-----+
```

Regexp_split

Разделяет строку `str` по регулярному выражению `pattern`, возвращает максимум `max_split` элементов типа `ARRAY<VARCHAR>`.

Синтаксис

```
regexp_split(str, pattern[, max_split])
```

Параметры

`str`: обязательный, строка для разделения, которая должна быть значением типа `VARCHAR`.

`pattern`: обязательный, шаблон регулярного выражения для разделения, который должен быть значением типа `VARCHAR`.

`max_split`: необязательный, максимальное количество элементов в возвращаемом значении, должен быть значением типа `INT`.

Возвращаемое значение

Возвращает значение типа `ARRAY<VARCHAR>`.

Примеры

```
mysql> select regexp_split('Selena', '');
+---------------------------------------+
| regexp_split('Selena', '')            |
+---------------------------------------+
| ["S","t","a","r","R","o","c","k","s"] |
+---------------------------------------+

mysql> select regexp_split('Selena', '[SR]');
+-----------------------------------+
| regexp_split('Selena', '[SR]') |
+-----------------------------------+
| ["","tar","ocks"]                 |
+-----------------------------------+

mysql> select regexp_split('Selena', '[SR]', 1);
+---------------------------------------+
| regexp_split('Selena', '[SR]', 1) |
+---------------------------------------+
| ["Selena"]                        |
+---------------------------------------+

mysql> select regexp_split('Selena', '[SR]', 2);
+---------------------------------------+
| regexp_split('Selena', '[SR]', 1) |
+---------------------------------------+
| ["","tarRocks"]                       |
+---------------------------------------+
```

Repeat

Эта функция повторяет `str` количество раз в соответствии с `count`. Когда `count` меньше 1, возвращает пустую строку. Когда `str` или `count` равен NULL, возвращает NULL.

Синтаксис

```
VARCHAR repeat(VARCHAR str, INT count)
```

Примеры

```
MySQL > SELECT repeat("a", 3);
+----------------+
| repeat('a', 3) |
+----------------+
| aaa            |
+----------------+

MySQL > SELECT repeat("a", -1);
+-----------------+
| repeat('a', -1) |
+-----------------+
|                 |
+-----------------+
```

Keyword

REPEAT,

Replace

Заменяет все вхождения символов в строке на другую строку. Эта функция выполняет поиск с учетом регистра при поиске `pattern`.

Эта функция поддерживается начиная с версии v1.5.2.

Примечание: До версии 3.0 эта функция была реализована как regexp_replace.

Syntax

```
VARCHAR replace(VARCHAR str, VARCHAR pattern, VARCHAR repl)
```

Parameters

- `str`: исходная строка.
- `pattern`: символы для замены. Обратите внимание, что это не регулярное выражение.
- `repl`: строка, используемая для замены символов в `pattern`.

Return value

Возвращает строку с замененными указанными символами.

Если любой аргумент равен NULL, результат равен NULL.

Если совпадающие символы не найдены, возвращается исходная строка.

Examples

```
-- Replace '.' in 'a.b.c' with '+'.

MySQL > SELECT replace('a.b.c', '.', '+');
+----------------------------+
| replace('a.b.c', '.', '+') |
+----------------------------+
| a+b+c                      |
+----------------------------+

-- No matching characters are found and the original string is returned.

MySQL > SELECT replace('a b c', '', '*');
+----------------------------+
| replace('a b c', '', '*') |
+----------------------------+
| a b c                      |
+----------------------------+

-- Replace 'like' with an empty string.

MySQL > SELECT replace('We like Selena', 'like', '');
+-------------------------------------------+
| replace('We like Selena', 'like', '') |
+-------------------------------------------+
| We  Selena                              |
+-------------------------------------------+

-- No matching characters are found and the original string is returned.

MySQL > SELECT replace('He is awesome', 'handsome', 'smart');
+-----------------------------------------------+
| replace('He is awesome', 'handsome', 'smart') |
+-----------------------------------------------+
| He is awesome                                 |
+-----------------------------------------------+
```

Keywords

REPLACE, replace

Reverse

Переворачивает строку или массив. Возвращает строку или массив с символами в строке или элементами массива в обратном порядке.

Syntax

```
reverse(param)
```

Parameters

`param`: строка или массив для переворачивания. Может быть типа VARCHAR, CHAR или ARRAY.

В настоящее время эта функция поддерживает только одномерные массивы, и элементы массива не могут быть типа DECIMAL. Эта функция поддерживает следующие типы элементов массива: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, VARCHAR, DECIMALV2, DATETIME, DATE и JSON. **JSON поддерживается начиная с версии 2.5.**

Return value

Тип возвращаемого значения совпадает с `param`.

Examples

Пример 1: Перевернуть строку.

```
MySQL > SELECT REVERSE('hello');
+------------------+
| REVERSE('hello') |
+------------------+
| olleh            |
+------------------+
1 row in set (0.00 sec)
```

Пример 2: Перевернуть массив.

```
MYSQL> SELECT REVERSE([4,1,5,8]);
+--------------------+
| REVERSE([4,1,5,8]) |
+--------------------+
| [8,5,1,4]          |
+--------------------+
```

Right

Эта функция возвращает указанное количество символов с правой стороны заданной строки. Единица длины: символ utf8. Примечание: Эта функция также называется strright.

Syntax

```
VARCHAR right(VARCHAR str,INT len)
```

Examples

```
MySQL > select right("Hello selena",9);
+-----------------------------+
| right('Hello selena', 9) |
+-----------------------------+
| selena                   |
+-----------------------------+
```

Keyword

RIGHT

Rpad

Эта функция возвращает строки длиной `len` (начиная отсчет с первого символа) в `str`. Если `len` длиннее, чем `str`, возвращаемое значение удлиняется до `len` символов путем добавления символов заполнения после `str`. Если `str` длиннее, чем `len`, возвращаемое значение сокращается до `len` символов. `len` означает длину символов, а не байтов.

Syntax

```
VARCHAR rpad(VARCHAR str, INT len[, VARCHAR pad])
```

Parameters

`str`: обязательный, строка для заполнения, которая должна иметь значение VARCHAR.

`len`: обязательный, длина возвращаемого значения, это означает длину символов, а не байтов, которая должна иметь значение INT.

`pad`: необязательный, символы, добавляемые после str, которые должны быть значением VARCHAR. Если этот параметр не указан, по умолчанию добавляются пробелы.

Return value

Возвращает значение VARCHAR.

Examples

```
MySQL > SELECT rpad("hi", 5, "xy");
+---------------------+
| rpad('hi', 5, 'xy') |
+---------------------+
| hixyx               |
+---------------------+

MySQL > SELECT rpad("hi", 1, "xy");
+---------------------+
| rpad('hi', 1, 'xy') |
+---------------------+
| h                   |
+---------------------+

MySQL > SELECT rpad("hi", 5);
+---------------------+
| rpad('hi', 5, ' ')  |
+---------------------+
| hi                  |
+---------------------+
```

Keyword

RPAD

Rtrim

Удаляет конечные пробелы или указанные символы с конца (справа) аргумента `str`. Удаление указанных символов поддерживается начиная с Selena 2.5.0.

Syntax

```
VARCHAR rtrim(VARCHAR str[, VARCHAR characters]);
```

Parameters

`str`: обязательный, строка для обрезки, которая должна иметь значение VARCHAR.

`characters`: необязательный, символы для удаления, которые должны быть значением VARCHAR. Если этот параметр не указан, по умолчанию из строки удаляются пробелы. Если этот параметр установлен на пустую строку, возвращается ошибка.

Return value

Возвращает значение VARCHAR.

Examples

Пример 1: Удалить три конечных пробела из строки.

```
mysql> SELECT rtrim('   ab d   ');
+---------------------+
| rtrim('   ab d   ') |
+---------------------+
|    ab d             |
+---------------------+
1 row in set (0.00 sec)
```

Пример 2: Удалить указанные символы с конца строки.

```
MySQL > SELECT rtrim("xxabcdxx", "x");
+------------------------+
| rtrim('xxabcdxx', 'x') |
+------------------------+
| xxabcd                 |
+------------------------+
```

References

- trim
- ltrim

Space

Возвращает строку из указанного количества пробелов.

Syntax

```
space(x);
```

Parameters

`x`: количество пробелов для возврата. Поддерживаемый тип данных — INT.

Return value

Возвращает значение типа VARCHAR.

Examples

```
mysql> select space(6);
+----------+
| space(6) |
+----------+
|          |
+----------+
1 row in set (0.00 sec)
```

Keywords

SPACE

Split

Эта функция разделяет заданную строку по разделителям и возвращает разделённые части в виде ARRAY.

Синтаксис

```
ARRAY<VARCHAR> split(VARCHAR content, VARCHAR delimiter)
```

Примеры

```
mysql> select split("a,b,c",",");
+---------------------+
| split('a,b,c', ',') |
+---------------------+
| ["a","b","c"]       |
+---------------------+

mysql> select split("a,b,c",",b,");
+------------------------+
| split('a,b,c', ',b,') |
+------------------------+
| ["a","c"]              |
+------------------------+

mysql> select split("abc","");
+------------------+
| split('abc', '') |
+------------------+
| ["a","b","c"]    |
+------------------+
```

Keyword

SPLIT

Split_part

Эта функция разделяет заданную строку по разделителям и возвращает запрошенную часть. (отсчёт начинается с начала)

Синтаксис

```
VARCHAR split_part(VARCHAR content, VARCHAR delimiter, INT field)
```

Параметры

`content`: Строка для разделения. Тип данных: VARCHAR.

`delimiter`: Разделитель, используемый для разделения строки. Тип данных: VARCHAR.

`field`: Позиция возвращаемой части. Положительные значения отсчитываются с начала, отрицательные — с конца. Тип данных: INT.

Возвращаемое значение

Возвращает указанную часть разделённой строки. Тип данных: VARCHAR.

© ООО «Платформа Данных Селена», 2026

Примеры

```
MySQL > select split_part("hello world", " ", 1);
+-----------------------------------+
|split_part('hello world', ' ', 1) |
+-----------------------------------+
| hello                             |
+-----------------------------------+

MySQL > select split_part("hello world", " ", 2);
+------------------------------------+
| split_part('hello world', ' ', 2) |
+------------------------------------+
| world                              |
+------------------------------------+

MySQL > select split_part("hello world", " ", -1);
+-----------------------------------+
|split_part('hello world', ' ', -1) |
+-----------------------------------+
| world                             |
+-----------------------------------+

MySQL > select split_part("hello world", " ", -2);
+------------------------------------+
| split_part('hello world', ' ', -2) |
+------------------------------------+
| hello                              |
+------------------------------------+

MySQL > select split_part("hello world", "|", 1);
+-----------------------------------+
| split_part('hello world', '|', 1) |
+-----------------------------------+
| hello world                       |
+-----------------------------------+

MySQL > select split_part("hello world", "|", -1);
+------------------------------------+
| split_part('hello world', '|', -1) |
+------------------------------------+
| hello world                        |
+------------------------------------+

MySQL > select split_part("hello world", "|", 2);
```

```
+----------------------------------+
| split_part('hello world', '|', 2) |
+----------------------------------+
|                                  |
+----------------------------------+

MySQL > select split_part("abca", "a", 1);
+----------------------------+
| split_part('abca', 'a', 1) |
+----------------------------+
|                            |
+----------------------------+

MySQL > select split_part("abca", "a", -1);
+-----------------------------+
| split_part('abca', 'a', -1) |
+-----------------------------+
|                             |
+-----------------------------+

MySQL > select split_part("abca", "a", -2);
+-----------------------------+
| split_part('abca', 'a', -2) |
+-----------------------------+
| bc                          |
+-----------------------------+

MySQL > select split_part("2019年7月8日", "月", 1);
+-----------------------------------------+
| split_part('2019年7月8日', '月', 1)     |
+-----------------------------------------+
| 2019年7                                 |
+-----------------------------------------+
```

Keyword

SPLIT_PART,SPLIT,PART

Starts_with

Эта функция возвращает 1, если строка начинается с указанного префикса. В противном случае возвращает 0. Если аргумент равен NULL, результат будет NULL.

Синтаксис

```
BOOLEAN starts_with(VARCHAR str, VARCHAR prefix)
```

Примеры

```
mysql> select starts_with("hello world","hello");
+-------------------------------------+
|starts_with('hello world', 'hello')  |
+-------------------------------------+
| 1                                   |
+-------------------------------------+

mysql> select starts_with("hello world","world");
+-------------------------------------+
|starts_with('hello world', 'world')  |
+-------------------------------------+
| 0                                   |
+-------------------------------------+
```

Keyword

START_WITH

Str_to_map

Разделяет заданную строку на пары ключ-значение с использованием двух разделителей и возвращает map из разделённых пар.

Эта функция поддерживается начиная с версии 3.1.

Синтаксис

```
MAP<VARCHAR, VARCHAR> str_to_map(VARCHAR content[, VARCHAR delimiter[,
VARCHAR map_delimiter]])
```

Параметры

- `content`: обязательный, строковое выражение для разделения.

- `delimiter`: необязательный, разделитель, используемый для разделения `content` на пары ключ-значение, по умолчанию `,`.
- `map_delimiter`: необязательный, разделитель, используемый для отделения каждой пары ключ-значение, по умолчанию `:`.

Возвращаемое значение

Возвращает MAP из элементов STRING. Любой входной параметр null приводит к результату NULL.

Примеры

```
mysql> SELECT str_to_map('a:1|b:2|c:3', '|', ':') as map;
+---------------------------+
| map                       |
+---------------------------+
| {"a":"1","b":"2","c":"3"} |
+---------------------------+

mysql> SELECT str_to_map('a:1;b:2;c:3', ';', ':') as map;
+---------------------------+
| map                       |
+---------------------------+
| {"a":"1","b":"2","c":"3"} |
+---------------------------+

mysql> SELECT str_to_map('a:1,b:2,c:3', ',', ':') as map;
+---------------------------+
| map                       |
+---------------------------+
| {"a":"1","b":"2","c":"3"} |
+---------------------------+

mysql> SELECT str_to_map('a') as map;
+------------+
| map        |
+------------+
| {"a":null} |
+------------+

mysql> SELECT str_to_map('a:1,b:2,c:3',null, ':') as map;
+------+
| map  |
+------+
| NULL |
+------+

mysql> SELECT str_to_map('a:1,b:2,c:null') as map;
+------------------------------+
| map                          |
+------------------------------+
| {"a":"1","b":"2","c":"null"} |
+------------------------------+
```

Keywords

STR_TO_MAP

Strcmp

Эта функция сравнивает две строки. Возвращает 0, если lhs и rhs равны. Возвращает -1, если lhs предшествует rhs в лексикографическом порядке. Возвращает 1, если lhs следует после rhs в лексикографическом порядке. Если аргумент равен NULL, результат будет NULL.

Синтаксис

```
INT strcmp(VARCHAR lhs, VARCHAR rhs)
```

Примеры

```
mysql> select strcmp("test1", "test1");
+--------------------------+
| strcmp('test1', 'test1') |
+--------------------------+
|                        0 |
+--------------------------+

mysql> select strcmp("test1", "test2");
+--------------------------+
| strcmp('test1', 'test2') |
+--------------------------+
|                       -1 |
+--------------------------+

mysql> select strcmp("test2", "test1");
+--------------------------+
| strcmp('test2', 'test1') |
+--------------------------+
|                        1 |
+--------------------------+

mysql> select strcmp("test1", NULL);
+-----------------------+
| strcmp('test1', NULL) |
+-----------------------+
|                  NULL |
+-----------------------+
```

Keyword

STRCMP

Strleft

Эта функция извлекает указанное количество символов из строки (начиная слева). Единица измерения длины: символ utf8. Примечание: Эта функция также называется left.

Синтаксис

```
VARCHAR strleft(VARCHAR str,INT len)
```

Примеры

```
MySQL > select strleft("Hello selena",5);
+-------------------------------+
| strleft('Hello selena', 5) |
+-------------------------------+
| Hello                         |
+-------------------------------+
```

Keyword

STRLEFT

Strpos

Возвращает позицию N-го вхождения подстроки в строке. Если N — отрицательное число, поиск начинается с конца строки. Позиции начинаются с 1. Если подстрока не найдена, возвращается 0.

Синтаксис

```
INT strpos(VARCHAR str, VARCHAR substr [, INT instance])
```

Параметры

- `str`: Строка, в которой выполняется поиск подстроки.
- `substr`: Искомая подстрока.
- `instance`: N-е вхождение подстроки для поиска. Если задано отрицательное значение, поиск начинается с конца строки. Значение по умолчанию: `1`.

Возвращаемое значение

Возвращает целое число. `0` возвращается, если подстрока не найдена.

Примеры

```
SELECT strpos('hello world', 'world');
+-----------------------------+
| strpos('hello world', 'world') |
+-----------------------------+
|                           7 |
+-----------------------------+

SELECT strpos('Hello World', 'world');
+-----------------------------+
| strpos('Hello World', 'world') |
+-----------------------------+
|                           0 |
+-----------------------------+

SELECT strpos('hello world hello', 'hello', 2);
+---------------------------------------+
| strpos('hello world hello', 'hello', 2) |
+---------------------------------------+
|                                    13 |
+---------------------------------------+

SELECT strpos('Selena', 'Spark');
+----------------------------+
| strpos('Selena', 'Spark') |
+----------------------------+
|                          0 |
+----------------------------+

SELECT strpos(NULL, 'test');
+--------------------+
| strpos(NULL, 'test') |
+--------------------+
|               NULL |
+--------------------+
```

Keywords

STRPOS, STRING, POSITION, SUBSTRING

Strright

Эта функция извлекает указанное количество символов из строки (начиная справа). Единица измерения длины: символ utf-8. Примечание: Эта функция также называется right.

Синтаксис

```
VARCHAR strright(VARCHAR str,INT len)
```

Примеры

```
MySQL > select strright("Hello selena",9);
+--------------------------------+
| strright('Hello selena', 9) |
+--------------------------------+
| selena                       |
+--------------------------------+
```

Keyword

STRRIGHT

Substring, substr

Извлекает символы, начиная с указанной позиции, и возвращает подстроку указанной длины.

Синтаксис

```
VARCHAR substr(VARCHAR str, pos[, len])
```

Параметры

- `str`: обязательный, строка для извлечения символов. Должно быть значение VARCHAR.
- `pos`: обязательный, целое число, указывающее начальную позицию. Обратите внимание, что первый символ в строке имеет позицию 1, а не 0.
  - Если `pos` равен 0, возвращается пустая строка.

  - `pos` может быть отрицательным целым числом. В этом случае функция извлекает символы, начиная с конца строки. См. Пример 2.
  - Если позиция, указанная в `pos`, выходит за пределы строки, возвращается пустая строка. См. Пример 3.
- `len`: необязательный, положительное целое число, указывающее количество извлекаемых символов.
  - Если `len` указан, функция извлекает `len` символов, начиная с позиции, указанной в `pos`.
  - Если `len` не указан, функция извлекает все символы, начиная с `pos`. См. Пример 1.
  - Если `len` превышает фактическую длину совпадающих символов, возвращаются все совпадающие символы. См. Пример 4.

Возвращаемое значение

Возвращает значение типа VARCHAR.

Примеры

```
-- Извлечь все символы, начиная с первого символа "s".
MySQL > select substring("selenacluster", 1);
+-------------------------------------+
| substring('selenacluster', 1)    |
+-------------------------------------+
| selenacluster                    |
+-------------------------------------+

-- Извлечь первые 9 символов, начиная с первого символа "s".
MySQL > select substring("selenacluster", 1, 9);
+-------------------------------------+
| substring('selenacluster', 1, 9) |
+-------------------------------------+
| selena                           |
+-------------------------------------+

-- Позиция отрицательная, отсчёт ведётся с конца строки.
MySQL > select substring("selena", -5, 5);
+--------------------------------+
| substring('selena', -5, 5) |
+--------------------------------+
| rocks                          |
+--------------------------------+

-- Позиция превышает длину строки, возвращается пустая строка.
MySQL > select substring("apple", 8, 2);
+---------------------------------+
| substring('apple', 8, 2)       |
+---------------------------------+
|                                 |
+---------------------------------+

-- Есть 5 совпадающих символов. Длина 9 превышает длину совпадающих
символов, и возвращаются все совпадающие символы.
MySQL > select substring("apple", 1, 9);
+--------------------------+
| substring('apple', 1, 9) |
+--------------------------+
| apple                    |
+--------------------------+
```

Keyword

substring,string,sub

Substring_index

Извлекает подстроку, которая предшествует или следует за `count` вхождениями разделителя.

- Если `count` положительный, отсчёт начинается с начала строки, и функция возвращает подстроку, предшествующую `count`-му разделителю. Например, `select substring_index('https://www.selena.io', '.', 2);` возвращает подстроку перед вторым разделителем `.`, то есть `https://www.selena`.
- Если `count` отрицательный, отсчёт начинается с конца строки, и функция возвращает подстроку, следующую за `count`-м разделителем. Например, `select substring_index('https://www.selena.io', '.', -2);` возвращает подстроку после второго разделителя `.`, то есть `selena.io`.

Если любой входной параметр равен null, возвращается NULL.

Эта функция поддерживается начиная с версии 3.2.

Синтаксис

```
VARCHAR substring_index(VARCHAR str, VARCHAR delimiter, INT count)
```

Параметры

- `str`: обязательный, строка для разделения.
- `delimiter`: обязательный, разделитель, используемый для разделения строки.
- `count`: обязательный, позиция разделителя. Значение не может быть 0. В противном случае возвращается NULL. Если значение больше фактического количества разделителей в строке, возвращается вся строка.

Возвращаемое значение

Возвращает значение VARCHAR.

Примеры

```
-- Вернуть подстроку, предшествующую второму разделителю ".".
mysql> select substring_index('https://www.selena.io', '.', 2);
+-----------------------------------------------------+
| substring_index('https://www.selena.io', '.', 2) |
+-----------------------------------------------------+
| https://www.selena                                  |
+-----------------------------------------------------+

-- count отрицательный.
mysql> select substring_index('https://www.selena.io', '.', -2);
+------------------------------------------------------+
| substring_index('https://www.selena.io', '.', -2) |
+------------------------------------------------------+
| selena.io                                            |
+------------------------------------------------------+

mysql> select substring_index("hello world", " ", 1);
+----------------------------------------+
| substring_index("hello world", " ", 1) |
+----------------------------------------+
| hello                                  |
+----------------------------------------+

mysql> select substring_index("hello world", " ", -1);
+-----------------------------------------+
| substring_index('hello world', ' ', -1) |
+-----------------------------------------+
| world                                   |
+-----------------------------------------+

-- count равен 0, возвращается NULL.
mysql> select substring_index("hello world", " ", 0);
+----------------------------------------+
| substring_index('hello world', ' ', 0) |
+----------------------------------------+
| NULL                                   |
+----------------------------------------+

-- count больше количества пробелов в строке, возвращается вся строка.
mysql> select substring_index("hello world", " ", 2);
+----------------------------------------+
| substring_index("hello world", " ", 2) |
+----------------------------------------+
```

```
| hello world                              |
+------------------------------------------+

-- count больше количества пробелов в строке, возвращается вся строка.
mysql> select substring_index("hello world", " ", -2);
+------------------------------------------+
| substring_index("hello world", " ", -2) |
+------------------------------------------+
| hello world                              |
+------------------------------------------+
```

Keyword

substring_index

Tokenize

Разделяет и разбирает текст на токены на основе указанного токенизатора.

Синтаксис

```
ARRRY<VARCHAR> tokenize(VARCHAR tokenizer_name, VARCHAR content);
```

Параметры

- `tokenizer_name`: Используемый токенизатор. Допустимые значения: `english`, `standard` и `chinese`.
- `content`: Текст для токенизации. Может быть константной строкой или именем столбца. Если указан столбец, он должен быть типа STRING или VARCHAR.

Возвращаемое значение

Возвращает массив VARCHAR.

Примеры

```
MYSQL > SELECT tokenize('english', 'Today is saturday');
+-----------------------------------------+
| tokenize('english', 'Today is saturday') |
+-----------------------------------------+
| ["today","is","saturday"]                |
+-----------------------------------------+
```

Keyword

TOKENIZE

Translate

Заменяет указанные символы в строке. Функция принимает строку (`source`) в качестве входных данных и заменяет символы `from_string` в `source` на `to_string`.

Эта функция поддерживается начиная с версии 3.2.

Синтаксис

```
TRANSLATE(source, from_string, to_string)
```

Параметры

- `source`: поддерживает тип `VARCHAR`. Исходная строка для преобразования. Если символ в `source` не найден в `from_string`, он просто включается в результирующую строку.

- `from_string`: поддерживает тип `VARCHAR`. Каждый символ в `from_string` либо заменяется соответствующим символом из `to_string`, либо, если соответствующего символа нет (то есть если `to_string` содержит меньше символов, чем `from_string`), символ исключается из результирующей строки. См. Примеры 2 и 3. Если символ встречается несколько раз в `from_string`, эффективным является только его первое вхождение. См. Пример 5.

- `to_string`: поддерживает тип `VARCHAR`. Строка, используемая для замены символов. Если в `to_string` указано больше символов, чем в аргументе `from_string`, лишние

символы из `to_string` игнорируются. См. Пример 4.

Возвращаемое значение

Возвращает значение типа `VARCHAR`.

Сценарии, когда результат равен `NULL`:

- Любой из входных параметров равен `NULL`.
- Длина результирующей строки после преобразования превышает максимальную длину (1048576) для `VARCHAR`.

Примеры

```
-- Заменить 'ab' в исходной строке на '12'.
mysql > select translate('abcabc', 'ab', '12') as test;
+--------+
| test   |
+--------+
| 12c12c |
+--------+

-- Заменить 'mf1' в исходной строке на 'to'. 'to' содержит меньше символов,
чем 'mf1', и '1' исключается из результирующей строки.
mysql > select translate('s1m1a1rrfcks','mf1','to') as test;
+-----------+
| test      |
+-----------+
| selena |
+-----------+

-- Заменить 'ab' в исходной строке на '1'. '1' содержит меньше символов,
чем 'ab', и 'b' исключается из результирующей строки.
mysql > select translate('abcabc', 'ab', '1') as test;
+------+
| test |
+------+
| 1c1c |
+------+

-- Заменить 'ab' в исходной строке на '123'. '123' содержит больше
символов, чем 'ab', и '3' игнорируется.
mysql > select translate('abcabc', 'ab', '123') as test;
+--------+
| test   |
+--------+
| 12c12c |
+--------+

-- Заменить 'aba' в исходной строке на '123'. 'a' встречается дважды, и
заменяется только первое вхождение 'a'.
mysql > select translate('abcabc', 'aba', '123') as test;
+--------+
| test   |
+--------+
| 12c12c |
+--------+
```

```
-- Использование этой функции с repeat() и concat(). Результирующая строка
превышает максимальную длину VARCHAR, и возвращается NULL.
mysql > select translate(concat('bcde', repeat('a', 1024*1024-3)), 'a',
'z') as test;
+--------+
| test   |
+--------+
| NULL   |
+--------+

-- Использование этой функции с length(), repeat() и concat() для
вычисления длины результирующей строки.
mysql > select length(translate(concat('bcd', repeat('a', 1024*1024-3)),
'a', 'z')) as test;
+---------+
| test    |
+---------+
| 1048576 |
+---------+
```

См. также

- concat
- length
- repeat

Trim

Удаляет последовательные пробелы или указанные символы с начала и конца аргумента `str`. Удаление указанных символов поддерживается начиная с Selena 2.5.0.

Синтаксис

```
VARCHAR trim(VARCHAR str[, VARCHAR characters])
```

Параметры

`str`: обязательный, строка для обрезки, которая должна быть значением VARCHAR.

`characters`: необязательный, символы для удаления, должно быть значением VARCHAR. Если этот параметр не указан, по умолчанию из строки удаляются пробелы. Если этот

параметр задан как пустая строка, возвращается ошибка.

Возвращаемое значение

Возвращает значение VARCHAR.

Примеры

Пример 1: Удалить пять пробелов с начала и конца строки.

```
MySQL > SELECT trim("   ab c  ");
+-------------------+
| trim('   ab c  ') |
+-------------------+
| ab c              |
+-------------------+
1 row in set (0.00 sec)
```

Пример 2: Удалить указанные символы с начала и конца строки.

```
MySQL > SELECT trim("abcd", "ad");
+--------------------+
| trim('abcd', 'ad') |
+--------------------+
| bc                 |
+--------------------+

MySQL > SELECT trim("xxabcdxx", "x");
+-----------------------+
| trim('xxabcdxx', 'x') |
+-----------------------+
| abcd                  |
+-----------------------+
```

Ссылки

- ltrim
- rtrim

Ucase

Эта функция преобразует строку в верхний регистр. Она аналогична функции upper.

Синтаксис

```
VARCHAR ucase(VARCHAR str)
```

Примеры

```
mysql> SELECT ucase("AbC123");
+-----------------+
|ucase('AbC123')  |
+-----------------+
|ABC123           |
+-----------------+
```

Keyword

UCASE

Unhex

Эта функция выполняет операцию, обратную hex().

Она интерпретирует каждую пару шестнадцатеричных цифр во входной строке как число и преобразует его в байт, представленный этим числом. Возвращаемое значение — двоичная строка.

Синтаксис

```
UNHEX(str);
```

Параметры

`str`: строка для преобразования. Поддерживаемый тип данных — VARCHAR. Пустая строка возвращается в любой из следующих ситуаций:

- Длина строки равна 0 или количество символов в строке нечётное.
- Строка содержит символы, отличные от `[0-9]`, `[a-z]` и `[A-Z]`.

Возвращаемое значение

Возвращает значение типа VARCHAR.

Примеры

```
mysql> select unhex('33');
+-------------+
| unhex('33') |
+-------------+
| 3           |
+-------------+

mysql> select unhex('6170706C65');
+---------------------+
| unhex('6170706C65') |
+---------------------+
| apple               |
+---------------------+

mysql> select unhex('4142@');
+----------------+
| unhex('4142@') |
+----------------+
|                |
+----------------+
1 row in set (0.01 sec)
```

Keywords

UNHEX

Upper

Преобразует строку в верхний регистр.

Синтаксис

```
upper(str)
```

Параметры

- `str`: строка для преобразования. Если `str` не является строковым типом, сначала будет выполнено неявное преобразование.

Возвращаемое значение

Возвращает строку в верхнем регистре.

Примеры

```
MySQL [test]> select C_String, upper(C_String) from ex_iceberg_tbl;
+-------------------+-------------------+
| C_String          | upper(C_String)   |
+-------------------+-------------------+
| Hello, Selena! | HELLO, SELENA! |
| Hello, World!     | HELLO, WORLD!     |
+-------------------+-------------------+
```

Url_decode

Преобразует строку обратно из формата application/x-www-form-urlencoded. Эта функция является обратной к url_encode.

Эта функция поддерживается начиная с версии 3.2.

Синтаксис

```
VARCHAR url_decode(VARCHAR str)
```

Параметры

- `str`: строка для декодирования. Если `str` не является строкой, система сначала попытается выполнить неявное преобразование.

Возвращаемое значение

Возвращает декодированную строку.

Примеры

```
mysql> select
url_decode('https%3A%2F%2Fdocs.selena.io%2Fdocs%2Fintroduction%2FSelena_intro
+-------------------------------------------------------------------------------
--------+
|
url_decode('https%3A%2F%2Fdocs.selena.io%2Fdocs%2Fintroduction%2FSelena_intro
|
+-------------------------------------------------------------------------------
--------+
| https://docs.selena.io/docs/introduction/Selena_intro/
|
+-------------------------------------------------------------------------------
--------+
```

Url_encode

Преобразует строку в формат application/x-www-form-urlencoded.

Эта функция поддерживается начиная с версии 3.2.

Синтаксис

```
VARCHAR url_encode(VARCHAR str)
```

Параметры

- `str`: строка для кодирования. Если `str` не является строкой, функция сначала попытается выполнить неявное преобразование.

Возвращаемое значение

Возвращает закодированную строку, соответствующую формату application/x-www-form-urlencoded.

Примеры

```
mysql> select
url_encode('https://docs.selena.io/docs/introduction/Selena_intro/');
+-----------------------------------------------------------------------------
--+
| url_encode('https://docs.selena.io/docs/introduction/Selena_intro/') |
+-----------------------------------------------------------------------------
--+
| https%3A%2F%2Fdocs.selena.io%2Fdocs%2Fintroduction%2FSelena_intro%2F |
+-----------------------------------------------------------------------------
--+
```

Url_extract_host

Извлекает раздел хоста из URL.

Эта функция поддерживается начиная с версии 3.3.

Синтаксис

```
VARCHAR url_extract_host(VARCHAR str)
```

Параметры

- `str`: строка для извлечения хоста. Если `str` не является строкой, функция сначала попытается выполнить неявное преобразование.

Возвращаемое значение

Возвращает строку хоста.

Примеры

```
mysql> select url_extract_host('httpa://selena.com/test/api/v1');
+-------------------------------------------------------+
| url_extract_host('httpa://selena.com/test/api/v1') |
+-------------------------------------------------------+
| selena.com                                            |
+-------------------------------------------------------+
```

Url_extract_parameter

Извлекает значение запрошенного параметра `name` из строки запроса URL. Извлечение параметров выполняется в соответствии с RFC 1866#section-8.2.1. Если указанное имя параметра не существует, возвращается NULL.

Эта функция поддерживается начиная с версии 3.2.

Синтаксис

```
VARCHAR url_extract_parameter(VARCHAR str, VARCHAR name)
```

Параметры

- `str`: строка URL для извлечения параметров.
- `name`: имя параметра в строке запроса.

Возвращаемое значение

Возвращает значение VARCHAR.

Примеры

```
mysql> select url_extract_parameter("https://selena.io/doc?k1=10&k2=3&k1=100"
+-------------------------------------------------------------------------------+
| url_extract_parameter('https://selena.io/doc?k1=10&k2=3&k1=100', 'k1') |
+-------------------------------------------------------------------------------+
| 10                                                                            |
+-------------------------------------------------------------------------------+

mysql> select url_extract_parameter('https://selena.com/doc?
k0=10&k1=%21%23%24%26%27%28%29%2A%2B%2C%2F%3A%3B%3D%3F%40%5B%5D%20%22%25%2D%2
+-------------------------------------------------------------------------------
-----------------------------------------------------+
| url_extract_parameter('https://selena.com/doc?
k0=10&k1=%21%23%24%26%27%28%29%2A%2B%2C%2F%3A%3B%3D%3F%40%5B%5D%20%22%25%2D%2
|
+-------------------------------------------------------------------------------
-----------------------------------------------------+
| !#$&'()*+,/:;=?@[] "%-.<>\^_`{|}~
|
+-------------------------------------------------------------------------------
-----------------------------------------------------+
```

2.25 Struct

Named_struct

Создает структуру с указанными именами полей и значениями.

Эта функция поддерживается начиная с версии 3.1.

Синтаксис

```
STRUCT named_struct({STRING name1, ANY val1} [, ...] )
```

Параметры

- `nameN`: поле типа STRING.
- `valN`: выражение любого типа, задающее значение для поля N. Значения могут быть NULL.

Выражения имен и значений должны быть парными. В противном случае структура не может быть создана. Необходимо передать как минимум одну пару имени поля и значения, разделенную запятой (`,`).

Возвращаемое значение

Возвращает значение типа STRUCT.

Примеры

```
SELECT named_struct('a', 1, 'b', 2, 'c', 3);
+--------------------------------------+
| named_struct('a', 1, 'b', 2, 'c', 3) |
+--------------------------------------+
| {"a":1,"b":2,"c":3}                  |
+--------------------------------------+

SELECT named_struct('a', null, 'b', 2, 'c', 3);
+-----------------------------------------+
| named_struct('a', null, 'b', 2, 'c', 3) |
+-----------------------------------------+
| {"a":null,"b":2,"c":3}                  |
+-----------------------------------------+
```

Ссылки

- Тип данных STRUCT
- row/struct

Row

Создает именованное значение STRUCT или ROW из заданных значений. Поддерживает безымянные структуры. Вам не нужно указывать имена полей. Selena автоматически генерирует имена столбцов, такие как `col1, col2,...`.

Эта функция поддерживается начиная с версии 3.1.

struct() является псевдонимом для row().

Синтаксис

```
STRUCT row(ANY val, ...)
```

Параметры

`val`: выражение любого поддерживаемого типа.

Эта функция принимает переменное количество аргументов. Необходимо передать как минимум один аргумент. `value` может быть NULL. Разделяйте несколько значений запятой (`,`).

Возвращаемое значение

Возвращает значение типа STRUCT, состоящее из входных значений.

Примеры

```
select row(1,"Apple","Pear");
+-----------------------------------------+
| row(1, 'Apple', 'Pear')                 |
+-----------------------------------------+
| {"col1":1,"col2":"Apple","col3":"Pear"} |
+-----------------------------------------+

select row("Apple", NULL);
+------------------------------+
| row('Apple', NULL)           |
+------------------------------+
| {"col1":"Apple","col2":null} |
+------------------------------+

select struct(1,2,3);
+------------------------------+
| row(1, 2, 3)                 |
+------------------------------+
| {"col1":1,"col2":2,"col3":3} |
+------------------------------+
```

Ссылки

- Тип данных STRUCT

- named_struct

2.26 Table

FILES

Определяет файлы данных в удаленном хранилище, используемые для загрузки и выгрузки данных:

- Загрузка или запрос данных из удаленной системы хранения
- Выгрузка данных в удаленную систему хранения

`FILES()` поддерживает следующие источники данных и форматы файлов:

- **Источники данных:**
  - HDFS
  - AWS S3
  - Google Cloud Storage
  - Другие S3-совместимые системы хранения
  - Microsoft Azure Blob Storage
  - NFS(NAS)
- **Форматы файлов:**
  - Parquet
  - ORC (поддерживается с версии 3.3)
  - CSV (поддерживается с версии 3.3)
  - Avro (поддерживается с версии 3.4.4, только для загрузки)

Начиная с версии 3.2, FILES() дополнительно поддерживает сложные типы данных, включая `ARRAY`, `JSON`, `MAP` и `STRUCT`, помимо базовых типов данных.

FILES() для загрузки

Начиная с версии 3.1.0, Selena поддерживает определение файлов только для чтения в удаленном хранилище с помощью табличной функции `FILES()`. Она может получить доступ к удаленному хранилищу с помощью свойств, связанных с путем к файлам, определить схему таблицы данных в файлах и вернуть строки данных. Вы можете напрямую запросить строки данных с помощью `SELECT`, загрузить строки данных в существующую таблицу с помощью `INSERT` или создать новую таблицу и загрузить в нее строки данных с помощью `CREATE TABLE AS SELECT`. Начиная с версии 3.3.4, вы также можете просмотреть схему файла данных с помощью `FILES()` и `DESC`.

Синтаксис

```
FILES( data_location , [data_format] [, schema_detect ] [,
StorageCredentialParams ] [, columns_from_path ] [, list_files_only ] [,
list_recursively])
```

Параметры

Все параметры задаются в виде пар `"key" = "value"`.

`data_location`

URI для доступа к файлам.

Вы можете указать путь или файл. Например, можно указать этот параметр как `"hdfs://<hdfs_host>:<hdfs_port>/user/data/tablename/20210411"` для загрузки файла данных с именем `20210411` из пути `/user/data/tablename` на сервере HDFS.

Вы также можете указать этот параметр как путь сохранения нескольких файлов данных с помощью подстановочных символов `?`, `*`, `[]`, `{}` или `^`. Например, можно указать этот параметр как `"hdfs://<hdfs_host>:<hdfs_port>/user/data/tablename/*/*"` или `"hdfs://<hdfs_host>:<hdfs_port>/user/data/tablename/dt=202104*/*"` для загрузки файлов данных из всех партиций или только партиций `202104` в пути `/user/data/tablename` на сервере HDFS.

> **ПРИМЕЧАНИЕ**
>
> Подстановочные символы также можно использовать для указания промежуточных путей.

- Для доступа к HDFS необходимо указать этот параметр как:

  ```
  "path" = "hdfs://<hdfs_host>:<hdfs_port>/<hdfs_path>"
  -- Пример: "path" = "hdfs://127.0.0.1:9000/path/file.parquet"
  ```

- Для доступа к AWS S3:

  - Если вы используете протокол S3, необходимо указать этот параметр как:

```
"path" = "s3://<s3_path>"
-- Пример: "path" = "s3://path/file.parquet"
```

- Если вы используете протокол S3A, необходимо указать этот параметр как:

```
"path" = "s3a://<s3_path>"
-- Пример: "path" = "s3a://path/file.parquet"
```

- Для доступа к Google Cloud Storage необходимо указать этот параметр как:

```
"path" = "s3a://<gcs_path>"
-- Пример: "path" = "s3a://path/file.parquet"
```

- Для доступа к Azure Blob Storage:
  - Если ваша учетная запись хранения разрешает доступ по HTTP, необходимо указать этот параметр как:

```
"path" =
"wasb://<container>@<storage_account>.blob.core.windows.net/<blob_path
-- Пример: "path" =
"wasb://testcontainer@testaccount.blob.core.windows.net/path/file.parc
```

  - Если ваша учетная запись хранения разрешает доступ по HTTPS, необходимо указать этот параметр как:

```
"path" =
"wasbs://<container>@<storage_account>.blob.core.windows.net/<blob_pat
-- Пример: "path" =
"wasbs://testcontainer@testaccount.blob.core.windows.net/path/file.par
```

- Для доступа к NFS(NAS):

```
"path" = "file:///<absolute_path>"
-- Пример: "path" = "file:///home/ubuntu/parquetfile/file.parquet"
```

> **ПРИМЕЧАНИЕ**
>
> Для доступа к файлам в NFS через протокол `file://` необходимо смонтировать устройство NAS как NFS в одном и том же каталоге на каждом узле BE или CN.

`data_format`

Формат файла данных. Допустимые значения:

- `parquet`
- `orc` (поддерживается с версии 3.3)
- `csv` (поддерживается с версии 3.3)
- `avro` (поддерживается с версии 3.4.4, только для загрузки)

Вы должны установить подробные опции для конкретных форматов файлов данных.

Когда `list_files_only` установлен в `true`, указывать `data_format` не требуется.

Parquet

Пример формата Parquet:

```
"format"="parquet",
"parquet.use_legacy_encoding" = "true",    -- только для выгрузки
"parquet.version" = "2.6"                  -- только для выгрузки
```

`parquet.use_legacy_encoding`

Управляет техникой кодирования, используемой для типов данных DATETIME и DECIMAL. Допустимые значения: `true` и `false` (по умолчанию). Это свойство поддерживается только для выгрузки данных.

Если этот параметр установлен в `true`:

- Для типа DATETIME система использует кодировку `INT96`.
- Для типа DECIMAL система использует кодировку `fixed_len_byte_array`.

Если этот параметр установлен в `false`:

- Для типа DATETIME система использует кодировку `INT64`.
- Для типа DECIMAL система использует кодировку `INT32` или `INT64`.

> **ПРИМЕЧАНИЕ**
>
> Для типа данных DECIMAL 128 доступна только кодировка `fixed_len_byte_array`. `parquet.use_legacy_encoding` не действует.

`parquet.version`

Управляет версией Parquet, в которую система выгружает данные. Поддерживается с версии 3.4.6. Допустимые значения: `1.0`, `2.4` и `2.6` (по умолчанию). Это свойство поддерживается только для выгрузки данных.

CSV

Пример формата CSV:

```
"format"="csv",
"csv.column_separator"="\\t",
"csv.enclose"='"',
"csv.skip_header"="1",
"csv.escape"="\\"
```

`csv.column_separator`

Указывает разделитель столбцов при формате данных CSV. Если вы не укажете этот параметр, по умолчанию используется `\\t`, что означает табуляцию. Указанный вами разделитель столбцов должен совпадать с разделителем столбцов, фактически используемым в файле данных. В противном случае задача загрузки завершится неудачей из-за недостаточного качества данных.

Задачи, использующие Files(), отправляются согласно протоколу MySQL. Selena и MySQL экранируют символы в запросах загрузки. Поэтому, если разделитель столбцов является невидимым символом, таким как табуляция, вы должны добавить обратную косую черту (`\`) перед разделителем столбцов. Например, вы должны ввести `\\t`, если разделитель столбцов - `\t`, и вы должны ввести `\\n`, если разделитель столбцов - `\n`. Файлы Apache

Hive™ используют `\x01` в качестве разделителя столбцов, поэтому вы должны ввести `\\x01`, если файл данных из Hive.

> **ПРИМЕЧАНИЕ**
>
> - Для данных CSV вы можете использовать строку UTF-8, такую как запятая (,), табуляция или вертикальная черта (|), длина которой не превышает 50 байт, в качестве текстового разделителя.
>
> > - Значения NULL обозначаются с помощью `\N`. Например, файл данных состоит из трех столбцов, и запись из этого файла данных содержит данные в первом и третьем столбцах, но не содержит данных во втором столбце. В этой ситуации вам нужно использовать `\N` во втором столбце для обозначения значения NULL. Это означает, что запись должна быть скомпилирована как `a,\N,b`, а не `a,,b`. `a,,b` означает, что второй столбец записи содержит пустую строку.

`csv.enclose`

Указывает символ, который используется для обертывания значений полей в файле данных согласно RFC4180, когда формат данных CSV. Тип: однобайтовый символ. Значение по умолчанию: `NONE`. Наиболее распространенными символами являются одинарная кавычка (`'`) и двойная кавычка (`"`).

Все специальные символы (включая разделители строк и разделители столбцов), обернутые с помощью символа, указанного в `enclose`, считаются обычными символами. Selena может делать больше, чем RFC4180, так как позволяет указать любой однобайтовый символ в качестве символа `enclose`.

Если значение поля содержит символ, указанный в `enclose`, вы можете использовать тот же символ для экранирования этого символа `enclose`. Например, вы установили `enclose` в `"`, а значение поля - `a "quoted" c`. В этом случае вы можете ввести значение поля как `"a ""quoted"" c"` в файл данных.

`csv.skip_header`

Указывает количество строк заголовка, которые нужно пропустить в данных формата CSV. Тип: `INTEGER`. Значение по умолчанию: `0`.

В некоторых файлах данных формата CSV несколько строк заголовка используются для определения метаданных, таких как имена столбцов и типы данных столбцов. Устанавливая параметр `skip_header`, вы можете настроить Selena на пропуск этих строк заголовка. Например, если вы установите этот параметр в `1`, Selena пропустит первую строку файла данных при загрузке данных.

Строки заголовка в файле данных должны быть разделены с помощью разделителя строк, который вы указываете в операторе загрузки.

`csv.escape`

Указывает символ, который используется для экранирования различных специальных символов, таких как разделители строк, разделители столбцов, символы экранирования и символы, указанные в `enclose`, которые затем рассматриваются Selena как обычные символы и анализируются как часть значений полей, в которых они находятся. Тип: однобайтовый символ. Значение по умолчанию: `NONE`. Наиболее распространенным символом является косая черта (`\`), которая должна быть записана как двойная косая черта (`\\`) в операторах SQL.

> **ПРИМЕЧАНИЕ**
>
> Символ, указанный в `escape`, применяется как внутри, так и снаружи каждой пары символов, указанных в `enclose`.
>
> > Два примера:
> >
> > - Когда вы устанавливаете `enclose` в `"` и `escape` в `\`, Selena анализирует `"say \"Hello world\""` в `say "Hello world"`.
> > - Предположим, что разделитель столбцов - запятая (`,`). Когда вы устанавливаете `escape` в `\`, Selena анализирует `a, b\, c` в два отдельных значения полей: `a` и `b, c`.

`schema_detect`

Начиная с версии 3.2, `FILES()` поддерживает автоматическое определение схемы и объединение одной и той же партии файлов данных. Selena сначала определяет схему данных путем выборки определенных строк данных из случайного файла данных в партии. Затем Selena объединяет столбцы из всех файлов данных в партии.

Вы можете настроить правило выборки с помощью следующих параметров:

- `auto_detect_sample_files`: количество случайных файлов данных для выборки в каждой партии. По умолчанию выбираются первый и последний файлы. Диапазон: `[0, + ∞]`. По умолчанию: `2`.
- `auto_detect_sample_rows`: количество строк данных для сканирования в каждом выбранном файле данных. Диапазон: `[0, + ∞]`. По умолчанию: `500`.
- `auto_detect_types`: (действует только для CSV) - определять ли типы данных выбранных столбцов или просто предполагать String. `{true | false}`. По умолчанию: `true`.

После выборки Selena объединяет столбцы из всех файлов данных согласно следующим правилам:

- Для столбцов с разными именами или индексами каждый столбец идентифицируется как отдельный столбец, и в итоге возвращается объединение всех отдельных столбцов.
- Для столбцов с одинаковым именем, но разными типами данных они идентифицируются как один и тот же столбец, но с общим типом данных на относительно мелком уровне детализации. Например, если столбец `col1` в файле A имеет тип `INT`, но `DECIMAL` в файле B, в возвращаемом столбце используется тип `DOUBLE`.
    - Все целочисленные столбцы будут объединены в целочисленный тип на общем более грубом уровне детализации.
    - Целочисленные столбцы вместе со столбцами типа `FLOAT` будут объединены в тип DECIMAL.
    - Строковые типы используются для объединения других типов.
- Как правило, тип `STRING` может использоваться для объединения всех типов данных.
- Если автоопределение типов отключено, все столбцы будут возвращены как `STRING`

Вы можете обратиться к Примеру 5.

Если Selena не удается объединить все столбцы, она генерирует отчет об ошибке схемы, который включает информацию об ошибке и все схемы файлов.

> **IMPORTANT**
>
> Все файлы данных в одной партии должны быть одного формата файла.

Проверка схемы целевой таблицы с проталкиванием

Начиная с версии 3.4.0, система поддерживает проталкивание проверки схемы целевой таблицы на этап Scan функции `FILES()`.

Определение схемы `FILES()` не является полностью строгим. Например, любой целочисленный столбец в файлах CSV выводится и проверяется как тип BIGINT при чтении файлов функцией. В этом случае, если соответствующий столбец в целевой таблице имеет тип `TINYINT`, записи данных CSV, которые превышают тип BIGINT, не будут отфильтрованы. Вместо этого они будут неявно заполнены значением `NULL`.

Для решения этой проблемы система вводит динамический элемент конфигурации FE `files_enable_insert_push_down_schema` для управления тем, проталкивать ли проверку схемы целевой таблицы на этап Scan функции `FILES()`. Установив `files_enable_insert_push_down_schema` в `true`, система будет фильтровать записи данных, которые не проходят проверку схемы целевой таблицы при чтении файла.

Объединение файлов с разной схемой

Начиная с версии 3.4.0, система поддерживает объединение файлов с разной схемой, и по умолчанию возвращается ошибка при наличии несуществующих столбцов. Установив свойство `fill_mismatch_column_with` в `null`, вы можете разрешить системе назначать значения `NULL` несуществующим столбцам вместо возврата ошибки.

`fill_mismatch_column_with`: поведение системы после обнаружения несуществующего столбца при объединении файлов с разной схемой. Допустимые значения:

- `none`: возвращается ошибка, если обнаружен несуществующий столбец.
- `null`: значения NULL будут назначены несуществующему столбцу.

Например, читаемые файлы из разных партиций таблицы Hive, и Schema Change была выполнена на более новых партициях. При чтении как новых, так и старых партиций вы можете установить `fill_mismatch_column_with` в `null`, и система объединит схему файлов новых и старых партиций и назначит значения NULL несуществующим столбцам.

Система объединяет схему файлов Parquet и ORC на основе имен столбцов, а файлов CSV - на основе позиции (порядка) столбцов.

Вывод типа STRUCT из Parquet

Начиная с версии 3.4.0, `FILES()` поддерживает вывод данных типа `STRUCT` из файлов Parquet.

`StorageCredentialParams`

Информация для аутентификации, используемая Selena для доступа к вашей системе хранения.

Selena в настоящее время поддерживает доступ к HDFS с простой аутентификацией, доступ к AWS S3 и GCS с аутентификацией на основе пользователя IAM и доступ к Azure Blob Storage с помощью Shared Key, SAS Token, Managed Identity и Service Principal.

HDFS

- Использование простой аутентификации для доступа к HDFS:

```
"hadoop.security.authentication" = "simple",
"username" = "xxxxxxxxxx",
"password" = "yyyyyyyyyy"
```

Ключ	Обязательный	Описание
`hadoop.security.authentication`	Нет	Метод аутентификации. Допустимое значение: `simple` (по умолчанию). `simple` означает простую аутентификацию, то есть без аутентификации.
`username`	Да	Имя пользователя учетной записи, которую вы хотите использовать для доступа к NameNode cluster HDFS.
`password`	Да	Пароль учетной записи, которую вы хотите использовать для доступа к NameNode cluster HDFS.

- Использование аутентификации Kerberos для доступа к HDFS:

  В настоящее время FILES() поддерживает аутентификацию Kerberos с HDFS только через файл конфигурации **`hdfs-site.xml`**, размещенный в каталогах **`fe/conf`**, **`be/conf`** и **`cn/conf`**.

  Кроме того, вам нужно добавить следующую опцию в элемент конфигурации `JAVA_OPTS` в каждом файле конфигурации FE **`fe.conf`**, файле конфигурации BE **`be.conf`** и файле конфигурации CN **`cn.conf`**:

```
# Укажите локальный путь, по которому хранится файл конфигурации
Kerberos.
-Djava.security.krb5.conf=<path_to_kerberos_conf_file>
```

Пример:

```
JAVA_OPTS="-Xlog:gc*:${LOG_DIR}/be.gc.log.$DATE:time -
XX:ErrorFile=${LOG_DIR}/hs_err_pid%p.log -
Djava.security.krb5.conf=/etc/krb5.conf"
```

Вам также нужно выполнить команду `kinit` на каждом узле FE, BE и CN для получения Ticket Granting Ticket (TGT) от Key Distribution Center (KDC).

```
kinit -kt <path_to_keytab_file> <principal>
```

Для выполнения этой команды используемый principal должен иметь права на запись в ваш cluster HDFS. Кроме того, вам нужно настроить crontab для команды, чтобы запланировать задачу с определенным интервалом, предотвращая истечение срока аутентификации.

Пример:

```
# Обновлять TGT каждые 6 часов.
0 */6 * * * kinit -kt sr.keytab sr/test.selena.com@SELENA.COM >
/tmp/kinit.log
```

- Доступ к HDFS с включенным режимом HA:

  В настоящее время `FILES()` поддерживает доступ к HDFS с включенным режимом HA только через файл конфигурации **`hdfs-site.xml`**, размещенный в каталогах **`fe/conf`**, **`be/conf`** и **`cn/conf`**.

AWS S3

Если вы выбрали AWS S3 в качестве системы хранения, выполните одно из следующих действий:

- Для выбора метода аутентификации на основе профиля экземпляра настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.region" = "<aws_s3_region>"
```

- Для выбора метода аутентификации на основе принятой роли настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.iam_role_arn" = "<iam_role_arn>",
"aws.s3.region" = "<aws_s3_region>"
```

- Для выбора метода аутентификации на основе пользователя IAM настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "false",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>",
"aws.s3.region" = "<aws_s3_region>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
`aws.s3.use_instance_profile`	Да	Указывает, включать ли методы учетных данных профиля экземпляра и принятой роли. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`.
`aws.s3.iam_role_arn`	Нет	ARN роли IAM, которая имеет привилегии на вашем bucket AWS S3. Если вы выбрали принятую роль в качестве метода учетных данных для доступа к AWS S3, вы должны указать этот параметр.
`aws.s3.region`	Да	Регион, в котором находится ваш bucket AWS S3. Пример: `us-west-1`.
`aws.s3.access_key`	Нет	Ключ доступа вашего пользователя IAM. Если вы выбрали пользователя IAM в качестве метода учетных данных для доступа к AWS S3, вы должны указать этот параметр.

Параметр	Обязательный	Описание
`aws.s3.secret_key`	Нет	Секретный ключ вашего пользователя IAM. Если вы выбрали пользователя IAM в качестве метода учетных данных для доступа к AWS S3, вы должны указать этот параметр.

Информацию о том, как выбрать метод аутентификации для доступа к AWS S3 и как настроить политику управления доступом в AWS IAM Console, см. в разделе Параметры аутентификации для доступа к AWS S3.

Региональные конечные точки AWS STS

AWS Security Token Service (AWS STS) доступен как глобальная и региональная служба.

Параметр	Обязательный	Описание
`aws.s3.sts.region`	Нет	Регион AWS Security Token Service для доступа.
`aws.s3.sts.endpoint`	Нет	Используется для переопределения конечной точки AWS Security Token Service по умолчанию.

> **IMPORTANT**
>
> При использовании конечных точек AWS STS для аутентификации и доступа к данным в S3-совместимом хранилище за пределами S3 необходимо установить `aws.s3.use_instance_profile` в `false`.

Google GCS

Если вы выбрали Google GCS в качестве системы хранения, выполните одно из следующих действий:

- Для выбора метода аутентификации на основе VM настройте `StorageCredentialParams` следующим образом:

  ```
  "gcp.gcs.use_compute_engine_service_account" = "true"
  ```

  В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
`gcp.gcs.use_compute_engine_service_account`	false	true	Указывает, использовать ли напрямую сервисную учетную запись, привязанную к вашему Compute Engine.

- Для выбора метода аутентификации на основе сервисной учетной записи настройте `StorageCredentialParams` следующим образом:

```
"gcp.gcs.service_account_email" = "<google_service_account_email>",
"gcp.gcs.service_account_private_key_id" = "
<google_service_private_key_id>",
"gcp.gcs.service_account_private_key" = "<google_service_private_key>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
`gcp.gcs.service_account_email`	""	`"user@hello.iam.gserviceaccount.com"`	Адрес электронной почты в JSON-файле, созданном при создании сервисной учетной записи.
`gcp.gcs.service_account_private_key_id`	""	"61d257bd8479547cb3e04f0b9b6b9ca07af3b7ea"	Идентификатор закрытого ключа в JSON-файле, созданном при создании сервисной учетной записи.
`gcp.gcs.service_account_private_key`	""	"`-----BEGIN PRIVATE KEY-----xxxx-----END PRIVATE KEY-----\n`"	Закрытый ключ в JSON-файле, созданном при создании сервисной учетной записи.

- Для выбора метода аутентификации на основе имперсонации настройте `StorageCredentialParams` следующим образом:
  - Сделать экземпляр VM имперсонирующим сервисную учетную запись:

```
"gcp.gcs.use_compute_engine_service_account" = "true",
"gcp.gcs.impersonation_service_account" = "
<assumed_google_service_account_email>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
`gcp.gcs.use_compute_engine_service_account`	false	true	Указывает, использовать ли напрямую сервисную учетную запись, привязанную к вашему Compute Engine.
`gcp.gcs.impersonation_service_account`	""	"hello"	Сервисная учетная запись, которую вы хотите имперсонировать.

- Сделать сервисную учетную запись (называемую мета-сервисной учетной записью) имперсонирующей другую сервисную учетную запись (называемую сервисной учетной записью данных):

```
"gcp.gcs.service_account_email" = "<google_service_account_email>",
"gcp.gcs.service_account_private_key_id" = "
<meta_google_service_account_email>",
"gcp.gcs.service_account_private_key" = "
<meta_google_service_account_email>",
"gcp.gcs.impersonation_service_account" = "
<data_google_service_account_email>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
`gcp.gcs.service_account_email`	""	`"user@hello.iam.gserviceaccount.com"`	Адрес электронной почты в JSON-файле, созданном при создании мета-сервисной учетной записи.
`gcp.gcs.service_account_private_key_id`	""	"61d257bd8479547cb3e04f0b9b6b9ca07af3b7ea"	Идентификатор закрытого ключа в JSON-файле, созданном при создании мета-сервисной учетной записи.
`gcp.gcs.service_account_private_key`	""	"-----BEGIN PRIVATE KEY----xxxx-----END PRIVATE KEY-----\n"	Закрытый ключ в JSON-файле, созданном при создании мета-сервисной учетной записи.

Параметр	Значение по умолчанию	Пример значения	Описание
`gcp.gcs.impersonation_service_account`	""	"hello"	Сервисная учетная запись данных, которую вы хотите имперсонировать.

Azure Blob Storage

- Использование Shared Key для доступа к Azure Blob Storage:

```
"azure.blob.shared_key" = "<shared_key>"
```

Ключ	Обязательный	Описание
`azure.blob.shared_key`	Да	Shared Key, который вы можете использовать для доступа к учетной записи Azure Blob Storage.

- Использование SAS Token для доступа к Azure Blob Storage:

```
"azure.blob.sas_token" = "<storage_account_SAS_token>"
```

Ключ	Обязательный	Описание
`azure.blob.sas_token`	Да	SAS token, который вы можете использовать для доступа к учетной записи Azure Blob Storage.

- Использование Managed Identity для доступа к Azure Blob Storage (поддерживается с версии 3.4.4):

> **ПРИМЕЧАНИЕ**
>
> - Поддерживаются только User-assigned Managed Identities с учетными данными Client ID.
> - Динамическая конфигурация FE `azure_use_native_sdk` (по умолчанию: `true`) управляет тем, разрешать ли системе использовать аутентификацию с Managed Identity и Service Principals.

```
"azure.blob.oauth2_use_managed_identity" = "true",
"azure.blob.oauth2_client_id" = "<oauth2_client_id>"
```

Ключ	Обязательный	Описание
`azure.blob.oauth2_use_managed_identity`	Да	Использовать ли Managed Identity для доступа к учетной записи Azure Blob Storage. Установите в `true`.
`azure.blob.oauth2_client_id`	Да	Client ID Managed Identity, который вы можете использовать для доступа к учетной записи Azure Blob Storage.

- Использование Service Principal для доступа к Azure Blob Storage (поддерживается с версии 3.4.4):

> **ПРИМЕЧАНИЕ**
>
> - Поддерживаются только учетные данные Client Secret.
> - Динамическая конфигурация FE `azure_use_native_sdk` (по умолчанию: `true`) управляет тем, разрешать ли системе использовать аутентификацию с Managed Identity и Service Principals.

```
"azure.blob.oauth2_client_id" = "<oauth2_client_id>",
"azure.blob.oauth2_client_secret" = "<oauth2_client_secret>",
"azure.blob.oauth2_tenant_id" = "<oauth2_tenant_id>"
```

Ключ	Обязательный	Описание
`azure.blob.oauth2_client_id`	Да	Client ID Service Principal, который вы можете использовать для доступа к учетной записи Azure Blob Storage.
`azure.blob.oauth2_client_secret`	Да	Client Secret Service Principal, который вы можете использовать для доступа к учетной записи Azure Blob Storage.
`azure.blob.oauth2_tenant_id`	Да	Tenant ID Service Principal, который вы можете использовать для доступа к учетной записи Azure Blob Storage.

Azure Data Lake Storage Gen2

Если вы выбрали Data Lake Storage Gen2 в качестве системы хранения, выполните одно из следующих действий:

- Для выбора метода аутентификации Managed Identity настройте `StorageCredentialParams` следующим образом:

```
"azure.adls2.oauth2_use_managed_identity" = "true",
"azure.adls2.oauth2_tenant_id" = "<service_principal_tenant_id>",
"azure.adls2.oauth2_client_id" = "<service_client_id>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
`azure.adls2.oauth2_use_managed_identity`	Да	Указывает, включать ли метод аутентификации Managed Identity. Установите значение в `true`.
`azure.adls2.oauth2_tenant_id`	Да	ID tenant, к данным которого вы хотите получить доступ.
`azure.adls2.oauth2_client_id`	Да	Client (application) ID управляемой идентичности.

- Для выбора метода аутентификации Shared Key настройте `StorageCredentialParams` следующим образом:

```
"azure.adls2.storage_account" = "<storage_account_name>",
"azure.adls2.shared_key" = "<storage_account_shared_key>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
`azure.adls2.storage_account`	Да	Имя пользователя вашей учетной записи хранения Data Lake Storage Gen2.
`azure.adls2.shared_key`	Да	Shared key вашей учетной записи хранения Data Lake Storage Gen2.

- Для выбора метода аутентификации Service Principal настройте `StorageCredentialParams` следующим образом:

```
"azure.adls2.oauth2_client_id" = "<service_client_id>",
"azure.adls2.oauth2_client_secret" = "
<service_principal_client_secret>",
"azure.adls2.oauth2_client_endpoint" = "
<service_principal_client_endpoint>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
`azure.adls2.oauth2_client_id`	Да	Client (application) ID service principal.
`azure.adls2.oauth2_client_secret`	Да	Значение нового созданного client (application) secret.
`azure.adls2.oauth2_client_endpoint`	Да	Конечная точка токена OAuth 2.0 (v1) service principal или application.

Azure Data Lake Storage Gen1

Если вы выбрали Data Lake Storage Gen1 в качестве системы хранения, выполните одно из следующих действий:

- Для выбора метода аутентификации Managed Service Identity настройте `StorageCredentialParams` следующим образом:

```
"azure.adls1.use_managed_service_identity" = "true"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
`azure.adls1.use_managed_service_identity`	Да	Указывает, включать ли метод аутентификации Managed Service Identity. Установите значение в `true`.

- Для выбора метода аутентификации Service Principal настройте `StorageCredentialParams` следующим образом:

```
"azure.adls1.oauth2_client_id" = "<application_client_id>",
"azure.adls1.oauth2_credential" = "<application_client_credential>",
"azure.adls1.oauth2_endpoint" = "<OAuth_2.0_authorization_endpoint_v2>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
`azure.adls1.oauth2_client_id`	Да	Client (application) ID.
`azure.adls1.oauth2_credential`	Да	Значение нового созданного client (application) secret.
`azure.adls1.oauth2_endpoint`	Да	Конечная точка токена OAuth 2.0 (v1) service principal или application.

Другие S3-совместимые системы хранения

Если вы выбрали другую S3-совместимую систему хранения, такую как MinIO, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.enable_ssl" = "false",
"aws.s3.enable_path_style_access" = "true",
"aws.s3.endpoint" = "<s3_endpoint>",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
`aws.s3.enable_ssl`	Да	Указывает, включать ли SSL-соединение. Допустимые значения: `true` и `false`. Значение по умолчанию: `true`.
`aws.s3.enable_path_style_access`	Да	Указывает, включать ли доступ по URL в стиле пути. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`. Для MinIO необходимо установить значение в `true`.
`aws.s3.endpoint`	Да	Конечная точка, которая используется для подключения к вашей S3-совместимой системе хранения вместо AWS S3.

Параметр	Обязательный	Описание
`aws.s3.access_key`	Да	Ключ доступа вашего пользователя IAM.
`aws.s3.secret_key`	Да	Секретный ключ вашего пользователя IAM.

`columns_from_path`

Начиная с версии 3.2, Selena может извлекать значение пары ключ/значение из пути к файлу как значение столбца.

```
"columns_from_path" = "<column_name> [, ...]"
```

Предположим, файл данных **file1** хранится по пути в формате `/geo/country=US/city=LA/`. Вы можете указать параметр `columns_from_path` как `"columns_from_path" = "country, city"` для извлечения географической информации из пути к файлу в качестве значения возвращаемых столбцов. Дополнительные инструкции см. в Примере 4.

`list_files_only`

Начиная с версии 3.4.0, `FILES()` поддерживает только перечисление файлов при их чтении.

```
"list_files_only" = "true"
```

Обратите внимание, что вам не нужно указывать `data_format`, когда `list_files_only` установлен в `true`.

Подробнее см. в разделе Возвращаемое значение.

`list_recursively`

Selena дополнительно поддерживает `list_recursively` для рекурсивного перечисления файлов и каталогов. `list_recursively` действует только когда `list_files_only` установлен в `true`. Значение по умолчанию: `false`.

```
"list_files_only" = "true",
"list_recursively" = "true"
```

Когда и `list_files_only`, и `list_recursively` установлены в `true`, Selena выполнит следующее:

- Если указанный `path` является файлом (указанным явно или представленным подстановочными символами), Selena покажет информацию о файле.
- Если указанный `path` является каталогом (указанным явно или представленным подстановочными символами, и независимо от того, заканчивается ли он на `/`), Selena покажет все файлы и подкаталоги в этом каталоге.

Подробнее см. в разделе Возвращаемое значение.

Возвращаемое значение

`SELECT FROM FILES()`

При использовании с SELECT, FILES() возвращает данные в файле как таблицу.

- При запросе файлов CSV вы можете использовать `$1`, `$2` и т.д. для представления каждого столбца в операторе SELECT или указать `*` для получения данных из всех столбцов.

```
SELECT * FROM FILES(
    "path" = "s3://inserttest/csv/file1.csv",
    "format" = "csv",
    "csv.column_separator"=",",
    "csv.row_delimiter"="\n",
    "csv.enclose"='"',
    "csv.skip_header"="1",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
    "aws.s3.region" = "us-west-2"
)
WHERE $1 > 5;
+------+---------+------------+
| $1   | $2      | $3         |
+------+---------+------------+
|    6 | 0.34413 | 2017-11-25 |
|    7 | 0.40055 | 2017-11-26 |
|    8 | 0.42437 | 2017-11-27 |
|    9 | 0.67935 | 2017-11-27 |
|   10 | 0.22783 | 2017-11-29 |
+------+---------+------------+
5 rows in set (0.30 sec)

SELECT $1, $2 FROM FILES(
    "path" = "s3://inserttest/csv/file1.csv",
    "format" = "csv",
    "csv.column_separator"=",",
    "csv.row_delimiter"="\n",
    "csv.enclose"='"',
    "csv.skip_header"="1",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
    "aws.s3.region" = "us-west-2"
);
+------+---------+
| $1   | $2      |
+------+---------+
|    1 | 0.71173 |
|    2 | 0.16145 |
|    3 | 0.80524 |
|    4 | 0.91852 |
|    5 | 0.37766 |
|    6 | 0.34413 |
|    7 | 0.40055 |
|    8 | 0.42437 |
```

```
|     9 | 0.67935 |
|    10 | 0.22783 |
+------+---------+
10 rows in set (0.38 sec)
```

- При запросе файлов Parquet или ORC вы можете напрямую указать имя нужных столбцов в операторе SELECT или указать `*` для получения данных из всех столбцов.

```
SELECT * FROM FILES(
    "path" = "s3://inserttest/parquet/file2.parquet",
    "format" = "parquet",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
    "aws.s3.region" = "us-west-2"
)
WHERE c1 IN (101,105);
+------+------+---------------------+
| c1   | c2   | c3                  |
+------+------+---------------------+
|  101 |    9 | 2018-05-15T18:30:00 |
|  105 |    6 | 2018-05-15T18:30:00 |
+------+------+---------------------+
2 rows in set (0.29 sec)

SELECT c1, c3 FROM FILES(
    "path" = "s3://inserttest/parquet/file2.parquet",
    "format" = "parquet",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
    "aws.s3.region" = "us-west-2"
);
+------+---------------------+
| c1   | c3                  |
+------+---------------------+
|  101 | 2018-05-15T18:30:00 |
|  102 | 2018-05-15T18:30:00 |
|  103 | 2018-05-15T18:30:00 |
|  104 | 2018-05-15T18:30:00 |
|  105 | 2018-05-15T18:30:00 |
|  106 | 2018-05-15T18:30:00 |
|  107 | 2018-05-15T18:30:00 |
|  108 | 2018-05-15T18:30:00 |
|  109 | 2018-05-15T18:30:00 |
|  110 | 2018-05-15T18:30:00 |
+------+---------------------+
10 rows in set (0.55 sec)
```

- При запросе файлов с `list_files_only` установленным в `true`, система вернет `PATH`, `SIZE`, `IS_DIR` (является ли данный путь каталогом) и `MODIFICATION_TIME`.

```
SELECT * FROM FILES(
    "path" = "s3://bucket/*.parquet",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
    "list_files_only" = "true"
);
+-----------------------+------+--------+---------------------+
| PATH                  | SIZE | IS_DIR | MODIFICATION_TIME   |
+-----------------------+------+--------+---------------------+
| s3://bucket/1.parquet | 5221 |      0 | 2024-08-15 20:47:02 |
| s3://bucket/2.parquet | 5222 |      0 | 2024-08-15 20:54:57 |
| s3://bucket/3.parquet | 5223 |      0 | 2024-08-20 15:21:00 |
| s3://bucket/4.parquet | 5224 |      0 | 2024-08-15 11:32:14 |
+-----------------------+------+--------+---------------------+
4 rows in set (0.03 sec)
```

- При запросе файлов с `list_files_only` и `list_recursively` установленными в `true`, система рекурсивно перечислит файлы и каталоги.

  Предположим, путь `s3://bucket/list/` содержит следующие файлы и подкаталоги:

```
s3://bucket/list/
├── basic1.csv
├── basic2.csv
├── orc0
│   └── orc1
│       └── basic_type.orc
├── orc1
│   └── basic_type.orc
└── parquet
    └── basic_type.parquet
```

  Рекурсивное перечисление файлов и каталогов:

```
SELECT * FROM FILES(
    "path"="s3://bucket/list/",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
    "list_files_only" = "true",
    "list_recursively" = "true"
);
+---------------------------------------------+------+--------+--------
-------------+
| PATH                                        | SIZE | IS_DIR |
MODIFICATION_TIME   |
+---------------------------------------------+------+--------+--------
-------------+
| s3://bucket/list                            |    0 |      1 | 2024-
12-24 22:15:59 |
| s3://bucket/list/basic1.csv                 |   52 |      0 | 2024-
12-24 11:35:53 |
| s3://bucket/list/basic2.csv                 |   34 |      0 | 2024-
12-24 11:35:53 |
| s3://bucket/list/orc0                       |    0 |      1 | 2024-
12-24 11:35:53 |
| s3://bucket/list/orc0/orc1                  |    0 |      1 | 2024-
12-24 11:35:53 |
| s3://bucket/list/orc0/orc1/basic_type.orc   | 1027 |      0 | 2024-
12-24 11:35:53 |
| s3://bucket/list/orc1                       |    0 |      1 | 2024-
12-24 22:16:00 |
| s3://bucket/list/orc1/basic_type.orc        | 1027 |      0 | 2024-
12-24 22:16:00 |
| s3://bucket/list/parquet                    |    0 |      1 | 2024-
12-24 11:35:53 |
| s3://bucket/list/parquet/basic_type.parquet | 2281 |      0 | 2024-
12-24 11:35:53 |
+---------------------------------------------+------+--------+--------
-------------+
10 rows in set (0.04 sec)
```

Нерекурсивное перечисление файлов и каталогов, соответствующих `orc*` в этом пути:

```
SELECT * FROM FILES(
    "path"="s3://bucket/list/orc*",
    "list_files_only" = "true",
    "list_recursively" = "false"
);
+---------------------------------------+------+--------+---------------
------+
| PATH                                  | SIZE | IS_DIR |
MODIFICATION_TIME   |
+---------------------------------------+------+--------+---------------
------+
| s3://bucket/list/orc0/orc1            |    0 |      1 | 2024-12-24
11:35:53 |
| s3://bucket/list/orc1/basic_type.orc | 1027 |      0 | 2024-12-24
22:16:00 |
+---------------------------------------+------+--------+---------------
------+
2 rows in set (0.03 sec)
```

`DESC FILES()`

При использовании с `DESC`, `FILES()` возвращает схему файла.

```
DESC FILES(
    "path" = "s3://inserttest/lineorder.parquet",
    "format" = "parquet",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
    "aws.s3.region" = "us-west-2"
);

+------------------+------------------+------+
| Field            | Type             | Null |
+------------------+------------------+------+
| lo_orderkey      | int              | YES  |
| lo_linenumber    | int              | YES  |
| lo_custkey       | int              | YES  |
| lo_partkey       | int              | YES  |
| lo_suppkey       | int              | YES  |
| lo_orderdate     | int              | YES  |
| lo_orderpriority | varchar(1048576) | YES  |
| lo_shippriority  | int              | YES  |
| lo_quantity      | int              | YES  |
| lo_extendedprice | int              | YES  |
| lo_ordtotalprice | int              | YES  |
| lo_discount      | int              | YES  |
| lo_revenue       | int              | YES  |
| lo_supplycost    | int              | YES  |
| lo_tax           | int              | YES  |
| lo_commitdate    | int              | YES  |
| lo_shipmode      | varchar(1048576) | YES  |
+------------------+------------------+------+
17 rows in set (0.05 sec)
```

При просмотре файлов с `list_files_only` установленным в `true`, система вернет свойства `Type` и `Null` для `PATH`, `SIZE`, `IS_DIR` (является ли данный путь каталогом) и `MODIFICATION_TIME`.

```
DESC FILES(
    "path" = "s3://bucket/*.parquet",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
    "list_files_only" = "true"
);
+-------------------+------------------+------+
| Field             | Type             | Null |
+-------------------+------------------+------+
| PATH              | varchar(1048576) | YES  |
| SIZE              | bigint           | YES  |
| IS_DIR            | boolean          | YES  |
| MODIFICATION_TIME | datetime         | YES  |
+-------------------+------------------+------+
4 rows in set (0.00 sec)
```

FILES() для выгрузки

Начиная с версии 3.2.0, `FILES()` поддерживает запись данных в файлы в удаленном хранилище. Вы можете использовать `INSERT INTO FILES()` для выгрузки данных из Selena в удаленное хранилище.

Синтаксис

```
FILES( data_location , data_format [, StorageCredentialParams ] ,
unload_data_param )
```

Параметры

Все параметры задаются в виде пар `"key" = "value"`.

data_location

См. `FILES()` для загрузки - Параметры - data_location.

data_format

См. `FILES()` для загрузки - Параметры - data_format.

StorageCredentialParams

См. `FILES()` для загрузки - Параметры - StorageCredentialParams.

`unload_data_param`

```
unload_data_param ::=
    "compression" = { "uncompressed" | "gzip" | "snappy" | "zstd | "lz4" },
    "partition_by" = "<column_name> [, ...]",
    "single" = { "true" | "false" } ,
    "target_max_file_size" = "<int>"
```

Ключ	Обязательный	Описание
`compression`	Да	Метод сжатия для использования при выгрузке данных. Допустимые значения: • `uncompressed`: алгоритм сжатия не используется. • `gzip`: использовать алгоритм сжатия gzip. • `snappy`: использовать алгоритм сжатия SNAPPY. • `zstd`: использовать алгоритм сжатия Zstd. • `lz4`: использовать алгоритм сжатия LZ4. **ПРИМЕЧАНИЕ** Выгрузка в файлы CSV не поддерживает сжатие данных. Необходимо установить этот параметр в `uncompressed`.
`partition_by`	Нет	Список столбцов, используемых для разделения файлов данных на разные пути хранения. Несколько столбцов разделяются запятыми (,). `FILES()` извлекает информацию ключ/значение указанных столбцов и сохраняет файлы данных по путям хранения с извлеченной парой ключ/значение. Дополнительные инструкции см. в Примере 7.
`single`	Нет	Выгружать ли данные в один файл. Допустимые значения: • `true`: данные сохраняются в одном файле данных. • `false` (по умолчанию): данные сохраняются в нескольких файлах, если объем выгружаемых данных превышает 512 МБ.
`target_max_file_size`	Нет	Приблизительный максимальный размер каждого файла в партии для выгрузки. Единица измерения: байты. Значение по умолчанию: 1073741824 (1 ГБ). Когда размер выгружаемых данных превышает это значение, данные будут разделены на несколько файлов, и размер каждого файла не будет значительно превышать это значение. Введено в версии 3.2.7.

Примеры

Пример 1: Запрос данных из файла

Запрос данных из файла Parquet **parquet/par-dup.parquet** в bucket AWS S3 `inserttest`:

```
SELECT * FROM FILES(
     "path" = "s3://inserttest/parquet/par-dup.parquet",
     "format" = "parquet",
     "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
     "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
     "aws.s3.region" = "us-west-2"
);
+------+-------------------------------------------------------------+
| c1   | c2                                                          |
+------+-------------------------------------------------------------+
|    1 | {"1": "key", "1": "1", "111": "1111", "111": "aaaa"}        |
|    2 | {"2": "key", "2": "NULL", "222": "2222", "222": "bbbb"} |
+------+-------------------------------------------------------------+
2 rows in set (22.335 sec)
```

Запрос данных из файлов Parquet в NFS(NAS):

```
SELECT * FROM FILES(
  'path' = 'file:///home/ubuntu/parquetfile/*.parquet',
  'format' = 'parquet'
);
```

Пример 2: Вставка строк данных из файла

Вставка строк данных из файла Parquet **parquet/insert_wiki_edit_append.parquet** в bucket AWS S3 `inserttest` в таблицу `insert_wiki_edit`:

```
INSERT INTO insert_wiki_edit
    SELECT * FROM FILES(
        "path" = "s3://inserttest/parquet/insert_wiki_edit_append.parquet",
        "format" = "parquet",
        "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
        "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
        "aws.s3.region" = "us-west-2"
);
Query OK, 2 rows affected (23.03 sec)
{'label':'insert_d8d4b2ee-ac5c-11ed-a2cf-4e1110a8f63b', 'status':'VISIBLE',
'txnId':'2440'}
```

Вставка строк данных из файлов CSV в NFS(NAS) в таблицу `insert_wiki_edit`:

```
INSERT INTO insert_wiki_edit
  SELECT * FROM FILES(
    'path' = 'file:///home/ubuntu/csvfile/*.csv',
    'format' = 'csv',
    'csv.column_separator' = ',',
    'csv.row_delimiter' = '\n'
  );
```

Пример 3: CTAS со строками данных из файла

Создание таблицы с именем `ctas_wiki_edit` и вставка строк данных из файла Parquet **parquet/insert_wiki_edit_append.parquet** в bucket AWS S3 `inserttest` в таблицу:

```
CREATE TABLE ctas_wiki_edit AS
    SELECT * FROM FILES(
        "path" = "s3://inserttest/parquet/insert_wiki_edit_append.parquet",
        "format" = "parquet",
        "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
        "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
        "aws.s3.region" = "us-west-2"
);
Query OK, 2 rows affected (22.09 sec)
{'label':'insert_1a217d70-2f52-11ee-9e4a-7a563fb695da', 'status':'VISIBLE',
'txnId':'3248'}
```

Пример 4: Запрос данных из файла и извлечение информации ключ/значение из его пути

Запрос данных из файла Parquet **/geo/country=US/city=LA/file1.parquet** (который содержит только два столбца - `id` и `user`) и извлечение информации ключ/значение из его пути как возвращаемых столбцов.

```
SELECT * FROM FILES(
    "path" =
"hdfs://xxx.xx.xxx.xx:9000/geo/country=US/city=LA/file1.parquet",
    "format" = "parquet",
    "hadoop.security.authentication" = "simple",
    "username" = "xxxxx",
    "password" = "xxxxx",
    "columns_from_path" = "country, city"
);
+------+---------+---------+------+
| id   | user    | country | city |
+------+---------+---------+------+
|    1 | richard | US      | LA   |
|    2 | amber   | US      | LA   |
+------+---------+---------+------+
2 rows in set (3.84 sec)
```

Пример 5: Автоматическое определение схемы и объединение

Следующий пример основан на двух файлах Parquet в bucket S3:

- Файл 1 содержит три столбца - INT столбец `c1`, FLOAT столбец `c2` и DATE столбец `c3`.

```
c1,c2,c3
1,0.71173,2017-11-20
2,0.16145,2017-11-21
3,0.80524,2017-11-22
4,0.91852,2017-11-23
5,0.37766,2017-11-24
6,0.34413,2017-11-25
7,0.40055,2017-11-26
8,0.42437,2017-11-27
9,0.67935,2017-11-27
10,0.22783,2017-11-29
```

- Файл 2 содержит три столбца - INT столбец `c1`, INT столбец `c2` и DATETIME столбец `c3`.

```
c1,c2,c3
101,9,2018-05-15T18:30:00
102,3,2018-05-15T18:30:00
103,2,2018-05-15T18:30:00
104,3,2018-05-15T18:30:00
105,6,2018-05-15T18:30:00
106,1,2018-05-15T18:30:00
107,8,2018-05-15T18:30:00
108,5,2018-05-15T18:30:00
109,6,2018-05-15T18:30:00
110,8,2018-05-15T18:30:00
```

Использование оператора CTAS для создания таблицы с именем `test_ctas_parquet` и вставки строк данных из двух файлов Parquet в таблицу:

```
CREATE TABLE test_ctas_parquet AS
SELECT * FROM FILES(
    "path" = "s3://inserttest/parquet/*",
    "format" = "parquet",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
    "aws.s3.region" = "us-west-2"
);
```

Просмотр схемы таблицы `test_ctas_parquet`:

```
SHOW CREATE TABLE test_ctas_parquet\G
```

```
*************************** 1. row ***************************
       Table: test_ctas_parquet
Create Table: CREATE TABLE `test_ctas_parquet` (
  `c1` bigint(20) NULL COMMENT "",
  `c2` decimal(38, 9) NULL COMMENT "",
  `c3` varchar(1048576) NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`c1`, `c2`)
COMMENT "OLAP"
DISTRIBUTED BY RANDOM
PROPERTIES (
"bucket_size" = "4294967296",
"compression" = "LZ4",
"replication_num" = "3"
);
```

Результат показывает, что столбец `c2`, который содержит данные FLOAT и INT, объединен в столбец DECIMAL, а `c3`, который содержит данные DATE и DATETIME, объединен в столбец VARCHAR.

Вышеуказанный результат остается таким же, когда файлы Parquet заменены на файлы CSV, содержащие те же данные:

```
CREATE TABLE test_ctas_csv AS
  SELECT * FROM FILES(
    "path" = "s3://inserttest/csv/*",
    "format" = "csv",
    "csv.column_separator"=",",
    "csv.row_delimiter"="\n",
    "csv.enclose"='"',
    "csv.skip_header"="1",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
    "aws.s3.region" = "us-west-2"
);
Query OK, 0 rows affected (30.90 sec)

SHOW CREATE TABLE test_ctas_csv\G
*************************** 1. row ***************************
       Table: test_ctas_csv
Create Table: CREATE TABLE `test_ctas_csv` (
  `c1` bigint(20) NULL COMMENT "",
  `c2` decimal(38, 9) NULL COMMENT "",
  `c3` varchar(1048576) NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`c1`, `c2`)
COMMENT "OLAP"
DISTRIBUTED BY RANDOM
PROPERTIES (
"bucket_size" = "4294967296",
"compression" = "LZ4",
"replication_num" = "3"
);
1 row in set (0.27 sec)
```

- Объединение схемы файлов Parquet и разрешение системе назначать значения NULL несуществующим столбцам путем установки `fill_mismatch_column_with` в `null`:

```
SELECT * FROM FILES(
  "path" =
"s3://inserttest/basic_type.parquet,s3://inserttest/basic_type_k2k5k7.parquet
  "format" = "parquet",
  "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
  "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
  "aws.s3.region" = "us-west-2",
  "fill_mismatch_column_with" = "null"
);
+------+------+------+-------+------------+---------------------+------+-----
| k1   | k2   | k3   | k4    | k5         | k6                  | k7   | k8
|
+------+------+------+-------+------------+---------------------+------+-----
| NULL |   21 | NULL |  NULL | 2024-10-03 | NULL                | c    | NULL
|
|    0 |    1 |    2 |  3.20 | 2024-10-01 | 2024-10-01 12:12:12 | a    |  4.3
|
|    1 |   11 |   12 | 13.20 | 2024-10-02 | 2024-10-02 13:13:13 | b    | 14.3
|
+------+------+------+-------+------------+---------------------+------+-----
3 rows in set (0.03 sec)
```

Пример 6: Просмотр схемы файла

Просмотр схемы файла Parquet `lineorder`, хранящегося в AWS S3, с помощью DESC.

```
DESC FILES(
    "path" = "s3://inserttest/lineorder.parquet",
    "format" = "parquet",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
    "aws.s3.region" = "us-west-2"
);

+------------------+------------------+------+
| Field            | Type             | Null |
+------------------+------------------+------+
| lo_orderkey      | int              | YES  |
| lo_linenumber    | int              | YES  |
| lo_custkey       | int              | YES  |
| lo_partkey       | int              | YES  |
| lo_suppkey       | int              | YES  |
| lo_orderdate     | int              | YES  |
| lo_orderpriority | varchar(1048576) | YES  |
| lo_shippriority  | int              | YES  |
| lo_quantity      | int              | YES  |
| lo_extendedprice | int              | YES  |
| lo_ordtotalprice | int              | YES  |
| lo_discount      | int              | YES  |
| lo_revenue       | int              | YES  |
| lo_supplycost    | int              | YES  |
| lo_tax           | int              | YES  |
| lo_commitdate    | int              | YES  |
| lo_shipmode      | varchar(1048576) | YES  |
+------------------+------------------+------+
17 rows in set (0.05 sec)
```

Пример 7: Выгрузка данных

Выгрузка всех строк данных из `sales_records` в несколько файлов Parquet по пути **/unload/partitioned/** в cluster HDFS. Эти файлы сохраняются в разных подпутях, различающихся значениями в столбце `sales_time`.

```
INSERT INTO FILES(
    "path" = "hdfs://xxx.xx.xxx.xx:9000/unload/partitioned/",
    "format" = "parquet",
    "hadoop.security.authentication" = "simple",
    "username" = "xxxxx",
    "password" = "xxxxx",
    "compression" = "lz4",
    "partition_by" = "sales_time"
)
SELECT * FROM sales_records;
```

Выгрузка результатов запроса в файлы CSV и Parquet в NFS(NAS):

```
-- CSV
INSERT INTO FILES(
    'path' = 'file:///home/ubuntu/csvfile/',
    'format' = 'csv',
    'csv.column_separator' = ',',
    'csv.row_delimitor' = '\n'
)
SELECT * FROM sales_records;

-- Parquet
INSERT INTO FILES(
    'path' = 'file:///home/ubuntu/parquetfile/',
    'format' = 'parquet'
)
SELECT * FROM sales_records;
```

Пример 8: Файлы Avro

Загрузка файла Avro:

```
INSERT INTO avro_tbl
  SELECT * FROM FILES(
    "path" = "hdfs://xxx.xx.xx.x:yyyy/avro/primitive.avro",
    "format" = "avro"
);
```

Запрос данных из файла Avro:

```
SELECT * FROM FILES("path" = "hdfs://xxx.xx.xx.x:yyyy/avro/complex.avro",
"format" = "avro")\G
*************************** 1. row ***************************
record_field: {"id":1,"name":"avro"}
  enum_field: HEARTS
 array_field: ["one","two","three"]
   map_field: {"a":1,"b":2}
 union_field: 100
 fixed_field: 0x6162616261626162616261626162616261626162
1 row in set (0.05 sec)
```

Просмотр схемы файла Avro:

```
DESC FILES("path" = "hdfs://xxx.xx.xx.x:yyyy/avro/logical.avro", "format" =
"avro");
+------------------------+------------------+------+
| Field                  | Type             | Null |
+------------------------+------------------+------+
| decimal_bytes          | decimal(10,2)    | YES  |
| decimal_fixed          | decimal(10,2)    | YES  |
| uuid_string            | varchar(1048576) | YES  |
| date                   | date             | YES  |
| time_millis            | int              | YES  |
| time_micros            | bigint           | YES  |
| timestamp_millis       | datetime         | YES  |
| timestamp_micros       | datetime         | YES  |
| local_timestamp_millis | bigint           | YES  |
| local_timestamp_micros | bigint           | YES  |
| duration               | varbinary(12)    | YES  |
+------------------------+------------------+------+
```

Пример 9: Доступ к Azure Blob Storage с использованием Managed Identity и Service Principal

```
-- Managed Identity
SELECT * FROM FILES(
    "path" = "wasbs://storage-container@storage-
account.blob.core.windows.net/ssb_1g/customer/*",
    "format" = "parquet",
    "azure.blob.oauth2_use_managed_identity" = "true",
    "azure.blob.oauth2_client_id" = "1d6bfdec-dd34-4260-b8fd-aaaaaaaaaaaa"
);
-- Service Principal
SELECT * FROM FILES(
    "path" = "wasbs://storage-container@storage-
account.blob.core.windows.net/ssb_1g/customer/*",
    "format" = "parquet",
    "azure.blob.oauth2_client_id" = "1d6bfdec-dd34-4260-b8fd-bbbbbbbbbbbb",
    "azure.blob.oauth2_client_secret" =
"C2M8Q~ZXXXXXX_5XsbDCeL2dqP7hIR60xxxxxxxx",
    "azure.blob.oauth2_tenant_id" = "540e19cc-386b-4a44-a7b8-cccccccccccc"
);
```

Пример 10: Файл CSV

Запрос данных из файла CSV:

```
SELECT * FROM FILES(
"path" = "s3://test-bucket/file1.csv",
    "format" = "csv",
    "csv.column_separator"=",",
    "csv.row_delimiter"="\r\n",
    "csv.enclose"='"',
    "csv.skip_header"="1",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
    "aws.s3.region" = "us-west-2"
);
+------+---------+--------------+
| $1   | $2      | $3           |
+------+---------+--------------+
|    1 | 0.71173 | 2017-11-20   |
|    2 | 0.16145 | 2017-11-21   |
|    3 | 0.80524 | 2017-11-22   |
|    4 | 0.91852 | 2017-11-23   |
|    5 | 0.37766 | 2017-11-24   |
|    6 | 0.34413 | 2017-11-25   |
|    7 | 0.40055 | 2017-11-26   |
|    8 | 0.42437 | 2017-11-27   |
|    9 | 0.67935 | 2017-11-27   |
|   10 | 0.22783 | 2017-11-29   |
+------+---------+--------------+
10 rows in set (0.33 sec)
```

Загрузка файла CSV:

```
INSERT INTO csv_tbl
  SELECT * FROM FILES(
    "path" = "s3://test-bucket/file1.csv",
    "format" = "csv",
    "csv.column_separator"=",",
    "csv.row_delimiter"="\r\n",
    "csv.enclose"='"',
    "csv.skip_header"="1",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
    "aws.s3.region" = "us-west-2"
);
```

Пример 11: Использование региональных конечных точек AWS STS

Здесь показаны два случая:

1. Использование региональной конечной точки STS вне среды AWS.
2. Использование STS внутри среды AWS (например, EC2).

Вне среды AWS

> **IMPORTANT**
>
> При работе вне среды AWS и использовании региональной STS требуется установить `"aws.s3.use_instance_profile" = "false"`.

```
SELECT COUNT(*)
FROM FILES("path" = "s3://aws-bucket/path/file.csv.gz",
    "format" = "csv",
    "compression" = "gzip",
    "aws.s3.endpoint"="https://s3.us-east-1.amazonaws.com",
    "aws.s3.region"="us-east-1",
    "aws.s3.use_aws_sdk_default_behavior" = "false",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.access_key" = "****",
    "aws.s3.secret_key" = "****",
    "aws.s3.iam_role_arn"="arn:aws:iam::1234567890:role/access-role",
    "aws.s3.sts.region" = "{sts_region}",
    "aws.s3.sts.endpoint" = "{sts_endpoint}"
);
```

Внутри среды AWS

```
SELECT COUNT(*)
FROM FILES("path" = "s3://aws-bucket/path/file.csv.gz",
    "format" = "csv",
    "compression" = "gzip",
    "aws.s3.endpoint"="https://s3.us-east-1.amazonaws.com",
    "aws.s3.region"="us-east-1",
    "aws.s3.use_aws_sdk_default_behavior" = "false",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.access_key" = "****",
    "aws.s3.secret_key" = "****",
    "aws.s3.iam_role_arn"="arn:aws:iam::1234567890:role/access-role",
    "aws.s3.sts.region" = "{sts_region}",
    "aws.s3.sts.endpoint" = "{sts_endpoint}"
);
```

Generate_series

Генерирует последовательность значений в интервале, заданном параметрами `start` и `end`, с опциональным шагом `step`.

generate_series() — это табличная функция. Табличная функция может возвращать набор строк для каждой входной строки. Набор строк может содержать ноль, одну или несколько строк. Каждая строка может содержать один или несколько столбцов.

Для использования generate_series() в Selena необходимо заключить её в ключевое слово TABLE, если входные параметры являются константами. Если входные параметры являются выражениями, такими как имена столбцов, ключевое слово TABLE не требуется (см. Пример 5).

Эта функция поддерживается начиная с версии 3.1.

Синтаксис

```
generate_series(start, end [,step])
```

Параметры

- `start`: начальное значение последовательности, обязательный параметр. Поддерживаемые типы данных: INT, BIGINT и LARGEINT.

- `end`: конечное значение последовательности, обязательный параметр. Поддерживаемые типы данных: INT, BIGINT и LARGEINT.
- `step`: значение приращения или уменьшения, необязательный параметр. Поддерживаемые типы данных: INT, BIGINT и LARGEINT. Если не указано, шаг по умолчанию равен 1. `step` может быть отрицательным или положительным, но не может быть равен нулю.

Все три параметра должны иметь одинаковый тип данных, например, `generate_series(INT start, INT end [, INT step])`.

Начиная с версии 3.3 поддерживаются именованные аргументы, где все параметры вводятся с именами в формате `name=>expr`, например, `generate_series(start=>3, end=>7, step=>2)`. Именованные аргументы могут менять порядок аргументов и опционально устанавливать аргументы по умолчанию, но не могут смешиваться с позиционными аргументами.

Возвращаемое значение

Возвращает последовательность значений того же типа, что и входные параметры `start` и `end`.

- Когда `step` положительный, возвращается ноль строк, если `start` больше `end`. И наоборот, когда `step` отрицательный, возвращается ноль строк, если `start` меньше `end`.
- Если `step` равен 0, возвращается ошибка.
- Обработка NULL: если любой входной параметр является литеральным null, возвращается ошибка. Если любой входной параметр является выражением и результат выражения равен null, возвращается 0 строк (см. Пример 5).

Примеры

Пример 1: Генерация последовательности значений в диапазоне [2,5] в порядке возрастания с шагом по умолчанию `1`.

```
MySQL > select * from TABLE(generate_series(2, 5));
+-----------------+
| generate_series |
+-----------------+
|               2 |
|               3 |
|               4 |
|               5 |
+-----------------+
```

Пример 2: Генерация последовательности значений в диапазоне [2,5] в порядке возрастания с указанным шагом `2`.

```
MySQL > select * from TABLE(generate_series(2, 5, 2));
+-----------------+
| generate_series |
+-----------------+
|               2 |
|               4 |
+-----------------+
```

Пример 3: Генерация последовательности значений в диапазоне [5,2] в порядке убывания с указанным шагом `-1`.

```
MySQL > select * from TABLE(generate_series(5, 2, -1));
+-----------------+
| generate_series |
+-----------------+
|               5 |
|               4 |
|               3 |
|               2 |
+-----------------+
```

Пример 4: Возвращается ноль строк, когда `step` отрицательный и `start` меньше `end`.

```
MySQL > select * from TABLE(generate_series(2, 5, -1));
Empty set (0.01 sec)
```

Пример 5: Использование столбцов таблицы в качестве входных параметров generate_series(). В этом случае не нужно использовать `TABLE()` с generate_series().

```
CREATE TABLE t_numbers(start INT, end INT)
DUPLICATE KEY (start)
DISTRIBUTED BY HASH(start) BUCKETS 1;

INSERT INTO t_numbers VALUES
(1, 3),
(5, 2),
(NULL, 10),
(4, 7),
(9,6);

SELECT * FROM t_numbers;
+-------+------+
| start | end  |
+-------+------+
|  NULL |   10 |
|     1 |    3 |
|     4 |    7 |
|     5 |    2 |
|     9 |    6 |
+-------+------+

-- Генерация нескольких строк для строк (1,3) и (4,7) с шагом 1.
SELECT * FROM t_numbers, generate_series(t_numbers.start, t_numbers.end);
+-------+------+-----------------+
| start | end  | generate_series |
+-------+------+-----------------+
|     1 |    3 |               1 |
|     1 |    3 |               2 |
|     1 |    3 |               3 |
|     4 |    7 |               4 |
|     4 |    7 |               5 |
|     4 |    7 |               6 |
|     4 |    7 |               7 |
+-------+------+-----------------+

-- Генерация нескольких строк для строк (5,2) и (9,6) с шагом -1.
SELECT * FROM t_numbers, generate_series(t_numbers.start, t_numbers.end,
-1);
+-------+------+-----------------+
| start | end  | generate_series |
+-------+------+-----------------+
|     5 |    2 |               5 |
|     5 |    2 |               4 |
|     5 |    2 |               3 |
```

```
|      5 |     2 |                2 |
|      9 |     6 |                9 |
|      9 |     6 |                8 |
|      9 |     6 |                7 |
|      9 |     6 |                6 |
+--------+-------+------------------+
```

Пример 6: Генерация последовательности значений в диапазоне [2,5] в порядке возрастания с указанным шагом `2`, используя именованные аргументы.

```
MySQL > select * from TABLE(generate_series(start=>2, end=>5, step=>2));
+------------------+
| generate_series |
+------------------+
|                2 |
|                4 |
+------------------+
```

Ключевые слова

table function, generate series

2.27 Utility

Bar

Рисует столбчатый график в виде гистограммы для проверки распределения данных.

Синтаксис

```
bar(size, min, max, width)
```

Параметры

- `size`: размер столбца, должен находиться в диапазоне `[min, max]`
- `min`: минимальное значение столбца
- `max`: максимальное значение столбца
- `width`: ширина столбца

Пример

```
MYSQL > select r, bar(r, 0, 10, 20) as x from table(generate_series(0, 10))
as s(r);

0
1       ▓
2       ▓▓
3       ▓▓▓
4       ▓▓▓▓
5       ▓▓▓▓▓
6       ▓▓▓▓▓▓
7       ▓▓▓▓▓▓▓
8       ▓▓▓▓▓▓▓▓
9       ▓▓▓▓▓▓▓▓▓
10      ▓▓▓▓▓▓▓▓▓▓
```

Catalog

Возвращает имя текущего каталога. Каталог может быть внутренним каталогом Selena или внешним каталогом, сопоставленным с внешним источником данных. Подробнее о каталогах см. в разделе Обзор каталогов.

Если каталог не выбран, возвращается внутренний каталог Selena `default_catalog`.

Синтаксис

```
catalog()
```

Параметры

Эта функция не требует параметров.

Возвращаемое значение

Возвращает имя текущего каталога в виде строки.

Примеры

Пример 1: Текущий каталог — внутренний каталог Selena `default_catalog`.

```
select catalog();
+-----------------+
| CATALOG()       |
+-----------------+
| default_catalog |
+-----------------+
1 row in set (0.01 sec)
```

Пример 2: Текущий каталог — внешний каталог `hudi_catalog`.

```
-- Переключиться на внешний каталог.
set catalog hudi_catalog;

-- Вернуть имя текущего каталога.
select catalog();
+--------------+
| CATALOG()    |
+--------------+
| hudi_catalog |
+--------------+
```

См. также

SET CATALOG: Переключение на целевой каталог.

Column_size и column_compressed_size

Эти функции возвращают информацию о размере столбцов таблицы для анализа и оптимизации хранения. Обе функции работают с подсказкой `[_META_]` для проверки метаданных файлов сегментов.

Column_size

Возвращает распакованный размер столбца в байтах.

Синтаксис

```
-- Не опускайте скобки [] в подсказке.
SELECT column_size(column_name) FROM table_name [_META_];
```

Параметры

- `column_name`: имя столбца, для которого требуется получить распакованный размер.

Возвращаемое значение

Возвращает распакованный размер столбца в байтах как значение типа BIGINT.

Column_compressed_size

Возвращает сжатый размер столбца в байтах.

Синтаксис

```
-- Не опускайте скобки [] в подсказке.
SELECT column_compressed_size(column_name) FROM table_name [_META_];
```

Параметры

- `column_name`: имя столбца, для которого требуется получить сжатый размер.

Возвращаемое значение

Возвращает сжатый размер столбца в байтах как значение типа BIGINT.

Примечания по использованию

- Обе функции должны использоваться с подсказкой `[_META_]` для доступа к информации о метаданных.
- Функции сканируют метаданные базовых файлов сегментов с помощью оператора META_SCAN.
- Для сложных типов данных (JSON, ARRAY, MAP, STRUCT) функции рекурсивно вычисляют размер всех подстолбцов.
- `column_size` возвращает несжатый размер столбца.
- `column_compressed_size` возвращает сжатый размер на диске.

Примеры

```
-- Получить распакованный и сжатый размеры столбцов
SELECT
    column_size(name) as name_decompressed_size,
    column_compressed_size(name) as name_compressed_size,
    column_size(description) as desc_decompressed_size,
    column_compressed_size(description) as desc_compressed_size
FROM products [_META_];
```

Current_role

Запрашивает роли, активированные для текущего пользователя.

Синтаксис

```
current_role();
current_role;
```

Параметры

Нет.

Возвращаемое значение

Возвращает значение типа VARCHAR.

Примеры

```
mysql> select current_role();
+----------------+
| current_role() |
+----------------+
| db_admin       |
+----------------+
```

Current_version

Возвращает текущую версию Selena. Предоставляются два синтаксиса для совместимости с различными клиентами.

Синтаксис

```
current_version();

@@version_comment;
```

Параметры

Нет

Возвращаемое значение

Возвращает значение типа VARCHAR.

Примеры

```
mysql> select current_version();
+-------------------+
| current_version() |
+-------------------+
| 2.1.2 0782ad7     |
+-------------------+
1 row in set (0.00 sec)

mysql> select @@version_comment;
+-------------------------+
| @@version_comment       |
+-------------------------+
| Selena version 2.1.2 |
+-------------------------+
1 row in set (0.01 sec)
```

CURRENT_WAREHOUSE

Возвращает имя активного warehouse для текущей сессии. Результат отражает warehouse, установленный через `SET WAREHOUSE`, или warehouse по умолчанию, если он не был явно выбран.

Синтаксис

```
current_warehouse()
```

Возвращаемое значение

`VARCHAR`. Имя warehouse, используемого в текущей сессии.

Примеры

```
mysql> SET WAREHOUSE = 'compute_pool_x';
mysql> SELECT current_warehouse();
+---------------------+
| CURRENT_WAREHOUSE() |
+---------------------+
| compute_pool_x      |
+---------------------+
1 row in set (0.00 sec)
```

Database

Возвращает имя текущей базы данных. Если база данных не выбрана, возвращается пустое значение.

Синтаксис

```
database()
```

Параметры

Эта функция не требует параметров.

Возвращаемое значение

Возвращает имя текущей базы данных в виде строки.

Примеры

```
-- Выбрать целевую базу данных.
use db_test

-- Запросить имя текущей базы данных.
select database();
+------------+
| DATABASE() |
+------------+
| db_test    |
+------------+
```

См. также

USE: Переключение на целевую базу данных.

Encode_sort_key

Создает составной бинарный ключ с сохранением порядка из нескольких разнородных входных столбцов. Эта функция необходима для создания эффективных ключей сортировки, которые сохраняют исходный порядок сортировки при лексикографическом сравнении.

Синтаксис

```
encode_sort_key(column1, column2, ..., columnN)
```

Параметры

- `column1, column2, ..., columnN`: один или несколько столбцов любого поддерживаемого типа данных. Функция принимает переменное количество аргументов.

Возвращаемое значение

Возвращает значение типа VARBINARY, представляющее составной ключ сортировки.

Поддерживаемые типы данных

Поддерживаются следующие типы данных:

Тип данных	Описание
`TINYINT`	8-битное знаковое целое
`SMALLINT`	16-битное знаковое целое
`INT`	32-битное знаковое целое
`BIGINT`	64-битное знаковое целое
`LARGEINT`	128-битное знаковое целое
`FLOAT`	32-битное число с плавающей точкой
`DOUBLE`	64-битное число с плавающей точкой
`VARCHAR`	Строка переменной длины
`CHAR`	Строка фиксированной длины
`DATE`	Значение даты
`DATETIME`	Значение даты и времени
`TIMESTAMP`	Значение временной метки

Следующие сложные типы не поддерживаются и вернут ошибку:

- `JSON`
- `ARRAY`
- `MAP`
- `STRUCT`
- `HLL`
- `BITMAP`
- `PERCENTILE`

Стратегия кодирования

Функция использует различные стратегии кодирования для разных типов данных, чтобы гарантировать, что лексикографическое сравнение результирующих бинарных ключей сохраняет исходный порядок сортировки:

- **Целочисленные типы**: порядок байтов big-endian с инвертированным битом знака для знаковых типов
- **Типы с плавающей точкой**: специальное кодирование для обеспечения правильного порядка сортировки
- **Строковые типы**: специальное кодирование с экранированием байта 0x00 и терминатором 0x00 0x00 для непоследних полей
- **Типы даты/времени**: внутреннее целочисленное представление, закодированное как целочисленное значение

Обработка NULL

- Каждый столбец получает маркер NULL (0x00 для NULL, 0x01 для NOT NULL) для каждой строки
- Маркеры NULL добавляются даже для ненулевых столбцов для обеспечения согласованного кодирования
- Байт-разделитель (0x00) используется между столбцами (кроме последнего столбца)

Примеры

Сгенерированный столбец ключа сортировки

```
CREATE TABLE user_analytics (
    user_id INT,
    region VARCHAR(50),
    score DOUBLE,
    created_date DATE,
    sort_key VARBINARY(1024) AS (
        encode_sort_key(region, score, created_date)
    )
) ORDER BY (sort_key);
```

Извлечение полей JSON

```
CREATE TABLE json_data (
    id INT,
    json_content JSON,
    sort_key VARBINARY(1024) AS (
        encode_sort_key(
            get_json_int(json_content, '$.priority'),
            get_json_string(json_content, '$.category'),
            get_json_double(json_content, '$.score')
        )
    )
) ORDER BY (sort_key);
```

Ограничения

Ограничения типов данных

Сложные типы, такие как JSON, ARRAY, MAP и STRUCT, не могут быть закодированы напрямую. Используйте функции извлечения JSON для получения примитивных значений:

```
-- Вместо: encode_sort_key(json_col)
-- Используйте: encode_sort_key(get_json_int(json_col, '$.field1'),
get_json_string(json_col, '$.field2'))
```

Соображения производительности

- Каждый вызов `encode_sort_key` требует кодирования всех входных столбцов
- Бинарные ключи могут быть значительно больше исходных данных
- Используйте сгенерированные столбцы, чтобы избежать повторного кодирования

Ограничения размера ключа

- Количество столбцов должно быть разумным (обычно < 10)
- Предпочитайте более короткие строковые столбцы, когда это возможно
- Рассмотрите использование хеш-функций для очень длинных строк

Equiwidth_bucket

Вычисляет bucket равноширинной гистограммы.

Синтаксис

```
equiwidth_bucket(value, min, max, buckets)
```

Параметры

- `value`: значение строки, должно находиться в диапазоне `[min, max]`
- `min`: минимальное значение гистограммы
- `max`: максимальное значение гистограммы
- `buckets`: количество bucket'ов гистограммы

Пример

```
MYSQL > select r, equiwidth_bucket(r, 0, 10, 20) as x from
table(generate_series(0, 10)) as s(r);

0       0
1       1
2       2
3       3
4       4
5       5
6       6
7       7
8       8
9       9
10      10
```

Get_query_profile

Получает профиль запроса по его `query_id`. Эта функция возвращает пустой результат, если `query_id` не существует или указан неверно.

Для использования этой функции необходимо включить функцию профилирования, то есть установить сессионную переменную `enable_profile` в `true` (`set enable_profile = true;`). Если эта функция не включена, возвращается пустой профиль.

Эта функция поддерживается начиная с версии 3.0.

Синтаксис

```
get_query_profile(x)
```

Параметры

`x`: строка query_id. Поддерживаемый тип данных: VARCHAR.

Возвращаемое значение

Профиль запроса содержит следующие поля. Подробнее о профиле запроса см. в разделе Профиль запроса.

```
Query:
  Summary:
  Planner:
  Execution Profile 7de16a85-761c-11ed-917d-00163e14d435:
    Fragment 0:
      Pipeline (id=2):
        EXCHANGE_SINK (plan_node_id=18):
        LOCAL_MERGE_SOURCE (plan_node_id=17):
      Pipeline (id=1):
        LOCAL_SORT_SINK (plan_node_id=17):
        AGGREGATE_BLOCKING_SOURCE (plan_node_id=16):
      Pipeline (id=0):
        AGGREGATE_BLOCKING_SINK (plan_node_id=16):
        EXCHANGE_SOURCE (plan_node_id=15):
    Fragment 1:
       ...
    Fragment 2:
       ...
```

Примеры

```
-- Включить функцию профилирования.
set enable_profile = true;

-- Выполнить простой запрос.
select 1;

-- Получить query_id запроса.
select last_query_id();
+--------------------------------------+
| last_query_id()                      |
+--------------------------------------+
| bd3335ce-8dde-11ee-92e4-3269eb8da7d1 |
+--------------------------------------+

-- Получить профиль запроса.
select get_query_profile('502f3c04-8f5c-11ee-a41f-b22a2c00f66b');

-- Использовать функцию regexp_extract для получения QueryPeakMemoryUsage
из профиля, соответствующего указанному шаблону.
select regexp_extract(get_query_profile('bd3335ce-8dde-11ee-92e4-
3269eb8da7d1'), 'QueryPeakMemoryUsage: [0-9\.]* [KMGB]*', 0);
+-----------------------------------------------------------------------------
---------------------------------------------+
| regexp_extract(get_query_profile('bd3335ce-8dde-11ee-92e4-3269eb8da7d1'),
'QueryPeakMemoryUsage: [0-9.]* [KMGB]*', 0) |
+-----------------------------------------------------------------------------
---------------------------------------------+
| QueryPeakMemoryUsage: 3.828 KB
|
+-----------------------------------------------------------------------------
---------------------------------------------+
```

Связанные функции

- last_query_id
- regexp_extract

Host_name

Получает имя хоста узла, на котором выполняются вычисления.

Синтаксис

```
host_name();
```

Параметры

Нет

Возвращаемое значение

Возвращает значение типа VARCHAR.

Примеры

```
select host_name();
+-------------+
| host_name() |
+-------------+
| sandbox-sql |
+-------------+
1 row in set (0.01 sec)
```

Is_role_in_session

Проверяет, активна ли роль (или вложенная роль) в текущей сессии.

Эта функция поддерживается начиная с версии 3.1.4.

Синтаксис

```
BOOLEAN is_role_in_session(VARCHAR role_name);
```

Параметры

`role_name`: роль, которую вы хотите проверить (может быть также вложенной ролью). Поддерживаемый тип данных: VARCHAR.

Возвращаемое значение

Возвращает значение типа BOOLEAN. `1` означает, что роль активна в текущей сессии. `0` означает противоположное.

Примеры

1. Создание ролей и пользователя.

```
-- Создать три роли.
create role r1;
create role r2;
create role r3;

-- Создать пользователя u1.
create user u1;

-- Передать роли r2 и r3 роли r1 и назначить r1 пользователю u1. Таким
образом, пользователь u1 имеет три роли: r1, r2 и r3.
grant r3 to role r2;
grant r2 to role r1;
grant r1 to user u1;

-- Переключиться на пользователя u1 и выполнять операции от его имени.
execute as u1 with no revert;
```

2. Проверить, активна ли роль `r1`. Результат показывает, что эта роль не активна.

```
select is_role_in_session("r1");
+--------------------------+
| is_role_in_session('r1') |
+--------------------------+
|                        0 |
+--------------------------+
```

3. Выполнить команду SET ROLE для активации `r1` и использовать `is_role_in_session` для проверки, активна ли роль. Результат показывает, что `r1` активна, а роли `r2` и `r3`, вложенные в `r1`, также активны.

```
set role "r1";

select is_role_in_session("r1");
+--------------------------+
| is_role_in_session('r1') |
+--------------------------+
|                        1 |
+--------------------------+

select is_role_in_session("r2");
+--------------------------+
| is_role_in_session('r2') |
+--------------------------+
|                        1 |
+--------------------------+

select is_role_in_session("r3");
+--------------------------+
| is_role_in_session('r3') |
+--------------------------+
|                        1 |
+--------------------------+
```

Isnotnull

Проверяет, не является ли значение `NULL`, возвращает `1`, если оно не `NULL`, и возвращает `0`, если оно `NULL`.

Синтаксис

```
ISNOTNULL(v)
```

Параметры

- `v`: значение для проверки. Поддерживаются все типы данных.

Возвращаемое значение

Возвращает 1, если значение не `NULL`, и возвращает 0, если значение `NULL`.

Примеры

```
MYSQL > SELECT c1, isnotnull(c1) FROM t1;
+------+--------------+
| c1   | `c1` IS NULL |
+------+--------------+
| NULL |            0 |
|    1 |            1 |
+------+--------------+
```

Isnull

Проверяет, является ли значение `NULL`, возвращает `1`, если оно `NULL`, и возвращает `0`, если оно не `NULL`.

Синтаксис

```
ISNULL(v)
```

Параметры

- `v`: значение для проверки. Поддерживаются все типы данных.

Возвращаемое значение

Возвращает 1, если значение `NULL`, и возвращает 0, если значение не `NULL`.

Пример

```
MYSQL > SELECT c1, isnull(c1) FROM t1;
+------+--------------+
| c1   | `c1` IS NULL |
+------+--------------+
| NULL |            1 |
|    1 |            0 |
+------+--------------+
```

Last_query_id

Получает идентификатор последнего выполненного запроса в текущей сессии.

Синтаксис

```
VARCHAR last_query_id();
```

Параметры

Нет

Возвращаемое значение

Возвращает значение типа VARCHAR.

Примеры

```
mysql> select last_query_id();
+--------------------------------------+
| last_query_id()                      |
+--------------------------------------+
| 7c1d8d68-bbec-11ec-af65-00163e1e238f |
+--------------------------------------+
1 row in set (0.00 sec)
```

Ключевые слова

LAST_QUERY_ID

Sleep

Задерживает выполнение операции на указанный период времени (в секундах) и возвращает логическое значение, указывающее, завершилась ли задержка без прерывания. Возвращает `1`, если задержка завершилась без прерывания. В противном случае возвращает `0`.

Синтаксис

```
BOOLEAN sleep(INT x);
```

Параметры

`x`: продолжительность задержки выполнения операции. Должен быть типа INT. Единица измерения: секунды. Если входное значение равно NULL, немедленно возвращается NULL без ожидания.

Возвращаемое значение

Возвращает значение типа BOOLEAN.

Примеры

```
select sleep(3);
+----------+
| sleep(3) |
+----------+
|        1 |
+----------+
1 row in set (3.00 sec)

select sleep(NULL);
+-------------+
| sleep(NULL) |
+-------------+
|        NULL |
+-------------+
1 row in set (0.00 sec)
```

Ключевые слова

SLEEP, sleep

Uuid

Возвращает случайный UUID типа VARCHAR. Два вызова этой функции могут сгенерировать два разных числа. UUID имеет длину 36 символов. Он содержит 5 шестнадцатеричных чисел, соединенных четырьмя дефисами в формате aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee.

Синтаксис

```
uuid();
```

Параметры

Нет

Возвращаемое значение

Возвращает значение типа VARCHAR.

Примеры

```
mysql> select uuid();
+--------------------------------------+
| uuid()                               |
+--------------------------------------+
| 74a2ed19-9d21-4a99-a67b-aa5545f26454 |
+--------------------------------------+
1 row in set (0.01 sec)
```

Uuid_numeric

Возвращает случайный UUID типа LARGEINT. Эта функция имеет производительность выполнения на 2 порядка лучше, чем функция `uuid`.

Синтаксис

```
uuid_numeric();
```

Параметры

Нет

Возвращаемое значение

Возвращает значение типа LARGEINT.

Примеры

```
MySQL > select uuid_numeric();
+--------------------------+
| uuid_numeric()           |
+--------------------------+
| 5587124452863678986612 05 |
+--------------------------+
1 row in set (0.00 sec)
```

Version

Возвращает текущую версию базы данных MySQL.

Вы можете использовать current_version для запроса версии Selena.

Синтаксис

```
VARCHAR version();
```

Параметры

Нет

Возвращаемое значение

Возвращает значение типа VARCHAR.

Примеры

```
mysql> select version();
+-----------+
| version() |
+-----------+
| 5.1.0     |
+-----------+
1 row in set (0.00 sec)
```

Ссылки

current_version

3 SQL statements

3.1 User account management

ALTER USER

ALTER USER изменяет информацию о пользователе, включая пароль, метод аутентификации, роли по умолчанию и свойства пользователя (поддерживается начиная с версии v1.5.2).

> **ПОДСКАЗКА**
> Обычные пользователи могут использовать эту команду для изменения информации о себе. Только пользователи с ролью `user_admin` могут изменять информацию о других пользователях.

Синтаксис

```
ALTER USER user_identity
[auth_option]
[default_role]
[DEFAULT ROLE <role_name>[, <role_name>, ...]]
[SET PROPERTIES ("key"="value", ...)]
```

Параметры

- `user_identity` состоит из двух частей: "user_name" и "host", в формате `username@'userhost'`. Для части "host" вы можете использовать `%` для нечёткого соответствия. Если "host" не указан, по умолчанию используется "%", что означает, что пользователь может подключаться к Selena с любого хоста. Однако при использовании этого выражения с `SET PROPERTIES` для изменения свойств пользователя необходимо указать `username` вместо `user_identity`.

- `auth_option` определяет метод аутентификации. В настоящее время поддерживаются три метода аутентификации: нативный пароль Selena, mysql_native_password и "authentication_ldap_simple". Нативный пароль Selena логически идентичен mysql_native_password, но немного отличается синтаксисом. Одна идентификация пользователя может использовать только один метод

аутентификации. Вы можете использовать ALTER USER для изменения паролей и методов аутентификации пользователей.

```
auth_option: {
    IDENTIFIED BY 'auth_string'
    IDENTIFIED WITH mysql_native_password BY 'auth_string'
    IDENTIFIED WITH mysql_native_password AS 'auth_string'
    IDENTIFIED WITH authentication_ldap_simple AS 'auth_string'

}
```

Метод аутентификации	Пароль при создании пользователя	Пароль для входа
Нативный пароль	Открытый текст или шифротекст	Открытый текст
`mysql_native_password BY`	Открытый текст	Открытый текст
`mysql_native_password WITH`	Шифротекст	Открытый текст
`authentication_ldap_simple`	Открытый текст	Открытый текст

> Примечание: Selena шифрует пароли пользователей перед их сохранением.

- `DEFAULT ROLE` устанавливает роль по умолчанию для пользователя.

```
 -- Установить указанные роли в качестве ролей по умолчанию.
 DEFAULT ROLE <role_name>[, <role_name>, ...]
 -- Установить все роли пользователя, включая роли, которые будут
назначены этому пользователю, в качестве ролей по умолчанию.
 DEFAULT ROLE ALL
 -- Роль по умолчанию не устанавливается, но роль public остаётся
активной после входа пользователя.
 DEFAULT ROLE NONE
```

Перед запуском ALTER USER для установки ролей по умолчанию убедитесь, что все роли назначены пользователям. Роли автоматически активируются после повторного входа пользователя.

- `SET PROPERTIES` устанавливает свойства пользователя, включая максимальное количество подключений пользователя (`max_user_connections`), catalog, базу данных или сессионные переменные на уровне пользователя. Сессионные переменные уровня пользователя вступают в силу при входе пользователя. Эта функция поддерживается начиная с версии v1.5.2.

```
-- Установить максимальное количество подключений пользователя.
SET PROPERTIES ("max_user_connections" = "<Integer>")
-- Установить catalog.
SET PROPERTIES ("catalog" = "<catalog_name>")
-- Установить базу данных.
SET PROPERTIES ("catalog" = "<catalog_name>", "database" = "
<database_name>")
-- Установить сессионные переменные.
SET PROPERTIES ("session.<variable_name>" = "<value>", ...)
-- Очистить свойства, установленные для пользователя.
SET PROPERTIES ("catalog" = "", "database" = "", "session.
<variable_name>" = "");
```

ПОДСКАЗКА

- `SET PROPERTIES` работает с пользователем, а не с идентификацией пользователя. Поэтому при изменении свойств пользователя необходимо указать `username` вместо `user_identity` в выражении `ALTER USER`.
- Глобальные переменные и переменные только для чтения не могут быть установлены для конкретного пользователя.
- Переменные вступают в силу в следующем порядке: SET_VAR > Session > User property > Global.
- Вы можете использовать SHOW PROPERTY для просмотра свойств конкретного пользователя.

Примеры

Пример 1: Изменить пароль пользователя на пароль в открытом тексте.

```
ALTER USER 'jack' IDENTIFIED BY '123456';
ALTER USER jack@'172.10.1.10' IDENTIFIED WITH mysql_native_password BY
'123456';
```

Пример 2: Изменить пароль пользователя на зашифрованный пароль.

```
ALTER USER jack@'172.10.1.10' IDENTIFIED BY PASSWORD
'*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9';
ALTER USER jack@'172.10.1.10' IDENTIFIED WITH mysql_native_password AS
'*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9';
```

> Вы можете получить зашифрованный пароль с помощью функции password().

Пример 3: Изменить метод аутентификации на LDAP.

```
ALTER USER jack@'172.10.1.10' IDENTIFIED WITH authentication_ldap_simple;
```

Пример 4: Изменить метод аутентификации на LDAP и указать отличительное имя (DN) пользователя в LDAP.

```
ALTER USER jack@'172.10.1.10' IDENTIFIED WITH authentication_ldap_simple AS
'uid=jack,ou=company,dc=example,dc=com';
```

Пример 5: Изменить роли по умолчанию пользователя на `db_admin` и `user_admin`. Обратите внимание, что пользователю должны быть назначены эти две роли.

```
ALTER USER 'jack'@'192.168.%' DEFAULT ROLE db_admin, user_admin;
```

Пример 6: Установить все роли пользователя, включая роли, которые будут назначены этому пользователю, в качестве ролей по умолчанию.

```
ALTER USER 'jack'@'192.168.%' DEFAULT ROLE ALL;
```

Пример 7: Очистить все роли по умолчанию пользователя.

```
ALTER USER 'jack'@'192.168.%' DEFAULT ROLE NONE;
```

> Примечание: По умолчанию роль `public` остаётся активной для пользователя.

Пример 8: Установить максимальное количество подключений пользователя на `600`.

```
ALTER USER 'jack' SET PROPERTIES ("max_user_connections" = "600");
```

Пример 9: Установить catalog пользователя на `hive_catalog`.

```
ALTER USER 'jack' SET PROPERTIES ('catalog' = 'hive_catalog');
```

Пример 10: Установить базу данных пользователя на `test_db` в catalog по умолчанию.

```
ALTER USER 'jack' SET PROPERTIES ('catalog' = 'default_catalog', 'database'
= 'test_db');
```

Пример 11: Установить сессионную переменную `query_timeout` на `600` для пользователя.

```
ALTER USER 'jack' SET PROPERTIES ('session.query_timeout' = '600');
```

Пример 12: Очистить свойства, установленные для пользователя.

```
ALTER USER 'jack' SET PROPERTIES ('catalog' = '', 'database' = '',
'session.query_timeout' = '');
```

Ссылки

- CREATE USER
- GRANT
- SHOW USERS
- DROP USER

CREATE ROLE

CREATE ROLE создаёт роль. После создания роли вы можете предоставить привилегии роли, а затем назначить эту роль пользователю или другой роли. Таким образом, привилегии, связанные с этой ролью, передаются пользователям или ролям.

ПОДСКАЗКА

Только пользователи с ролью `user_admin` имеют привилегии для выполнения этой операции.

Синтаксис

```
CREATE ROLE <role_name>
```

Параметры

`role_name`: имя роли. Соглашения об именовании см. в Системные ограничения.

Обратите внимание, что имя создаваемой роли не может совпадать с системными ролями: `root`, `cluster_admin`, `db_admin`, `user_admin` и `public`.

Ограничения

По умолчанию пользователь может иметь максимум 64 роли. Вы можете изменить эту настройку с помощью динамического параметра FE `privilege_max_total_roles_per_user`. Роль может иметь максимум 16 уровней наследования. Вы можете изменить эту настройку с помощью динамического параметра FE `privilege_max_role_depth`.

Примеры

Создать роль.

```
CREATE ROLE role1;
```

Ссылки

- GRANT
- SHOW ROLES
- DROP ROLE

CREATE USER

CREATE USER создает пользователя Selena. В Selena "user_identity" уникально идентифицирует пользователя. Начиная с версии 3.3.3, Selena поддерживает установку свойств пользователя при его создании.

> **ПОДСКАЗКА**
>
> Только пользователи с ролью `user_admin` имеют привилегии для выполнения этой операции.

Синтаксис

```
CREATE USER [IF NOT EXISTS] <user_identity>
[auth_option]
[DEFAULT ROLE <role_name>[, <role_name>, ...]]
[PROPERTIES ("key"="value", ...)]
```

Параметры

- `user_identity` состоит из двух частей, "user_name" и "host", в формате `username@'userhost'`. В части "host" можно использовать `%` для нечеткого совпадения. Если "host" не указан, по умолчанию используется "%", что означает, что пользователь может подключаться к Selena с любого хоста.

  Правила именования пользователей см. в разделе Системные ограничения.

- `auth_option` указывает метод аутентификации. В настоящее время поддерживается пять методов аутентификации: встроенный пароль Selena, `mysql_native_password`, `authentication_ldap_simple`, аутентификация JSON Web Token (JWT) и аутентификация OAuth 2.0. Встроенный пароль Selena логически совпадает с `mysql_native_password`, но немного отличается синтаксисом. Один идентификатор пользователя может использовать только один метод аутентификации.

```
auth_option: {
    IDENTIFIED BY 'auth_string'
    IDENTIFIED WITH mysql_native_password BY 'auth_string'
    IDENTIFIED WITH mysql_native_password AS 'auth_string'
    IDENTIFIED WITH authentication_ldap_simple AS 'auth_string'
    IDENTIFIED WITH authentication_jwt [AS 'auth_properties']
    IDENTIFIED WITH authentication_oauth2 [AS 'auth_properties']
}
```

Метод аутентификации	Пароль при создании пользователя	Пароль при входе
Native password	Открытый текст или шифротекст	Открытый текст
`mysql_native_password BY`	Открытый текст	Открытый текст
`mysql_native_password WITH`	Шифротекст	Открытый текст
`authentication_ldap_simple`	Открытый текст	Открытый текст

> **ПРИМЕЧАНИЕ**
>
> Selena шифрует пароли пользователей перед их сохранением.

Подробности о `auth_properties` для аутентификации JSON Web Token (JWT) и OAuth 2.0 см. в соответствующих документах:

- Аутентификация JSON Web Token
- Аутентификация OAuth 2.0

- `DEFAULT ROLE <role_name>[, <role_name>, ...]`: Если этот параметр указан, роли автоматически назначаются пользователю и активируются по умолчанию при входе пользователя в систему. Если не указан, у пользователя не будет никаких привилегий. Убедитесь, что все указанные роли уже существуют.

- `PROPERTIES` устанавливает свойства пользователя, включая максимальное количество соединений пользователя (`max_user_connections`), catalog, базу данных или переменные сеанса на уровне пользователя. Переменные сеанса на уровне пользователя вступают в силу при входе пользователя в систему. Эта функция поддерживается начиная с версии 3.3.3.

```
-- Установить максимальное количество соединений пользователя.
PROPERTIES ("max_user_connections" = "<Integer>")
-- Установить catalog.
PROPERTIES ("catalog" = "<catalog_name>")
-- Установить базу данных.
PROPERTIES ("catalog" = "<catalog_name>", "database" = "
<database_name>")
-- Установить переменные сеанса.
PROPERTIES ("session.<variable_name>" = "<value>", ...)
```

ПОДСКАЗКА

- `PROPERTIES` работает на уровне пользователя, а не идентификатора пользователя.
- Глобальные переменные и переменные только для чтения не могут быть установлены для конкретного пользователя.
- Переменные вступают в силу в следующем порядке: SET_VAR > Session > User property > Global.
- Вы можете использовать SHOW PROPERTY для просмотра свойств конкретного пользователя.

Примеры

Пример 1: Создание пользователя с паролем в открытом тексте без указания хоста, что эквивалентно `jack@'%'`.

```
CREATE USER 'jack' IDENTIFIED BY '123456';
```

Пример 2: Создание пользователя с паролем в открытом тексте и разрешением входа с адреса `'172.10.1.10'`.

```
CREATE USER jack@'172.10.1.10' IDENTIFIED WITH mysql_native_password BY
'123456';
```

Пример 3: Создание пользователя с зашифрованным паролем и разрешением входа с адреса `'172.10.1.10'`.

```
CREATE USER jack@'172.10.1.10' IDENTIFIED BY PASSWORD
'*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9';
CREATE USER jack@'172.10.1.10' IDENTIFIED WITH mysql_native_password AS
'*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9';
```

> Примечание: Вы можете получить зашифрованный пароль с помощью функции password().

Пример 4: Создание пользователя, которому разрешен вход с доменного имени 'example_domain'.

```
CREATE USER 'jack'@['example_domain'] IDENTIFIED BY '123456';
```

Пример 5: Создание пользователя, использующего аутентификацию LDAP.

```
CREATE USER jack@'172.10.1.10' IDENTIFIED WITH authentication_ldap_simple;
```

Пример 6: Создание пользователя, использующего аутентификацию LDAP, с указанием distinguished name (DN) пользователя в LDAP.

```
CREATE USER jack@'172.10.1.10' IDENTIFIED WITH authentication_ldap_simple
AS 'uid=jack,ou=company,dc=example,dc=com';
```

Пример 7: Создание пользователя, которому разрешен вход из подсети '192.168', с установкой `db_admin` и `user_admin` в качестве ролей по умолчанию.

```
CREATE USER 'jack'@'192.168.%' DEFAULT ROLE db_admin, user_admin;
```

Пример 8: Создание пользователя и установка максимального количества соединений `600`.

```
CREATE USER 'jack'@'192.168.%' PROPERTIES ("max_user_connections" = "600");
```

Пример 9: Создание пользователя и установка catalog пользователя в `hive_catalog`.

```
CREATE USER 'jack'@'192.168.%' PROPERTIES ('catalog' = 'hive_catalog');
```

Пример 10: Создание пользователя и установка базы данных пользователя в `test_db` в catalog по умолчанию.

```
CREATE USER 'jack'@'192.168.%' PROPERTIES ('catalog' = 'default_catalog',
'database' = 'test_db');
```

Пример 11: Создание пользователя и установка переменной сеанса `query_timeout` в `600` для пользователя.

```
CREATE USER 'jack'@'192.168.%' PROPERTIES ('session.query_timeout' =
'600');
```

Пример 12: Создание пользователя с аутентификацией JSON Web Token.

```
CREATE USER tom IDENTIFIED WITH authentication_jwt AS
'{
  "jwks_url": "http://localhost:38080/realms/master/protocol/jwt/certs",
  "principal_field": "preferred_username",
  "required_issuer": "http://localhost:38080/realms/master",
  "required_audience": "selena"
}';
```

Пример 13: Создание пользователя с аутентификацией OAuth 2.0.

```
CREATE USER tom IDENTIFIED WITH authentication_oauth2 AS
'{
  "auth_server_url": "http://localhost:38080/realms/master/protocol/openid-connect/auth",
  "token_server_url": "http://localhost:38080/realms/master/protocol/openid-connect/token",
  "client_id": "12345",
  "client_secret": "LsWyD9vPcM3LHxLZfzJsuoBwWQFBLcoR",
  "redirect_url": "http://localhost:8030/api/oauth2",
  "jwks_url": "http://localhost:38080/realms/master/protocol/openid-connect/certs",
  "principal_field": "preferred_username",
  "required_issuer": "http://localhost:38080/realms/master",
  "required_audience": "12345"
}';
```

DROP ROLE

DROP ROLE удаляет роль. Если роль была предоставлена пользователю, пользователь потеряет привилегии, связанные с этой ролью, после её удаления.

ПОДСКАЗКА

- Только пользователи с ролью `user_admin` могут удалять роли.
- Системные роли Selena не могут быть удалены.

Синтаксис

```
DROP ROLE <role_name>
```

Примеры

Удаление роли.

```
DROP ROLE role1;
```

DROP USER

DROP USER удаляет указанный идентификатор пользователя.

> **ПОДСКАЗКА**
>
> Только пользователи с ролью `user_admin` имеют привилегии для выполнения этой операции.

Синтаксис

```
 DROP USER '<user_identity>'

`user_identity`:

 user@'host'
user@['domain']
```

Примеры

Удаление пользователя `jack@'192.%'`.

```
DROP USER 'jack'@'192.%'
```

EXECUTE AS

Используйте привилегию IMPERSONATE с командами EXECUTE AS для переключения контекста выполнения текущей сессии на пользователя, которого вы имитируете.

Эта команда поддерживается начиная с версии v1.5.2.

Синтаксис

```
EXECUTE AS user WITH NO REVERT
```

Параметры

`user`: Пользователь должен уже существовать.

Примечания по использованию

- Текущий пользователь входа (который вызывает команду EXECUTE AS) должен иметь привилегию имитации другого пользователя. Для получения дополнительной

информации см. GRANT.

- Команда EXECUTE AS должна содержать предложение WITH NO REVERT, что означает, что контекст выполнения текущей сессии не может быть переключен обратно на исходного пользователя входа до завершения текущей сессии.

Примеры

Переключить контекст выполнения текущей сессии на пользователя `test2`.

```
EXECUTE AS test2 WITH NO REVERT;
```

После успешного переключения вы можете выполнить команду `select current_user()`, чтобы получить текущего пользователя.

```
select current_user();
+-----------------------------+
| CURRENT_USER()              |
+-----------------------------+
| 'default_cluster:test2'@'%' |
+-----------------------------+
```

GRANT

Предоставляет роли или привилегии на конкретные объекты пользователю, роли или внешней группе.

Для получения дополнительной информации о привилегиях, которые могут быть предоставлены, см. Элементы привилегий.

Для получения дополнительной информации о создании внешней группы см. Group Provider.

После выполнения операции GRANT вы можете выполнить SHOW GRANTS для просмотра детальной информации о привилегиях или выполнить REVOKE для отзыва привилегии или роли.

Перед выполнением операции GRANT убедитесь, что соответствующий пользователь или роль были созданы. Для получения дополнительной информации см. CREATE USER и CREATE ROLE.

ПОДСКАЗКА

- Только пользователи с ролью `user_admin` могут предоставлять любые привилегии другим пользователям и ролям.
- После предоставления роли пользователю необходимо выполнить SET ROLE, чтобы активировать эту роль перед выполнением операций от имени этой роли. Если вы хотите, чтобы все роли по умолчанию активировались при входе, выполните ALTER USER или SET DEFAULT ROLE. Если вы хотите, чтобы все привилегии в системе активировались для всех пользователей при входе, установите глобальную переменную `SET GLOBAL activate_all_roles_on_login = TRUE;`.
- Обычные пользователи могут предоставлять только привилегии с ключевым словом `WITH GRANT OPTION` другим пользователям и ролям.

Синтаксис

Предоставление привилегий ролям или пользователям

System

```
GRANT
    { CREATE RESOURCE GROUP | CREATE RESOURCE | CREATE EXTERNAL CATALOG |
REPOSITORY | BLACKLIST | FILE | OPERATE | CREATE STORAGE VOLUME | SECURITY
}
    ON SYSTEM
    TO { ROLE | USER } { <role_name> | <user_identity> } [ WITH GRANT
OPTION ]
```

Resource group

```
GRANT
    { ALTER | DROP | ALL [PRIVILEGES] }
    ON { RESOURCE GROUP <resource_group_name> [, <resource_group_name
>,...] | ALL RESOURCE GROUPS}
    TO { ROLE | USER } { <role_name> | <user_identity> } [ WITH GRANT
OPTION ]
```

Resource

```
GRANT
    { USAGE | ALTER | DROP | ALL [PRIVILEGES] }
    ON { RESOURCE <resource_name> [, < resource_name >,...] | ALL
RESOURCES}
    TO { ROLE | USER } { <role_name> | <user_identity> } [ WITH GRANT
OPTION ]
```

Global UDF

```
GRANT
    { USAGE | DROP | ALL [PRIVILEGES]}
    ON { GLOBAL FUNCTION <function_name>(input_data_type) [,
<function_name>(input_data_type),...]
       | ALL GLOBAL FUNCTIONS }
    TO { ROLE | USER } { <role_name> | <user_identity> } [ WITH GRANT
OPTION ]
```

Пример: `GRANT usage ON GLOBAL FUNCTION a(string) to kevin;`

Internal catalog

```
GRANT
    { USAGE | CREATE DATABASE | ALL [PRIVILEGES]}
    ON CATALOG default_catalog
    TO { ROLE | USER } { <role_name> | <user_identity> } [ WITH GRANT
OPTION ]
```

External catalog

```
GRANT
   { USAGE | DROP | ALL [PRIVILEGES] }
   ON { CATALOG <catalog_name> [, <catalog_name>,...] | ALL CATALOGS}
   TO { ROLE | USER } { <role_name> | <user_identity> } [ WITH GRANT OPTION
]
```

Database

```
GRANT
    { ALTER | DROP | CREATE TABLE | CREATE VIEW | CREATE FUNCTION | CREATE
MATERIALIZED VIEW | ALL [PRIVILEGES] }
    ON { DATABASE <database_name> [, <database_name>,...] | ALL DATABASES }
    TO { ROLE | USER } { <role_name> | <user_identity> } [ WITH GRANT
OPTION ]
```

- Сначала необходимо выполнить SET CATALOG перед выполнением этой команды.
- Для баз данных в External Catalog вы можете предоставить привилегию CREATE TABLE только для баз данных Hive (начиная с версии v1.5.2) и Iceberg (начиная с версии v1.5.2).

Table

```
-- Предоставление привилегий на КОНКРЕТНЫЕ ТАБЛИЦЫ.
  GRANT
    { ALTER | DROP | SELECT | INSERT | EXPORT | UPDATE | DELETE | ALL
[PRIVILEGES]}
    ON TABLE <table_name> [, < table_name >,...]
    TO { ROLE | USER } { <role_name> | <user_identity> } [ WITH GRANT
OPTION ]

-- Предоставление привилегий на ВСЕ ТАБЛИЦЫ в конкретной базе данных или во
всех базах данных.
  GRANT
    { ALTER | DROP | SELECT | INSERT | EXPORT | UPDATE | DELETE | ALL
[PRIVILEGES]}
    ON ALL TABLES IN { { DATABASE <database_name> } | ALL DATABASES }
    TO { ROLE | USER } { <role_name> | <user_identity> } [ WITH GRANT
OPTION ]
```

- Сначала необходимо выполнить SET CATALOG перед выполнением этой команды.
- Вы также можете использовать `<db_name>.<table_name>` для представления таблицы.
- Вы можете предоставить привилегию SELECT на все таблицы в Internal и External Catalogs для чтения данных из этих таблиц. Для таблиц в Hive и Iceberg Catalogs вы

можете предоставить привилегию INSERT для записи данных в такие таблицы (поддерживается начиная с v1.5.2 для Iceberg и v1.5.2 для Hive)

```
GRANT <priv> ON TABLE <db_name>.<table_name> TO { ROLE <role_name> |
USER <user_name> }
```

View

```
GRANT
    { ALTER | DROP | SELECT | ALL [PRIVILEGES]}
    ON { VIEW <view_name> [, < view_name >,...]
        | ALL VIEWS} IN
            { { DATABASE <database_name> } | ALL DATABASES }
    TO { ROLE | USER } { <role_name> | <user_identity> } [ WITH GRANT
OPTION ]
```

- Сначала необходимо выполнить SET CATALOG перед выполнением этой команды.
- Вы также можете использовать `<db_name>.<view_name>` для представления view.
- Для таблиц в External Catalog вы можете предоставить только привилегию SELECT на view таблиц Hive (начиная с версии v1.5.2).

```
GRANT <priv> ON VIEW <db_name>.<view_name> TO { ROLE <role_name> | USER
<user_name> }
```

Materialized view

```
GRANT
    { SELECT | ALTER | REFRESH | DROP | ALL [PRIVILEGES]}
    ON { MATERIALIZED VIEW <mv_name> [, < mv_name >,...]
        | ALL MATERIALIZED VIEWS} IN
            { { DATABASE <database_name> } | ALL DATABASES }
    TO { ROLE | USER } { <role_name> | <user_identity> } [ WITH GRANT
OPTION ]
```

- Сначала необходимо выполнить SET CATALOG перед выполнением этой команды.
- Вы также можете использовать `<db_name>.<mv_name>` для представления mv.

```
GRANT <priv> ON MATERIALIZED VIEW <db_name>.<mv_name> TO { ROLE
<role_name> | USER <user_name> }
```

Function

```
GRANT
    { USAGE | DROP | ALL [PRIVILEGES]}
    ON { FUNCTION <function_name>(input_data_type) [, < function_name >
(input_data_type),...]
        | ALL FUNCTIONS} IN
            { { DATABASE <database_name> } | ALL DATABASES }
    TO { ROLE | USER } { <role_name> | <user_identity> } [ WITH GRANT
OPTION ]
```

- Сначала необходимо выполнить SET CATALOG перед выполнением этой команды.
- Вы также можете использовать `<db_name>.<function_name>` для представления function.

```
GRANT <priv> ON FUNCTION <db_name>.<function_name>(input_data_type) TO
{ ROLE <role_name> | USER <user_name> }
```

User

```
GRANT IMPERSONATE
ON USER <user_identity>
TO USER <user_identity_1> [ WITH GRANT OPTION ]
```

Storage volume

```
GRANT
    { USAGE | ALTER | DROP | ALL [PRIVILEGES] }
    ON { STORAGE VOLUME < name > [, < name >,...] | ALL STORAGE VOLUMES}
    TO { ROLE | USER } { <role_name> | <user_identity> } [ WITH GRANT
OPTION ]
```

Предоставление ролей ролям, пользователям или внешним группам

```
GRANT <role_name> [,<role_name>, ...] TO ROLE <role_name>
GRANT <role_name> [,<role_name>, ...] TO USER <user_identity>
GRANT <role_name> [,<role_name>, ...] TO EXTERNAL GROUP
<external_group_name>
```

Примеры

Пример 1: Предоставить привилегию чтения данных из всех таблиц во всех базах данных пользователю `jack`.

```
GRANT SELECT ON *.* TO 'jack'@'%';
```

Пример 2: Предоставить привилегию загрузки данных во все таблицы базы данных `db1` роли `my_role`.

```
GRANT INSERT ON db1.* TO ROLE 'my_role';
```

Пример 3: Предоставить привилегии чтения, обновления и загрузки данных в таблицу `tbl1` базы данных `db1` пользователю `jack`.

```
GRANT SELECT,ALTER,INSERT ON db1.tbl1 TO 'jack'@'192.8.%';
```

Пример 4: Предоставить привилегию использования всех ресурсов пользователю `jack`.

```
GRANT USAGE ON RESOURCE * TO 'jack'@'%';
```

Пример 5: Предоставить привилегию использования ресурса `spark_resource` пользователю `jack`.

```
GRANT USAGE ON RESOURCE 'spark_resource' TO 'jack'@'%';
```

Пример 6: Предоставить привилегию использования ресурса `spark_resource` роли `my_role`.

```
GRANT USAGE ON RESOURCE 'spark_resource' TO ROLE 'my_role';
```

Пример 7: Предоставить привилегию чтения данных из таблицы `sr_member` пользователю `jack` и разрешить пользователю `jack` предоставлять эту привилегию другим пользователям или ролям (указав WITH GRANT OPTION).

```
GRANT SELECT ON TABLE sr_member TO USER jack@'172.10.1.10' WITH GRANT
OPTION;
```

Пример 8: Предоставить системные роли `db_admin`, `user_admin` и `cluster_admin` пользователю `user_platform`.

```
GRANT db_admin, user_admin, cluster_admin TO USER user_platform;
```

Пример 9: Разрешить пользователю `jack` выполнять операции от имени пользователя `rose`.

```
GRANT IMPERSONATE ON USER 'rose'@'%' TO USER 'jack'@'%';
```

Пример 10: Предоставить системные роли `db_admin`, `user_admin` и `cluster_admin` внешней группе `admin_group`.

```
GRANT db_admin, user_admin, cluster_admin TO EXTERNAL GROUP admin_group;
```

Лучшие практики

Настройка ролей в зависимости от сценариев

Мы рекомендуем создавать пользовательские роли для управления привилегиями и пользователями. Следующие примеры классифицируют несколько комбинаций привилегий для распространённых сценариев.

Предоставление глобальных привилегий только на чтение для таблиц Selena

```
-- Создать роль.
CREATE ROLE read_only;
-- Предоставить привилегию USAGE на все каталоги роли.
GRANT USAGE ON ALL CATALOGS TO ROLE read_only;
-- Предоставить привилегию на запрос всех таблиц роли.
GRANT SELECT ON ALL TABLES IN ALL DATABASES TO ROLE read_only;
-- Предоставить привилегию на запрос всех представлений роли.
GRANT SELECT ON ALL VIEWS IN ALL DATABASES TO ROLE read_only;
-- Предоставить привилегию на запрос всех материализованных представлений и
привилегию на ускорение запросов с их помощью роли.
GRANT SELECT ON ALL MATERIALIZED VIEWS IN ALL DATABASES TO ROLE read_only;
```

Вы также можете предоставить привилегию на использование UDF в запросах:

```
-- Предоставить привилегию USAGE на все UDF уровня базы данных роли.
GRANT USAGE ON ALL FUNCTIONS IN ALL DATABASES TO ROLE read_only;
-- Предоставить привилегию USAGE на глобальные UDF роли.
GRANT USAGE ON ALL GLOBAL FUNCTIONS TO ROLE read_only;
```

Предоставление глобальных привилегий только на запись для таблиц Selena

```
-- Создать роль.
CREATE ROLE write_only;
-- Предоставить привилегию USAGE на все каталоги роли.
GRANT USAGE ON ALL CATALOGS TO ROLE write_only;
-- Предоставить привилегии INSERT и UPDATE на все таблицы роли.
GRANT INSERT, UPDATE ON ALL TABLES IN ALL DATABASES TO ROLE write_only;
-- Предоставить привилегию REFRESH на все материализованные представления
роли.
GRANT REFRESH ON ALL MATERIALIZED VIEWS IN ALL DATABASES TO ROLE
write_only;
```

Предоставление привилегий только на чтение для определённого внешнего каталога

```
-- Создать роль.
CREATE ROLE read_catalog_only;
-- Предоставить привилегию USAGE на целевой каталог роли.
GRANT USAGE ON CATALOG hive_catalog TO ROLE read_catalog_only;
-- Переключиться на соответствующий каталог.
SET CATALOG hive_catalog;
-- Предоставить привилегии на запрос всех таблиц и всех представлений во
внешнем каталоге.
GRANT SELECT ON ALL TABLES IN ALL DATABASES TO ROLE read_catalog_only;
```

ПОДСКАЗКА

Для представлений во внешних каталогах можно запрашивать только представления таблиц Hive (начиная с v1.5.2).

Предоставление привилегий только на запись для определённого внешнего каталога

Вы можете записывать данные только в таблицы Iceberg (начиная с v1.5.2) и таблицы Hive (начиная с v1.5.2).

```
-- Создать роль.
CREATE ROLE write_catalog_only;
-- Предоставить привилегию USAGE на целевой каталог роли.
GRANT USAGE ON CATALOG iceberg_catalog TO ROLE read_catalog_only;
-- Переключиться на соответствующий каталог.
SET CATALOG iceberg_catalog;
-- Предоставить привилегию на запись данных в таблицы Iceberg.
GRANT INSERT ON ALL TABLES IN ALL DATABASES TO ROLE write_catalog_only;
```

Предоставление административных привилегий для определённой базы данных

```
-- Создать роль.
CREATE ROLE db1_admin;
-- Предоставить ВСЕ привилегии на целевую базу данных роли. Эта роль может
создавать таблицы, представления, материализованные представления и UDF в
этой базе данных. Она также может удалять или изменять эту базу данных.
GRANT ALL ON DATABASE db1 TO ROLE db1_admin;
-- Переключиться на соответствующий каталог.
SET CATALOG iceberg_catalog;
-- Предоставить все привилегии на таблицы, представления, материализованные
представления и UDF в этой базе данных роли.
GRANT ALL ON ALL TABLES IN DATABASE db1 TO ROLE db1_admin;
GRANT ALL ON ALL VIEWS IN DATABASE db1 TO ROLE db1_admin;
GRANT ALL ON ALL MATERIALIZED VIEWS IN DATABASE db1 TO ROLE db1_admin;
GRANT ALL ON ALL FUNCTIONS IN DATABASE db1 TO ROLE db1_admin;
```

Предоставление привилегий для выполнения операций резервного копирования и восстановления на глобальном уровне, уровне базы данных, таблицы и partition

- Предоставление привилегий для выполнения глобальных операций резервного копирования и восстановления:

  Привилегии для выполнения глобальных операций резервного копирования и восстановления позволяют роли создавать резервные копии и восстанавливать любую базу данных, таблицу или partition. Требуется привилегия REPOSITORY на уровне SYSTEM, привилегии на создание баз данных в каталоге по умолчанию, на создание таблиц в любой базе данных, а также на загрузку и экспорт данных в любой таблице.

```
-- Создать роль.
CREATE ROLE recover;
-- Предоставить привилегию REPOSITORY на уровне SYSTEM.
GRANT REPOSITORY ON SYSTEM TO ROLE recover;
-- Предоставить привилегию на создание баз данных в каталоге по
умолчанию.
GRANT CREATE DATABASE ON CATALOG default_catalog TO ROLE recover;
-- Предоставить привилегию на создание таблиц в любой базе данных.
GRANT CREATE TABLE ON ALL DATABASES TO ROLE recover;
-- Предоставить привилегию на загрузку и экспорт данных в любой
таблице.
GRANT INSERT, EXPORT ON ALL TABLES IN ALL DATABASES TO ROLE recover;
```

- Предоставление привилегий для выполнения операций резервного копирования и восстановления на уровне базы данных:

  Привилегии для выполнения операций резервного копирования и восстановления на уровне базы данных требуют привилегии REPOSITORY на уровне SYSTEM, привилегии на создание баз данных в каталоге по умолчанию, привилегии на создание таблиц в любой базе данных, привилегии на загрузку данных в любую таблицу и привилегии на экспорт данных из любой таблицы в базе данных для резервного копирования.

```
-- Создать роль.
CREATE ROLE recover_db;
-- Предоставить привилегию REPOSITORY на уровне SYSTEM.
GRANT REPOSITORY ON SYSTEM TO ROLE recover_db;
-- Предоставить привилегию на создание баз данных.
GRANT CREATE DATABASE ON CATALOG default_catalog TO ROLE recover_db;
-- Предоставить привилегию на создание таблиц.
GRANT CREATE TABLE ON ALL DATABASES TO ROLE recover_db;
-- Предоставить привилегию на загрузку данных в любую таблицу.
GRANT INSERT ON ALL TABLES IN ALL DATABASES TO ROLE recover_db;
-- Предоставить привилегию на экспорт данных из любой таблицы в базе
данных для резервного копирования.
GRANT EXPORT ON ALL TABLES IN DATABASE <db_name> TO ROLE recover_db;
```

- Предоставление привилегий для выполнения операций резервного копирования и восстановления на уровне таблицы:

Привилегии для выполнения операций резервного копирования и восстановления на уровне таблицы требуют привилегии REPOSITORY на уровне SYSTEM, привилегии на создание таблиц в соответствующих базах данных, привилегии на загрузку данных в любую таблицу в базе данных и привилегии на экспорт данных из таблицы для резервного копирования.

```
-- Создать роль.
CREATE ROLE recover_tbl;
-- Предоставить привилегию REPOSITORY на уровне SYSTEM.
GRANT REPOSITORY ON SYSTEM TO ROLE recover_tbl;
-- Предоставить привилегию на создание таблиц в соответствующих базах
данных.
GRANT CREATE TABLE ON DATABASE <db_name> TO ROLE recover_tbl;
-- Предоставить привилегию на загрузку данных в любую таблицу в базе
данных.
GRANT INSERT ON ALL TABLES IN DATABASE <db_name> TO ROLE recover_db;
-- Предоставить привилегию на экспорт данных из таблицы, которую вы
хотите скопировать.
GRANT EXPORT ON TABLE <table_name> TO ROLE recover_tbl;
```

- Предоставление привилегий для выполнения операций резервного копирования и восстановления на уровне partition:

  Привилегии для выполнения операций резервного копирования и восстановления на уровне partition требуют привилегии REPOSITORY на уровне SYSTEM и привилегии на загрузку и экспорт данных в соответствующей таблице.

```
-- Создать роль.
CREATE ROLE recover_par;
-- Предоставить привилегию REPOSITORY на уровне SYSTEM.
GRANT REPOSITORY ON SYSTEM TO ROLE recover_par;
-- Предоставить привилегию на загрузку и экспорт данных в
соответствующей таблице.
GRANT INSERT, EXPORT ON TABLE <table_name> TO ROLE recover_par;
```

Лучшие практики управления многосервисным доступом см. в разделе Управление многосервисным доступом.

REVOKE

Используйте REVOKE для удаления конкретных привилегий или ролей у пользователя или роли. Список привилегий, поддерживаемых Selena, см. в разделе Привилегии, поддерживаемые Selena.

ПОДСКАЗКА

- Обычные пользователи могут отзывать у других пользователей и ролей только свои привилегии, имеющие ключевое слово `WITH GRANT OPTION`. Для получения информации о `WITH GRANT OPTION` см. GRANT.
- Только пользователи с ролью `user_admin` имеют привилегию отзыва привилегий у других пользователей.

Синтаксис

Отзыв привилегий

Привилегии, которые могут быть отозваны, зависят от объекта. В следующей части описывается синтаксис в зависимости от объектов.

System

```
REVOKE
    { CREATE RESOURCE GROUP | CREATE RESOURCE | CREATE EXTERNAL CATALOG |
REPOSITORY | BLACKLIST | FILE | OPERATE }
    ON SYSTEM
    FROM { ROLE | USER} {<role_name>|<user_identity>}
```

Resource group

```
REVOKE
    { ALTER | DROP | ALL [PRIVILEGES] }
    ON { RESOURCE GROUP <resourcegroup_name> [, <resourcegroup_name>,...]
 | ALL RESOURCE GROUPS}
    FROM { ROLE | USER} {<role_name>|<user_identity>}
```

Resource

```
REVOKE
    { USAGE | ALTER | DROP | ALL [PRIVILEGES] }
    ON { RESOURCE <resource_name> [, <resource_name>,...] | ALL RESOURCES}
    FROM { ROLE | USER} {<role_name>|<user_identity>}
```

User

```
REVOKE IMPERSONATE ON USER <user_identity> FROM USER <user_identity>;
```

Global UDF

```
REVOKE
    { USAGE | DROP | ALL [PRIVILEGES]}
    ON { GLOBAL FUNCTION <function_name>(input_data_type) [,
<function_name>(input_data_type),...]
       | ALL GLOBAL FUNCTIONS }
    FROM { ROLE | USER} {<role_name>|<user_identity>}
```

Internal catalog

```
REVOKE
    { USAGE | CREATE DATABASE | ALL [PRIVILEGES]}
    ON CATALOG default_catalog
    FROM { ROLE | USER} {<role_name>|<user_identity>}
```

External catalog

```
REVOKE
   { USAGE | DROP | ALL [PRIVILEGES] }
   ON { CATALOG <catalog_name> [, <catalog_name>,...] | ALL CATALOGS}
   FROM { ROLE | USER} {<role_name>|<user_identity>}
```

Database

```
REVOKE
    { ALTER | DROP | CREATE TABLE | CREATE VIEW | CREATE FUNCTION | CREATE
MATERIALIZED VIEW | ALL [PRIVILEGES] }
    ON {{ DATABASE <database_name> [, <database_name>,...]} | ALL DATABASES
}
    FROM { ROLE | USER} {<role_name>|<user_identity>}
```

- Сначала необходимо выполнить SET CATALOG перед выполнением этой команды.

Table

```
REVOKE
    { ALTER | DROP | SELECT | INSERT | EXPORT | UPDATE | DELETE | ALL
[PRIVILEGES]}
    ON { TABLE <table_name> [, < table_name >,...]
       | ALL TABLES} IN
           { { DATABASE <database_name> [, <database_name>,...]} | ALL
DATABASES }
    FROM { ROLE | USER} {<role_name>|<user_identity>}
```

- Сначала необходимо выполнить SET CATALOG перед выполнением этой команды.
- Вы также можете использовать db.tbl для представления таблицы.

```
REVOKE <priv> ON TABLE db.tbl FROM {ROLE <role_name> | USER
<user_identity>}
```

View

```
REVOKE
    { ALTER | DROP | SELECT | ALL [PRIVILEGES]}
    ON { VIEW <view_name> [, < view_name >,...]
       | ALL VIEWS} IN
           { { DATABASE <database_name> [, <database_name>,...]}  | ALL
DATABASES }
    FROM { ROLE | USER} {<role_name>|<user_identity>}
```

- Сначала необходимо выполнить SET CATALOG перед выполнением этой команды.

- Вы также можете использовать db.view для представления view.

```
REVOKE <priv> ON VIEW db.view FROM {ROLE <role_name> | USER
<user_identity>}
```

Materialized view

```
REVOKE
    { SELECT | ALTER | REFRESH | DROP | ALL [PRIVILEGES]}
    ON { MATERIALIZED VIEW <mv_name> [, < mv_name >,...]
       | ALL MATERIALIZED VIEWS} IN
           { { DATABASE <database_name> [, <database_name>,...] } | ALL
[DATABASES] }
    FROM { ROLE | USER} {<role_name>|<user_identity>}
```

- Сначала необходимо выполнить SET CATALOG перед выполнением этой команды.

- Вы также можете использовать db.mv для представления mv.

```
REVOKE <priv> ON MATERIALIZED VIEW db.mv FROM {ROLE <role_name> | USER
<user_identity>}
```

Function

```
REVOKE
    { USAGE | DROP | ALL [PRIVILEGES]}
    ON { FUNCTION <function_name>(input_data_type) [, <function_name>
(input_data_type),...]
       | ALL FUNCTIONS} IN
           { { DATABASE <database_name> [, <database_name>,...] } | ALL
DATABASES }
    FROM { ROLE | USER} {<role_name>|<user_identity>}
```

- Сначала необходимо выполнить SET CATALOG перед выполнением этой команды.

- Вы также можете использовать db.function для представления function.

```
REVOKE <priv> ON FUNCTION <db_name>.<function_name>(input_data_type)
FROM {ROLE <role_name> | USER <user_identity>}
```

Storage volume

```
REVOKE
    CREATE STORAGE VOLUME
    ON SYSTEM
    FROM { ROLE | USER} {<role_name>|<user_identity>}

REVOKE
    { USAGE | ALTER | DROP | ALL [PRIVILEGES] }
    ON { STORAGE VOLUME < name > [, < name >,...] | ALL STORAGE VOLUME}
    FROM { ROLE | USER} {<role_name>|<user_identity>}
```

Отзыв ролей

```
REVOKE <role_name> [,<role_name>, ...] FROM ROLE <role_name>
REVOKE <role_name> [,<role_name>, ...] FROM USER <user_identity>
```

Параметры

Параметр	Описание
role_name	Имя роли.
user_identity	Идентификатор пользователя, например 'jack'@'192.%'.
resourcegroup_name	Имя resource group
resource_name	Имя ресурса.
function_name	Имя функции.
catalog_name	Имя external catalog.
database_name	Имя базы данных.
table_name	Имя таблицы.
view_name	Имя view.
mv_name	Имя materialized view.

Примеры

Отзыв привилегий

Отозвать привилегию SELECT на таблицу `sr_member` у пользователя `jack`:

```
REVOKE SELECT ON TABLE sr_member FROM USER 'jack'@'192.%'
```

Отозвать привилегию USAGE на ресурс `spark_resource` у роли `test_role`:

```
REVOKE USAGE ON RESOURCE 'spark_resource' FROM ROLE 'test_role';
```

Отзыв ролей

Отозвать роль `example_role` у пользователя `jack`:

```
REVOKE example_role FROM 'jack'@'%';
```

Отозвать роль `example_role` у роли `test_role`:

```
REVOKE example_role FROM ROLE 'test_role';
```

Ссылки

GRANT

SET DEFAULT ROLE

SET DEFAULT ROLE устанавливает роли, которые активируются по умолчанию, когда пользователь подключается к серверу.

Эта команда поддерживается начиная с версии v1.5.2.

> **ПОДСКАЗКА**
>
> Обычные пользователи могут устанавливать роли по умолчанию для себя. Пользователи с ролью `user_admin` могут устанавливать роли по умолчанию для других пользователей. Перед выполнением этой операции убедитесь, что пользователю уже назначены эти роли.
>
> Вы можете запросить роли пользователя с помощью SHOW GRANTS.

Синтаксис

```
-- Установить указанные роли в качестве ролей по умолчанию.
SET DEFAULT ROLE <role_name>[,<role_name>,..] TO <user_identity>;
-- Установить все роли пользователя, включая роли, которые будут назначены
этому пользователю, в качестве ролей по умолчанию.
SET DEFAULT ROLE ALL TO <user_identity>;
-- Роль по умолчанию не установлена, но публичная роль (public) все еще
активна после входа пользователя.
SET DEFAULT ROLE NONE TO <user_identity>;
```

Параметры

`role_name`: имя роли

`user_identity`: идентификатор пользователя

Примеры

Запросить роли текущего пользователя.

```
SHOW GRANTS FOR test;
+--------------+---------+-----------------------------------------------+
| UserIdentity | Catalog | Grants                                        |
+--------------+---------+-----------------------------------------------+
| 'test'@'%'   | NULL    | GRANT 'db_admin', 'user_admin' TO 'test'@'%' |
+--------------+---------+-----------------------------------------------+
```

Пример 1: Установить роли `db_admin` и `user_admin` в качестве ролей по умолчанию для пользователя `test`.

```
SET DEFAULT ROLE db_admin TO test;
```

Пример 2: Установить все роли пользователя `test`, включая роли, которые будут назначены этому пользователю, в качестве ролей по умолчанию.

```
SET DEFAULT ROLE ALL TO test;
```

Пример 3: Очистить все роли по умолчанию пользователя `test`.

```
SET DEFAULT ROLE NONE TO test;
```

SET PASSWORD

SET PASSWORD изменяет пароль для входа пользователей. Команда ALTER USER также может использоваться для изменения пароля.

> **ПОДСКАЗКА**
>
> - Все пользователи могут сбросить свой собственный пароль.
> - Только пользователи с ролью `user_admin` могут изменять пароль других пользователей.
> - Только пользователь `root` может изменить свой собственный пароль. Для получения дополнительной информации см. обзор привилегий.

Синтаксис

```
SET PASSWORD [FOR user_identity] =
[PASSWORD('plain password')]|['hashed password']
```

`user_identity` должен точно соответствовать `user_identity`, указанному при создании пользователя с помощью CREATE USER. В противном случае будет сообщено, что пользователь не существует. Если `user_identity` не указан, будет изменен пароль текущего пользователя. Текущего пользователя можно просмотреть через SHOW GRANTS.

`PASSWORD()` принимает пароль в открытом виде. Если вы вводите строку напрямую без использования `PASSWORD()`, строка должна быть зашифрована.

Примеры

1. Установить пароль для текущего пользователя.

```
SET PASSWORD = PASSWORD('123456')
SET PASSWORD = '*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9'
```

2. Установить пароль для указанного пользователя.

```
SET PASSWORD FOR 'jack'@'192.%' = PASSWORD('123456')
SET PASSWORD FOR 'jack'@['domain'] =
'*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9'
```

SET ROLE

SET ROLE активирует роли вместе со всеми связанными привилегиями и вложенными ролями для текущей сессии. После активации роли пользователи могут использовать эту роль для выполнения операций.

После выполнения этой команды вы можете запустить `select is_role_in_session("<role_name>");`, чтобы проверить, активна ли эта роль в текущей сессии.

Эта команда поддерживается начиная с версии v1.5.2.

Примечания по использованию

Пользователи могут активировать только роли, которые были им назначены.

Вы можете запросить роли пользователя с помощью SHOW GRANTS.

Вы можете запросить активные роли текущего пользователя с помощью `SELECT CURRENT_ROLE()`. Для получения дополнительной информации см. current_role.

Синтаксис

```
-- Активировать определенные роли и выполнять операции от имени этой роли.
SET ROLE <role_name>[,<role_name>,..];
-- Активировать все роли пользователя, за исключением определенных ролей.
SET ROLE ALL EXCEPT <role_name>[,<role_name>,..];
-- Активировать все роли пользователя.
SET ROLE ALL;
```

Параметры

`role_name`: имя роли

Примеры

Запросить все роли текущего пользователя.

```
SHOW GRANTS;
+--------------+---------+-----------------------------------------------+
| UserIdentity | Catalog | Grants                                        |
+--------------+---------+-----------------------------------------------+
| 'test'@'%'   | NULL    | GRANT 'db_admin', 'user_admin' TO 'test'@'%' |
+--------------+---------+-----------------------------------------------+
```

Активировать роль `db_admin`.

```
SET ROLE db_admin;
```

Запросить активные роли текущего пользователя.

```
SELECT CURRENT_ROLE();
+--------------------+
| CURRENT_ROLE()     |
+--------------------+
| db_admin           |
+--------------------+
```

Ссылки

- CREATE ROLE: создает роль.
- GRANT: назначает роли пользователям или другим ролям.
- ALTER USER: изменяет роли.
- SHOW ROLES: показывает все роли в системе.
- current_role: показывает роли текущего пользователя.
- is_role_in_session: проверяет, активна ли роль (или вложенная роль) в текущей сессии.
- DROP ROLE: удаляет роль.

SHOW AUTHENTICATION

SHOW AUTHENTICATION отображает информацию об аутентификации текущего пользователя или всех пользователей в текущем кластере.

ПОДСКАЗКА

Все пользователи имеют право просматривать свою собственную информацию об аутентификации. Только пользователи с ролью `user_admin` могут просматривать информацию об аутентификации всех пользователей или указанных пользователей.

Синтаксис

```
SHOW [ALL] AUTHENTICATION [FOR USERNAME]
```

Параметры

Параметр	Обязательный	Описание
ALL	Нет	Если указано это ключевое слово, возвращается информация об аутентификации всех пользователей в текущем кластере. Если это ключевое слово не указано, возвращается только информация об аутентификации текущего пользователя.
USERNAME	Нет	Если указан этот параметр, можно просмотреть информацию об аутентификации указанного пользователя. Если этот параметр не указан, можно просмотреть только информацию об аутентификации текущего пользователя.

Вывод

```
+---------------+----------+-------------+-------------------+
| UserIdentity  | Password | AuthPlugin  | UserForAuthPlugin |
+---------------+----------+-------------+-------------------+
```

Поле	Описание
UserIdentity	Идентификатор пользователя.
Password	Используется ли пароль для входа в кластер Selena. • `Yes`: Пароль используется. • `No`: Пароль не используется.
AuthPlugin	Интерфейс, который используется для аутентификации. Допустимые значения: `MYSQL_NATIVE_PASSWORD` и `AUTHENTICATION_LDAP_SIMPLE`. Если интерфейс не используется, возвращается `NULL`.
UserForAuthPlugin	Имя пользователя, использующего аутентификацию LDAP или Kerberos. Если аутентификация не используется, возвращается `NULL`.

Примеры

Пример 1: Отображение информации об аутентификации текущего пользователя.

```
SHOW AUTHENTICATION;
+--------------+----------+------------+-------------------+
| UserIdentity | Password | AuthPlugin | UserForAuthPlugin |
+--------------+----------+------------+-------------------+
| 'root'@'%'   | No       | NULL       | NULL              |
+--------------+----------+------------+-------------------+
```

Пример 2: Отображение информации об аутентификации всех пользователей в текущем кластере.

```
SHOW ALL AUTHENTICATION;
+---------------+----------+----------------------------+------------------
-+
| UserIdentity  | Password | AuthPlugin                 | UserForAuthPlugin
|
+---------------+----------+----------------------------+------------------
-+
| 'root'@'%'    | Yes      | NULL                       | NULL
|
| 'chelsea'@'%' | No       | AUTHENTICATION_LDAP_SIMPLE | NULL
|
+---------------+----------+----------------------------+------------------
-+
```

Пример 3: Отображение информации об аутентификации указанного пользователя.

```
SHOW AUTHENTICATION FOR root;
+--------------+----------+------------+-------------------+
| UserIdentity | Password | AuthPlugin | UserForAuthPlugin |
+--------------+----------+------------+-------------------+
| 'root'@'%'   | Yes      | NULL       | NULL              |
+--------------+----------+------------+-------------------+
```

SHOW GRANTS

SHOW GRANTS отображает все привилегии, которые были предоставлены пользователю или роли.

Для получения дополнительной информации о ролях и привилегиях см. Обзор привилегий.

> **ПОДСКАЗКА**
>
> Все роли и пользователи могут просматривать привилегии, предоставленные им, или роли, назначенные им. Только пользователи с ролью `user_admin` могут просматривать привилегии указанного пользователя или роли.

Синтаксис

```
SHOW GRANTS; -- Просмотр привилегий текущего пользователя.
SHOW GRANTS FOR ROLE <role_name>; -- Просмотр привилегий конкретной роли.
SHOW GRANTS FOR <user_identity>; -- Просмотр привилегий конкретного
пользователя.
```

Параметры

- role_name
- user_identity

Возвращаемые поля:

```
-- Просмотр привилегий конкретного пользователя.
+--------------+--------+---------------------------------------------+
|UserIdentity  |Catalog | Grants                                      |
+--------------+--------+---------------------------------------------+

-- Просмотр привилегий конкретной роли.
+-------------+--------+-------------------------------------------------
----+
|RoleName     |Catalog | Grants
|
+-------------+----------------+-----------------------------------------
----+
```

Поле	Описание
UserIdentity	Идентификатор пользователя, который отображается при запросе привилегий пользователя.
RoleName	Имя роли, которое отображается при запросе привилегий роли.
Catalog	Имя каталога. `default` возвращается, если операция GRANT выполняется над внутренним каталогом Selena. Имя внешнего каталога возвращается, если операция GRANT выполняется над внешним каталогом. `NULL` возвращается, если операция, показанная в столбце `Grants`, является назначением ролей.
Grants	Конкретная операция GRANT.

Примеры

```
mysql> SHOW GRANTS;
+--------------+---------+----------------------------------------+
| UserIdentity | Catalog | Grants                                 |
+--------------+---------+----------------------------------------+
| 'root'@'%'   | NULL    | GRANT 'root', 'testrole' TO 'root'@'%' |
+--------------+---------+----------------------------------------+

mysql> SHOW GRANTS FOR 'user_g'@'%';
+-------------+-------------+-----------------------------------------------
-------------------------------------------------+
|UserIdentity |Catalog      |Grants
|
+-------------+-------------------------------------------------------------
-------------------------------------------------+
|'user_g'@'%' |NULL         |GRANT role_g, public to `user_g`@`%`;
|
|'user_g'@'%' |NULL         |GRANT IMPERSONATE ON USER `user_a`@`%` TO USER
`user_g`@`%`;                                |
|'user_g'@'%' |default      |GRANT CREATE_DATABASE ON CATALOG
default_catalog TO USER `user_g`@`%`;                        |
|'user_g'@'%' |default      |GRANT ALTER, DROP, CREATE_TABLE ON DATABASE
db1 TO USER `user_g`@`%`;                          |
|'user_g'@'%' |default      |GRANT CREATE_VIEW ON DATABASE db1 TO USER
`user_g`@`%` WITH GRANT OPTION;                      |
|'user_g'@'%' |default      |GRANT ALTER, DROP, SELECT, INGEST, EXPORT,
DELETE, UPDATE ON TABLE db.* TO USER `user_g`@`%`;  |
|'user_g'@'%' |default      |GRANT ALTER, DROP, SELECT ON VIEW db2.view TO
USER `user_g`@`%`;                                |
|'user_g'@'%' |Hive_catalog |GRANT USAGE ON CATALOG Hive_catalog TO USER
`user_g`@`%`                                       |
+-------------+--------------+----------------------------------------------
-------------------------------------------------+

mysql> SHOW GRANTS FOR ROLE role_g;
+-------------+--------+-------------------------------------------------
----+
|RoleName     |Catalog | Grants
|
+-------------+-----------------+-----------------------------------------
----+
|role_g       |NULL    | GRANT role_p, role_test TO ROLE role_g;
|
|role_g       |default | GRANT SELECT ON *.* TO ROLE role_g WITH GRANT
```

```
OPTION; |
+-------------+--------+-------------------------------------------------
-----+
```

Ссылки

GRANT

SHOW PROPERTY

SHOW PROPERTY отображает свойства пользователя. В настоящее время с помощью этой команды можно просмотреть только максимальное количество подключений.

> **ПОДСКАЗКА**
>
> Текущий пользователь может просматривать свои собственные свойства. Только пользователи с ролью `user_admin` могут просматривать свойства других пользователей.

Синтаксис

```
SHOW PROPERTY [FOR 'user_name'] [LIKE '<property_name>']
```

Параметры

Параметр	Обязательный	Описание
user_name	Нет	Имя пользователя. Если не указано, просматривается свойство текущего пользователя.
property_name	Нет	Имя свойства пользователя.

Примеры

Пример 1: Просмотр свойства текущего пользователя.

```
SHOW PROPERTY;

+----------------------+-------+
| Key                  | Value |
+----------------------+-------+
| max_user_connections | 10000 |
+----------------------+-------+
```

Пример 2: Просмотр свойства пользователя `jack`.

```
SHOW PROPERTY FOR 'jack';
```

Или

```
SHOW PROPERTY FOR 'jack' LIKE 'max_user_connections';
```

```
+----------------------+-------+
| Key                  | Value |
+----------------------+-------+
| max_user_connections | 100   |
+----------------------+-------+
```

См. также

ALTER USER: Устанавливает свойства для пользователя.

SHOW ROLES

SHOW ROLES отображает все роли в системе. Вы можете использовать `SHOW GRANTS FOR ROLE <role_name>;` для просмотра привилегий конкретной роли. Для получения дополнительной информации см. SHOW GRANTS.

Эта команда поддерживается начиная с v1.5.2.

ПОДСКАЗКА

Только пользователи с ролью `user_admin` имеют привилегии для выполнения этой операции.

Синтаксис

```
SHOW ROLES
```

Возвращаемые поля:

Поле	Описание
Name	Имя роли.

Примеры

Отображение всех ролей в системе.

```
mysql> SHOW ROLES;
+---------------+
| Name          |
+---------------+
| root          |
| db_admin      |
| cluster_admin |
| user_admin    |
| public        |
| testrole      |
+---------------+
```

Ссылки

- CREATE ROLE
- ALTER USER
- DROP ROLE

SHOW USERS

SHOW USERS отображает всех пользователей в системе. Пользователи, упоминаемые здесь, являются пользовательскими идентификаторами, а не именами пользователей.

Для получения дополнительной информации о пользовательских идентификаторах см. CREATE USER. Эта команда поддерживается начиная с v1.5.2.

Вы можете использовать `SHOW GRANTS FOR <user_identity>;` для просмотра привилегий конкретного пользователя. Для получения дополнительной информации см. SHOW GRANTS.

> **ПОДСКАЗКА**
>
> Только пользователи с ролью `user_admin` имеют привилегии для выполнения этой операции.

Синтаксис

```
SHOW USERS
```

Возвращаемые поля:

Поле	Описание
User	Идентификатор пользователя.

Примеры

Отображение всех пользователей в системе.

```
mysql> SHOW USERS;
+-----------------+
| User            |
+-----------------+
| 'wybing5'@'%'   |
| 'root'@'%'      |
| 'admin'@'%'     |
| 'star'@'%'      |
| 'wybing_30'@'%' |
| 'simo'@'%'      |
| 'wybing1'@'%'   |
| 'wybing2'@'%'   |
+-----------------+
```

Ссылки

CREATE USER, ALTER USER, DROP USER

3.2 Cluster management

Nodes and processes

ADD BACKEND/COMPUTE NOTE BLACKLIST

Добавляет узел BE или CN в чёрный список BE и CN. Вы можете вручную добавлять узлы BE/CN в чёрный список, чтобы запретить использование этих узлов при выполнении запросов, тем самым избегая частых сбоев запросов или другого неожиданного поведения, вызванного потерей соединения с узлами.

Чёрный список BE поддерживается начиная с версии v1.5.2, а чёрный список CN — начиная с версии v2.0.0. Для получения дополнительной информации см. Управление чёрным списком BE и CN.

> **ПРИМЕЧАНИЕ**
>
> Только пользователи с привилегией BLACKLIST на уровне SYSTEM могут выполнять эту операцию.

По умолчанию Selena может автоматически управлять чёрным списком BE и CN, добавляя узлы BE/CN, потерявшие соединение, в чёрный список и удаляя их из чёрного списка при восстановлении соединения. Однако Selena не удалит узел из чёрного списка, если он был добавлен вручную.

Синтаксис

```
ADD { BACKEND | COMPUTE NODE } BLACKLIST { <be_id>[, ...] | <cn_id>[, ...]
}
```

Параметры

`be_id` или `cn_id`: ID узла BE или CN, который нужно добавить в чёрный список. Вы можете получить ID BE, выполнив SHOW BACKENDS, и ID CN, выполнив SHOW COMPUTE NODES.

Примеры

```
-- Получить ID BE.
SHOW BACKENDS\G
*************************** 1. row ***************************
            BackendId: 10001
                   IP: xxx.xx.xx.xxx
                   ...
-- Добавить BE в чёрный список.
ADD BACKEND BLACKLIST 10001;

-- Получить ID CN.
SHOW COMPUTE NODES\G
*************************** 1. row ***************************
        ComputeNodeId: 10005
                   IP: xxx.xx.xx.xxx
                   ...
-- Добавить CN в чёрный список.
ADD COMPUTE NODE BLACKLIST 10005;
```

Связанные SQL-команды

- DELETE BACKEND/COMPUTE NODE BLACKLIST
- SHOW BACKEND/COMPUTE NODE BLACKLIST
- SHOW BACKENDS
- SHOW COMPUTE NODES

ALTER SYSTEM

ALTER SYSTEM управляет узлами FE, BE, CN, Broker и снимками метаданных в cluster.

> **ПРИМЕЧАНИЕ**
>
> Только роль `cluster_admin` имеет привилегию для выполнения этой операции.

Синтаксис и параметры

FE

- Добавить Follower FE.

```
ALTER SYSTEM ADD FOLLOWER "<fe_host>:<edit_log_port>"[, ...]
```

Вы можете проверить статус нового Follower FE, выполнив `SHOW PROC '/frontends'\G`.

- Удалить Follower FE.

```
ALTER SYSTEM DROP FOLLOWER "<fe_host>:<edit_log_port>"[, ...]
```

- Добавить Observer FE.

```
ALTER SYSTEM ADD OBSERVER "<fe_host>:<edit_log_port>"[, ...]
```

Вы можете проверить статус нового Observer FE, выполнив `SHOW PROC '/frontends'\G`.

- Удалить Observer FE.

```
ALTER SYSTEM DROP OBSERVER "<fe_host>:<edit_log_port>"[, ...]
```

Параметр	Обязательный	Описание
fe_host	Да	Имя хоста или IP-адрес экземпляра FE. Используйте значение параметра конфигурации `priority_networks`, если у вашего экземпляра несколько IP-адресов.
edit_log_port	Да	Порт связи BDB JE узла FE. По умолчанию: `9010`.

BE

- Добавить узел BE.

```
ALTER SYSTEM ADD BACKEND "<be_host>:<heartbeat_service_port>"[, ...]
```

Вы можете проверить статус нового BE, выполнив SHOW BACKENDS.

- Удалить узел BE.

> **ПРИМЕЧАНИЕ**

> Вы не можете удалить узел BE, на котором хранятся tablet таблиц с одной replica.

```
ALTER SYSTEM DROP BACKEND "<be_host>:<heartbeat_service_port>"[, ...]
```

- Вывести из эксплуатации узел BE.

```
ALTER SYSTEM DECOMMISSION BACKEND "<be_host>:<heartbeat_service_port>"
[, ...]
```

В отличие от удаления узла BE, которое принудительно удаляет его из cluster, вывод из эксплуатации означает безопасное удаление. Это асинхронная операция. При выводе из эксплуатации данные на BE сначала мигрируются на другие BE, а затем BE удаляется из cluster. Загрузка данных и запросы не будут затронуты во время миграции данных. Вы можете проверить успешность операции с помощью SHOW BACKENDS. Если операция прошла успешно, выведенный из эксплуатации BE не будет возвращён. Если операция не удалась, BE останется в сети. Вы можете вручную отменить операцию с помощью CANCEL DECOMMISSION.

Параметр	Обязательный	Описание
be_host	Да	Имя хоста или IP-адрес экземпляра BE. Используйте значение параметра конфигурации `priority_networks`, если у вашего экземпляра несколько IP-адресов.
heartbeat_service_port	Да	Порт службы heartbeat BE. BE использует этот порт для получения heartbeat от FE. По умолчанию: `9050`.

CN

- Добавить узел CN.

```
ALTER SYSTEM ADD COMPUTE NODE "<cn_host>:<heartbeat_service_port>"[,
...]
```

Вы можете проверить статус нового CN, выполнив SHOW COMPUTE NODES.

- Удалить узел CN.

```
ALTER SYSTEM DROP COMPUTE NODE "<cn_host>:<heartbeat_service_port>"[,
...]
```

> **ПРИМЕЧАНИЕ**
>
> Вы не можете вывести из эксплуатации узел CN с помощью команды `ALTER SYSTEM DECOMMISSION`.

Параметр	Обязательный	Описание
cn_host	Да	Имя хоста или IP-адрес экземпляра CN. Используйте значение параметра конфигурации `priority_networks`, если у вашего экземпляра несколько IP-адресов.
heartbeat_service_port	Да	Порт службы heartbeat CN. CN использует этот порт для получения heartbeat от FE. По умолчанию: `9050`.

Broker

- Добавить узлы Broker. Вы можете использовать узлы Broker для загрузки данных из HDFS или облачного хранилища в Selena. Для получения дополнительной информации см. Загрузка.

  ```
  ALTER SYSTEM ADD BROKER <broker_name> "<broker_host>:<broker_ipc_port>"
  [, ...]
  ```

  Вы можете добавить несколько узлов Broker одной SQL-командой. Каждая пара `<broker_host>:<broker_ipc_port>` представляет один узел Broker. Они используют общее имя `broker_name`. Вы можете проверить статус нового узла Broker, выполнив SHOW BROKER.

- Удалить узлы Broker.

> **ВНИМАНИЕ**
>
> Удаление узла Broker прерывает текущие задачи, выполняющиеся на нём.

- Удалить один или несколько узлов Broker с одинаковым `broker_name`.

```
ALTER SYSTEM DROP BROKER <broker_name> "<broker_host>:
<broker_ipc_port>"[, ...]
```

- Удалить все узлы Broker с одинаковым `broker_name`.

```
ALTER SYSTEM DROP ALL BROKER <broker_name>
```

Параметр	Обязательный	Описание
broker_name	Да	Имя узла(ов) Broker. Несколько узлов Broker могут использовать одно имя.
broker_host	Да	Имя хоста или IP-адрес экземпляра Broker. Используйте значение параметра конфигурации `priority_networks`, если у вашего экземпляра несколько IP-адресов.
broker_ipc_port	Да	Порт thrift-сервера на узле Broker. Узел Broker использует его для получения запросов от FE или BE. По умолчанию: `8000`.

Создание образа

Создать файл образа. Файл образа — это снимок метаданных FE.

```
ALTER SYSTEM CREATE IMAGE
```

Создание образа — это асинхронная операция на Leader FE. Вы можете проверить время начала и окончания операции в файле журнала FE **fe.log**. Запись типа `triggering a new checkpoint manually...` указывает на начало создания образа, а запись типа `finished save image...` указывает на завершение создания образа.

Примечания по использованию

- Добавление и удаление узлов FE, BE, CN или Broker являются синхронными операциями. Вы не можете отменить операции удаления узлов.
- Вы не можете удалить узел FE в cluster с одним FE.
- Вы не можете напрямую удалить Leader FE в cluster с несколькими FE. Чтобы удалить его, необходимо сначала перезапустить его. После того как Selena выберет нового Leader FE, вы сможете удалить предыдущего.
- Вы не можете удалить узлы BE, если количество оставшихся узлов BE меньше количества replica данных. Например, если в вашем cluster три узла BE и вы храните

данные в трёх replica, вы не можете удалить ни один из узлов BE. А если у вас четыре узла BE и три replica, вы можете удалить один узел BE.

- Разница между удалением и выводом из эксплуатации узла BE заключается в том, что при удалении узла BE Selena принудительно удаляет его из cluster и восстанавливает удалённые tablet после удаления, а при выводе из эксплуатации узла BE Selena сначала мигрирует tablet с выводимого из эксплуатации узла BE на другие, а затем удаляет узел.

Примеры

Пример 1: Добавить Follower FE узел.

```
ALTER SYSTEM ADD FOLLOWER "x.x.x.x:9010";
```

Пример 2: Удалить два Observer FE узла одновременно.

```
ALTER SYSTEM DROP OBSERVER "x.x.x.x:9010","x.x.x.x:9010";
```

Пример 3: Добавить узел BE.

```
ALTER SYSTEM ADD BACKEND "x.x.x.x:9050";
```

Пример 4: Удалить два узла BE одновременно.

```
ALTER SYSTEM DROP BACKEND "x.x.x.x:9050", "x.x.x.x:9050";
```

Пример 5: Вывести из эксплуатации два узла BE одновременно.

```
ALTER SYSTEM DECOMMISSION BACKEND "x.x.x.x:9050", "x.x.x.x:9050";
```

Пример 6: Добавить два узла Broker с одинаковым `broker_name` - `hdfs`.

```
ALTER SYSTEM ADD BROKER hdfs "x.x.x.x:8000", "x.x.x.x:8000";
```

Пример 7: Удалить два узла Broker из `amazon_s3`.

```
ALTER SYSTEM DROP BROKER amazon_s3 "x.x.x.x:8000", "x.x.x.x:8000";
```

Пример 8: Удалить все узлы Broker в `amazon_s3`.

```
ALTER SYSTEM DROP ALL BROKER amazon_s3;
```

CANCEL DECOMMISSION

CANCEL DECOMMISSION используется для отмены вывода узла из эксплуатации.

> **ПОДСКАЗКА**
>
> Только роль `cluster_admin` имеет привилегию для выполнения этой операции. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис:

```
CANCEL DECOMMISSION BACKEND "<host>:<heartbeat_port>"[,"<host>:
<heartbeat_port>"...]
```

Примеры

1. Отменить вывод из эксплуатации двух узлов.

```
CANCEL DECOMMISSION BACKEND "host1:port", "host2:port";
```

DELETE BACKEND/COMPUTE NOT BLACKLIST

Удаляет узел BE/CN из чёрного списка BE. Обратите внимание, что Selena не удаляет узлы BE/CN, которые были вручную добавлены в чёрный список пользователями.

Чёрный список BE поддерживается начиная с версии v1.5.2, а чёрный список CN — начиная с версии v2.0.0. Для получения дополнительной информации см. Управление чёрным списком BE и CN.

> **ПРИМЕЧАНИЕ**
>
> Только пользователи с привилегией BLACKLIST на уровне SYSTEM могут выполнять эту операцию.

Синтаксис

```
DELETE { BACKEND | COMPUTE NODE } BLACKLIST { <be_id>[, ...] | <cn_id>[,
...] }
```

Параметры

`be_id` или `cn_id`: ID узла BE или CN, который нужно удалить из чёрного списка. Вы можете получить ID BE/CN из чёрного списка, выполнив SHOW BACKEND/COMPUTE NODE BLACKLIST.

Примеры

```
DELETE BACKEND BLACKLIST 10001;
```

Связанные SQL-команды

- ADD BACKEND/COMPUTE NODE BLACKLIST
- SHOW BACKEND/COMPUTE NODE BLACKLIST
- SHOW BACKENDS
- SHOW COMPUTE NODES

KILL

Завершает соединение или запрос, выполняемый в данный момент потоками в Selena.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции не требуются привилегии.

Синтаксис

```
KILL [ CONNECTION | QUERY ] { <processlist_id> | <query_id> }
```

Параметры

Параметр	Описание
Модификатор: • CONNECTION • QUERY	• С модификатором `CONNECTION` команда KILL завершает соединение, связанное с указанным `processlist_id`, после завершения любой команды, выполняемой соединением. • С модификатором `QUERY` команда KILL завершает команду, выполняемую в данный момент соединением, но само соединение остаётся активным. • Если модификатор не указан, по умолчанию используется `CONNECTION`.
processlist_id	ID потока, который вы хотите завершить. Вы можете получить ID выполняемых потоков с помощью SHOW PROCESSLIST.
query_id	ID запроса, который вы хотите завершить. Вы можете получить ID ожидающих или выполняемых запросов с помощью SHOW RUNNING QUERIES или SHOW PROC с `'/current_queries'` и `'/global_current_queries'`.

Примеры

```
mysql> SHOW FULL PROCESSLIST;
+------+------+---------------------+--------+---------+-------------------
--+------+-------+-----------------------+-----------+
| Id   | User | Host                | Db     | Command |
ConnectionStartTime | Time | State | Info                  | IsPending |
+------+------+---------------------+--------+---------+-------------------
--+------+-------+-----------------------+-----------+
|   20 | root | xxx.xx.xxx.xx:xxxxx | sr_hub | Query   | 2023-01-05
16:30:19 |    0 | OK    | show full processlist | false     |
+------+------+---------------------+--------+---------+-------------------
--+------+-------+-----------------------+-----------+
1 row in set (0.01 sec)

mysql> KILL 20;
Query OK, 0 rows affected (0.00 sec)
```

SHOW BACKENDS

SHOW BACKENDS отображает информацию обо всех узлах BE в cluster.

ПОДСКАЗКА

Только пользователи с привилегией OPERATE на уровне SYSTEM или ролью `cluster_admin` могут выполнять эту операцию.

Синтаксис

```
SHOW BACKENDS
```

Возвращаемые данные

```
+-----------+--------------+---------------+--------+----------+----------
+---------------------+---------------------+-------+----------------------
+-----------------------+-----------+------------------+---------------+---
------------+---------+----------------+--------+--------------+-----------
---------------------------------------------+-------------------+---------
----+----------+----------+-------------------+------------+------------+--
-------------------------------------------------------+----------+---------
---+
| BackendId | IP           | HeartbeatPort | BePort | HttpPort | BrpcPort |
LastStartTime       | LastHeartbeat       | Alive | SystemDecommissioned |
ClusterDecommissioned | TabletNum | DataUsedCapacity | AvailCapacity |
TotalCapacity | UsedPct | MaxDiskUsedPct | ErrMsg | Version      | Status
| DataTotalCapacity | DataUsedPct | CpuCores | MemLimit | NumRunningQueries
| MemUsedPct | CpuUsedPct | DataCacheMetrics
| Location | StatusCode |
+-----------+--------------+---------------+--------+----------+----------
+---------------------+---------------------+-------+----------------------
+-----------------------+-----------+------------------+---------------+---
------------+---------+----------------+--------+--------------+-----------
---------------------------------------------+-------------------+---------
----+----------+----------+-------------------+------------+------------+--
-------------------------------------------------------+----------+---------
---+
```

Возвращаемое значение	Описание
BackendId	ID узла BE.
IP	IP-адрес узла BE.
HeartbeatPort	Порт heartbeat узла BE. Используется для получения heartbeat от узла FE.
BePort	Порт Thrift-сервера узла BE. Используется для получения запросов от узла FE.

Возвращаемое значение	Описание
HttpPort	Порт HTTP-сервера узла BE. Используется для доступа к узлу BE через веб-страницу.
BrpcPort	Порт bRPC узла BE. Используется для связи между узлами BE.
LastStartTime	Время последнего запуска узла BE.
LastHeartbeat	Время последнего heartbeat, отправленного узлом FE и подтверждённого BE.
Alive	Активен ли узел BE. • `true`: узел BE активен. • `false`: узел BE неактивен.
SystemDecommissioned	Выведен ли узел BE из эксплуатации. Перед выводом из эксплуатации узел BE мигрирует данные на другие узлы BE. Загрузка данных и запросы не затрагиваются во время миграции. • `true`: cluster отметил узел BE как выведенный из эксплуатации. Данные успешно перенесены или переносятся. • `false`: узел BE работает.
ClusterDecommissioned	Этот параметр используется для совместимости системы.
TabletNum	Количество tablet на узле BE.
DataUsedCapacity	Объём хранилища, занятый файлами данных.
AvailCapacity	Доступный объём хранилища на узле BE.
TotalCapacity	Общий объём хранилища. Эквивалентно `DataUsedCapacity` + `AvailCapacity` + Объём хранилища, занятый файлами, не являющимися данными.
UsedPct	Процент использованного объёма хранилища.
MaxDiskUsedPct	Если узел BE имеет несколько каталогов для хранения данных, этот параметр показывает максимальный процент использования хранилища среди всех каталогов.
ErrMsg	Сообщение об ошибке, возвращаемое, когда узел BE не получает heartbeat от узла FE.
Version	Версия Selena cluster.
Status	Время последнего отчёта о количестве tablet, отправленного узлом BE в узел FE. Отображается в формате JSON.
DataTotalCapacity	Эквивалентно `DataUsedCapacity` + `AvailCapacity`. Указывает сумму объёма хранилища, занятого файлами данных, и доступного объёма хранилища на узле BE.
DataUsedPct	Эквивалентно `DataUsedCapacity` / `DataTotalCapacity`. Указывает долю объёма хранилища, занятого файлами данных, к сумме объёма, занятого данными, и доступного объёма хранилища.
CpuCores	Количество ядер CPU на узле BE.
MemLimit	Лимит памяти узла BE.
NumRunningQueries	Количество выполняемых запросов на узле BE.

Возвращаемое значение	Описание
MemUsedPct	Процент использованной памяти.
CpuUsedPct	Процент использованных ядер CPU.
DataCacheMetrics	Статус Data Cache: • `Disabled`: Отключён • `Normal`: Нормальный • `Abnormal`: Аномальный • `Updating`: Data Cache обновляется, например, масштабируется.
Location	Метка узла BE.
StatusCode	Код статуса BE. Допустимые значения: • `CONNECTING`: Узел BE впервые добавляется в cluster. • `OK`: Узел BE работает. • `SHUTDOWN`: Узел BE корректно завершает работу. • `DISCONNECTED`: Узел BE отключён.

Пример

Просмотр информации обо всех узлах BE в cluster.

```
SHOW BACKENDS;

+-----------+---------+---------------+--------+----------+----------+-----
----------------+---------------------+-------+----------------------+-----
------------------+-----------+------------------+---------------+---------
------+---------+----------------+--------+----------------------+---------
-----------------------------------------------+-------------------+-------
------+----------+
| BackendId | IP      | HeartbeatPort | BePort | HttpPort | BrpcPort |
LastStartTime       | LastHeartbeat       | Alive | SystemDecommissioned |
ClusterDecommissioned | TabletNum | DataUsedCapacity | AvailCapacity |
TotalCapacity | UsedPct | MaxDiskUsedPct | ErrMsg | Version              |
Status                                                | DataTotalCapacity
| DataUsedPct | CpuCores |
+-----------+---------+---------------+--------+----------+----------+-----
----------------+---------------------+-------+----------------------+-----
------------------+-----------+------------------+---------------+---------
------+---------+----------------+--------+----------------------+---------
-----------------------------------------------+-------------------+-------
------+----------+
| 10002     | x.x.x.x | 9254          | 9260   | 8238     | 8260     |
2022-09-08 14:37:10 | 2022-09-08 15:14:21 | true  | false                |
false                 | 21753     | 25.122 GB        | 1.088 TB      |
1.968 TB      | 44.72 % | 44.72 %        |        | MAIN-RELEASE-d094052 |
{"lastSuccessReportTabletsTime":"2022-09-08 15:14:13"} | 1.113 TB
| 2.20 %      | 16       |
| 10003     | x.x.x.x | 9254          | 9260   | 8238     | 8260     |
2022-09-08 14:37:20 | 2022-09-08 15:14:21 | true  | false                |
false                 | 21754     | 25.121 GB        | 1.169 TB      |
1.968 TB      | 40.61 % | 40.61 %        |        | MAIN-RELEASE-d094052 |
{"lastSuccessReportTabletsTime":"2022-09-08 15:14:22"} | 1.194 TB
| 2.06 %      | 16       |
| 10004     | x.x.x.x | 9254          | 9260   | 8238     | 8260     |
2022-09-08 14:37:39 | 2022-09-08 15:14:21 | true  | false                |
false                 | 21754     | 25.112 GB        | 1.172 TB      |
1.968 TB      | 40.46 % | 40.46 %        |        | MAIN-RELEASE-d094052 |
{"lastSuccessReportTabletsTime":"2022-09-08 15:13:42"} | 1.197 TB
| 2.05 %      | 16       |
+-----------+---------+---------------+--------+----------+----------+-----
----------------+---------------------+-------+----------------------+-----
------------------+-----------+------------------+---------------+---------
------+---------+----------------+--------+----------------------+---------
-----------------------------------------------+-------------------+-------
------+----------+
```

SHOW BACKEND/COMPUTE NODE BLACKLIST

Отображает узлы BE/CN в чёрном списке BE и CN.

Чёрный список BE поддерживается начиная с версии v1.5.2, а чёрный список CN — начиная с версии v2.0.0. Для получения дополнительной информации см. Управление чёрным списком BE и CN.

> **ПРИМЕЧАНИЕ**
>
> Только пользователи с привилегией BLACKLIST на уровне SYSTEM могут выполнять эту операцию.

Синтаксис

```
SHOW { BACKEND | COMPUTE NODE } BLACKLIST
```

Возвращаемое значение

Возвращаемое значение	Описание
AddBlackListType	Способ добавления узла BE/CN в чёрный список. `MANUAL` указывает, что узел был добавлен вручную пользователем. `AUTO` указывает, что узел был автоматически добавлен Selena.
LostConnectionTime	Для типа `MANUAL` указывает время, когда узел BE/CN был вручную добавлен в чёрный список. Для типа `AUTO` указывает время последнего успешного подключения.
LostConnectionNumberInPeriod	Количество отключений, обнаруженных в течение `CheckTimePeriod(s)`.
CheckTimePeriod(s)	Интервал, с которым Selena проверяет статус подключения узлов BE/CN из чёрного списка. Значение соответствует значению, указанному для параметра конфигурации FE `black_host_history_sec`. Единица измерения: секунды.

Примеры

```
-- Просмотр чёрного списка BE.
SHOW BACKEND BLACKLIST;
+-----------+-----------------+--------------------+--------------------
---------+--------------------+
| BackendId | AddBlackListType | LostConnectionTime  |
LostConnectionNumberInPeriod | CheckTimePeriod(s) |
+-----------+-----------------+--------------------+--------------------
---------+--------------------+
| 10001     | MANUAL           | 2024-04-28 11:52:09 | 0
| 5                  |
+-----------+-----------------+--------------------+--------------------
---------+--------------------+

-- Просмотр чёрного списка CN.
SHOW COMPUTE NODE BLACKLIST;
+---------------+-----------------+--------------------+----------------
-------------+--------------------+
| ComputeNodeId | AddBlackListType | LostConnectionTime  |
LostConnectionNumberInPeriod | CheckTimePeriod(s) |
+---------------+-----------------+--------------------+----------------
-------------+--------------------+
| 10005         | MANUAL           | 2025-08-18 10:47:51 | 0
| 5                  |
+---------------+-----------------+--------------------+----------------
-------------+--------------------+
```

Связанные SQL-команды

- ADD BACKEND/COMPUTE NODE BLACKLIST
- DELETE BACKEND/COMPUTE NODE BLACKLIST
- SHOW BACKENDS
- SHOW COMPUTE NODES

SHOW BROKER

SHOW BROKER используется для просмотра существующих брокеров.

> **ПОДСКАЗКА**
>
> Только пользователи с привилегией OPERATE на уровне SYSTEM или ролью `cluster_admin` могут выполнять эту операцию.

Синтаксис

```
SHOW BROKER
```

Примечание:

1. LastStartTime представляет время последнего запуска BE.
2. LastHeartbeat представляет последний heartbeat.
3. Alive указывает, активен ли узел.
4. ErrMsg используется для отображения сообщений об ошибках при сбое heartbeat.

SHOW COMPUTE NODES

SHOW COMPUTE NODES отображает информацию обо всех узлах CN в cluster.

> **ПОДСКАЗКА**
>
> Только пользователи с привилегией OPERATE на уровне SYSTEM или ролью `cluster_admin` могут выполнять эту операцию.

Синтаксис

```
SHOW COMPUTE NODES
```

Возвращаемые данные

```
+---------------+--------------+---------------+--------+----------+-------
---+---------------------+---------------------+-------+------------------
---+-----------------------+--------+--------------------+----------+------
-------------+------------+------------+----------------+-------------+----
------+-------------------+-----------+------------+
| ComputeNodeId | IP           | HeartbeatPort | BePort | HttpPort |
BrpcPort | LastStartTime       | LastHeartbeat       | Alive |
SystemDecommissioned | ClusterDecommissioned | ErrMsg | Version
| CpuCores | NumRunningQueries | MemUsedPct | CpuUsedPct | HasStoragePath |
StarletPort | WorkerId | WarehouseName     | TabletNum | StatusCode |
+---------------+--------------+---------------+--------+----------+-------
---+---------------------+---------------------+-------+------------------
---+-----------------------+--------+--------------------+----------+------
-------------+------------+------------+----------------+-------------+----
------+-------------------+-----------+------------+
```

В следующей таблице описаны параметры, возвращаемые этой командой.

Параметр	Описание
ComputeNodeId	ID узла CN.
IP	IP-адрес узла CN.
HeartbeatPort	Порт heartbeat узла CN. Используется для получения heartbeat от узла FE.
BePort	Порт Thrift-сервера узла CN. Используется для получения запросов от узла FE.
HttpPort	Порт HTTP-сервера узла CN. Используется для доступа к узлу CN через веб-страницу.
BrpcPort	Порт bRPC узла CN. Используется для связи между узлами CN.
LastStartTime	Время последнего запуска узла CN.
LastHeartbeat	Время последнего heartbeat узла CN.
Alive	Активен ли узел CN. • `true`: узел CN активен. • `false`: узел CN неактивен.
SystemDecommissioned	Если значение этого параметра `true`, узел CN удалён из вашего cluster Selena.
ClusterDecommissioned	Этот параметр используется для совместимости системы.
ErrMsg	Сообщение об ошибке, если узел CN не смог отправить heartbeat.
Version	Версия Selena узла CN.
CpuCores	Количество ядер CPU на узле CN.

Параметр	Описание
NumRunningQueries	Количество выполняемых запросов на узле CN.
MemUsedPct	Процент использованной памяти.
CpuUsedPct	Процент использованных ядер CPU.
HasStoragePath	Настроены ли пути хранения на узле CN.
StarletPort	Порт `starlet_port` узла CN. Это дополнительный порт агентской службы.
WorkerId	ID узла CN для внутреннего планирования.
WarehouseName	Название хранилища, которому принадлежит узел CN. Значение всегда `default_warehouse`.
TabletNum	Количество tablet (кэшированных данных) на узле CN.
StatusCode	Код статуса CN. Допустимые значения: • `CONNECTING`: Узел CN впервые добавляется в cluster. • `OK`: Узел CN работает. • `SHUTDOWN`: Узел CN корректно завершает работу. • `DISCONNECTED`: Узел CN отключён.

Пример

Просмотр информации обо всех узлах CN в cluster.

```
MySQL > SHOW COMPUTE NODES\G
*************************** 1. row ***************************
        ComputeNodeId: 10001
                   IP: x.x.x.x
        HeartbeatPort: 9050
               BePort: 9060
             HttpPort: 8040
             BrpcPort: 8060
        LastStartTime: 2024-05-14 15:45:34
        LastHeartbeat: 2024-05-14 15:47:59
                Alive: true
 SystemDecommissioned: false
ClusterDecommissioned: false
               ErrMsg:
              Version: 3.3.0-rc01-3b8cb0c
             CpuCores: 4
    NumRunningQueries: 0
           MemUsedPct: 1.95 %
           CpuUsedPct: 0.0 %
       HasStoragePath: true
          StarletPort: 8167
             WorkerId: 1
        WarehouseName: default_warehouse
            TabletNum: 58
1 row in set (0.00 sec)
```

SHOW FRONTENDS

SHOW FRONTENDS используется для просмотра узлов FE.

> **ПОДСКАЗКА**
>
> Только пользователи с привилегией OPERATE на уровне SYSTEM или ролью `cluster_admin` могут выполнять эту операцию.

Синтаксис

```
SHOW FRONTENDS
```

Примечание:

1. Name представляет имя узла FE в BDBJE.
2. Значение Join равное true означает, что этот узел уже присоединился к cluster. Однако это не означает, что узел всё ещё находится в cluster, так как он может отсутствовать.
3. Alive указывает, активен ли узел.
4. ReplayedJournalId представляет максимальный ID журнала метаданных, который узел в настоящее время воспроизвёл.
5. LastHeartbeat — последний heartbeat.
6. IsHelper указывает, является ли узел вспомогательным узлом из BDBJE.
7. ErrMsg используется для отображения сообщений об ошибках при сбое heartbeat.

SHOW PROC

SHOW PROC отображает определённые показатели cluster Selena.

SHOW PROC эквивалентен файловой системе `/proc` в Linux. Подобно путям в `/proc`, параметры в SHOW PROC имеют древовидную иерархическую структуру, и каждый уровень отображает различную информацию. Возьмём `SHOW PROC '/dbs'` в качестве примера:

- `SHOW PROC '/dbs'` вернёт информацию обо всех базах данных в cluster.
- `SHOW PROC '/dbs/<db_name>'` вернёт информацию об указанной базе данных.
- `SHOW PROC '/dbs/<db_name>/tables'` вернёт информацию обо всех таблицах в указанной базе данных.
- `SHOW PROC '/dbs/<db_name>/<table_name>'` вернёт информацию об указанной таблице.
- `SHOW PROC '/dbs/<db_name>/<table_name>/partitions'` вернёт информацию обо всех partition в указанной таблице.

Можно указать путь вплоть до конечного узла.

ПОДСКАЗКА

Для выполнения этой операции требуется привилегия OPERATE на уровне SYSTEM. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис

```
SHOW PROC { '/backends' | '/compute_nodes' | '/dbs' | '/jobs'
          | '/statistic' | '/tasks' | '/frontends' | '/brokers'
          | '/resources' | '/load_error_hub' | '/transactions'
          | '/monitor' | '/current_queries' | '/current_backend_instances'
          | '/cluster_balance' | '/routine_loads' | '/colocation_group'
          | '/catalog' | '/replications'  | '/global_current_queries' }
```

Параметры

Параметр	Описание
'/backends'	Отображает информацию об узлах BE в cluster.
'/compute_nodes'	Отображает информацию об узлах CN в cluster.
'/dbs'	Отображает информацию о базах данных, таблицах, partition и tablet в cluster.
'/jobs'	Отображает информацию о заданиях в cluster.
'/statistic'	Отображает статистику по каждой базе данных в cluster.
'/tasks'	Отображает общее количество всех общих задач и неудачных задач в cluster.
'/frontends'	Отображает информацию об узлах FE в cluster.
'/brokers'	Отображает информацию об узлах Broker в cluster.
'/resources'	Отображает информацию о ресурсах в cluster.
'/load_error_hub'	Отображает конфигурацию Load Error Hub cluster, который используется для управления сообщениями об ошибках заданий загрузки.
'/transactions'	Отображает информацию о транзакциях в cluster.
'/monitor'	Отображает информацию мониторинга в cluster.
'/current_queries'	Отображает информацию о выполняемых запросах на текущем узле FE.
'/current_backend_instances'	Отображает узлы BE, обрабатывающие запросы в cluster.
'/cluster_balance'	Отображает информацию о балансировке нагрузки в cluster.
'/routine_loads'	Отображает информацию о Routine Load в cluster.
'/colocation_group'	Отображает информацию о группах Colocate JOIN в cluster.
'/catalog'	Отображает информацию о каталогах в cluster.
'/replications'	Отображает информацию о задачах репликации данных в cluster.
'/global_current_queries'	Отображает информацию о выполняемых запросах на всех узлах FE в cluster.

Примеры

Пример 1: Отображает информацию об узлах BE в cluster.

```
mysql> SHOW PROC '/backends'\G
*************************** 1. row ***************************
            BackendId: 10004
                   IP: xxx.xx.92.200
        HeartbeatPort: 9354
               BePort: 9360
             HttpPort: 8338
             BrpcPort: 8360
        LastStartTime: 2023-04-21 09:56:10
        LastHeartbeat: 2023-04-21 09:56:10
                Alive: true
 SystemDecommissioned: false
ClusterDecommissioned: false
            TabletNum: 2199
     DataUsedCapacity: 0.000
        AvailCapacity: 584.578 GB
        TotalCapacity: 1.968 TB
              UsedPct: 71.00 %
       MaxDiskUsedPct: 71.00 %
               ErrMsg:
              Version: BRANCH-3.0-RELEASE-8eb8705
               Status: {"lastSuccessReportTabletsTime":"N/A"}
    DataTotalCapacity: 584.578 GB
          DataUsedPct: 0.00 %
             CpuCores: 16
    NumRunningQueries: 0
           MemUsedPct: 0.52 %
           CpuUsedPct: 0.0 %
```

Возвращаемое значение	Описание
BackendId	ID узла BE.
IP	IP-адрес узла BE.
HeartbeatPort	Порт heartbeat узла BE.
BePort	Порт Thrift Server узла BE.
HttpPort	Порт HTTP Server узла BE.
BrpcPort	Порт bRPC узла BE.
LastStartTime	Время последнего запуска узла BE.
LastHeartbeat	Время последнего heartbeat узла BE.
Alive	Активен ли узел BE.

Возвращаемое значение	Описание
SystemDecommissioned	Выведен ли узел BE из эксплуатации.
ClusterDecommissioned	Выведен ли узел BE из эксплуатации в рамках cluster.
TabletNum	Количество tablet на узле BE.
DataUsedCapacity	Ёмкость хранилища, используемая для данных на узле BE.
AvailCapacity	Доступная ёмкость хранилища на узле BE.
TotalCapacity	Общая ёмкость хранилища на узле BE.
UsedPct	Процент использования ёмкости хранилища на узле BE.
MaxDiskUsedPct	Максимальный процент использования ёмкости хранилища на узле BE.
ErrMsg	Сообщения об ошибках на узле BE.
Version	Версия Selena узла BE.
Status	Информация о статусе узла BE, включая время последнего отчёта о tablet.
DataTotalCapacity	Сумма использованной и доступной ёмкости хранилища данных. Сумма `DataUsedCapacity` и `AvailCapacity`.
DataUsedPct	Процент, который занимает хранилище данных от общей ёмкости данных (DataUsedCapacity/DataTotalCapacity).
CpuCores	Количество ядер CPU на узле BE.
NumRunningQueries	Количество запросов, выполняемых в данный момент в cluster.
MemUsedPct	Текущий процент использования памяти.
CpuUsedPct	Текущий процент использования CPU.

Пример 2: Отображает информацию о базах данных в cluster.

```
mysql> show proc "/dbs";
+-------+--------------------+----------+----------------+-----------------
---------+---------------------+
| DbId  | DbName             | TableNum | Quota          |
LastConsistencyCheckTime | ReplicaQuota        |
+-------+--------------------+----------+----------------+-----------------
---------+---------------------+
| 1     | information_schema | 54       | 8388608.000 TB | NULL
| 9223372036854775807 |
| 100   | sys                | 6        | 8388608.000 TB | NULL
| 9223372036854775807 |
| 10001 | _statistics_       | 12       | 8388608.000 TB | NULL
| 9223372036854775807 |
| 10015 | test               | 1        | 8388608.000 TB | NULL
| 9223372036854775807 |
+-------+--------------------+----------+----------------+-----------------
---------+---------------------+
4 rows in set (0.00 sec)
mysql> show proc "/dbs/test";
+---------+------------------+----------+---------------------+------------
--+--------+--------------+--------------------------+--------------+------
---------+-------------------------------------------------------------------
--------------------------------------+
| TableId | TableName        | IndexNum | PartitionColumnName |
PartitionNum | State  | Type         | LastConsistencyCheckTime |
ReplicaCount | PartitionType | StoragePath
|
+---------+------------------+----------+---------------------+------------
--+--------+--------------+--------------------------+--------------+------
---------+-------------------------------------------------------------------
--------------------------------------+
| 10207   | source_wiki_edit | 1        | event_time          | 4
| NORMAL | CLOUD_NATIVE | NULL                     | 8            | RANGE
| s3://test/xxx
|
+---------+------------------+----------+---------------------+------------
--+--------+--------------+--------------------------+--------------+------
---------+-------------------------------------------------------------------
--------------------------------------+
1 row in set (0.00 sec)
mysql> show proc "/dbs/test/source_wiki_edit";
+-----------------+
| Nodes           |
+-----------------+
| partitions      |
```

```
| temp_partitions |
| index_schema    |
+-----------------+
3 rows in set (0.00 sec)
```

Возвращаемое значение	Описание
DbId	ID базы данных.
DbName	Название базы данных.
TableNum	Количество таблиц в базе данных.
Quota	Квота хранилища базы данных.
LastConsistencyCheckTime	Время последней проверки согласованности.
ReplicaQuota	Квота replica данных базы данных.
TableId	ID таблицы.
TableName	Название таблицы.
PartitionColumnName	Название столбца partition.
PartitionNum	Количество partition в таблице.
State	Состояние таблицы.
Type	Тип таблицы.
LastConsistencyCheckTime	Время последней проверки согласованности.
ReplicaCount	Количество replica для таблицы.
PartitionType	Тип partition в таблице.
StoragePath	Путь хранения таблицы.
partitions	Partition в таблице.
temp_partitions	Временные partition в таблице.
index_schema	Синхронное материализованное представление в таблице.

Пример 3: Отображает информацию о заданиях в cluster.

```
mysql> SHOW PROC '/jobs';
+-------+-------------------------------------+
| DbId  | DbName                              |
+-------+-------------------------------------+
| 10005 | default_cluster:_statistics_        |
| 0     | default_cluster:information_schema  |
| 12711 | default_cluster:selena_audit_db__ |
+-------+-------------------------------------+
3 rows in set (0.00 sec)

mysql> SHOW PROC '/jobs/10005';
+---------------+---------+---------+----------+-----------+-------+
| JobType       | Pending | Running | Finished | Cancelled | Total |
+---------------+---------+---------+----------+-----------+-------+
| load          | 0       | 0       | 3        | 0         | 3     |
| rollup        | 0       | 0       | 0        | 0         | 0     |
| schema_change | 0       | 0       | 0        | 0         | 0     |
| export        | 0       | 0       | 0        | 0         | 0     |
+---------------+---------+---------+----------+-----------+-------+
4 rows in set (0.00 sec)
```

Возвращаемое значение	Описание
DbId	ID базы данных.
DbName	Название базы данных.
JobType	Тип задания.
Pending	Количество ожидающих заданий.
Running	Количество выполняемых заданий.
Finished	Количество завершённых заданий.
Cancelled	Количество отменённых заданий.
Total	Общее количество заданий.

Пример 4: Отображает статистику по каждой базе данных в cluster.

```
mysql> SHOW PROC '/statistic';
+--------+----------------------------------------------------------------+------
----+--------------+----------+-----------+------------+-------------------
-+-----------------------+------------------+---------------------+
| DbId   | DbName                                                         |
TableNum | PartitionNum | IndexNum | TabletNum | ReplicaNum |
UnhealthyTabletNum | InconsistentTabletNum | CloningTabletNum |
ErrorStateTabletNum |
+--------+----------------------------------------------------------------+------
----+--------------+----------+-----------+------------+-------------------
-+-----------------------+------------------+---------------------+
| 10004  | _statistics_                                                   | 3
| 3            | 3        | 30        | 60         | 0                  | 0
| 0                | 0                   |
| 1      | information_schema                                             | 0
| 0            | 0        | 0         | 0          | 0                  | 0
| 0                | 0                   |
| 92498  | stream_load_test_db_03afc714_b1cb_11ed_a82c_00163e237e98 | 0
| 0            | 0        | 0         | 0          | 0                  | 0
| 0                | 0                   |
| 92542  | stream_load_test_db_79876e92_b1da_11ed_b50e_00163e237e98 | 1
| 1            | 1        | 3         | 3          | 0                  | 0
| 0                | 0                   |
| 115476 | testdb                                                         | 0
| 0            | 0        | 0         | 0          | 0                  | 0
| 0                | 0                   |
| 10002  | zq_test                                                        | 8
| 8            | 8        | 5043      | 7063       | 0                  | 0
| 0                | 2                   |
| Total  | 6                                                              | 12
| 12           | 12       | 5076      | 7126       | 0                  | 0
| 0                | 2                   |
+--------+----------------------------------------------------------------+------
----+--------------+----------+-----------+------------+-------------------
-+-----------------------+------------------+---------------------+
7 rows in set (0.01 sec)

mysql> show proc '/statistic/10002';
+------------------+---------------------+----------------+----------------
---+
| UnhealthyTablets | InconsistentTablets | CloningTablets |
ErrorStateTablets |
+------------------+---------------------+----------------+----------------
---+
| []               | []                  | []             | [116703,
```

```
116706]  |
+-----------------+--------------------+---------------+---------------
---+
```

Возвращаемое значение	Описание
DbId	ID базы данных.
DbName	Название базы данных.
TableNum	Количество таблиц в базе данных.
PartitionNum	Количество partition в базе данных.
IndexNum	Количество индексов в базе данных.
TabletNum	Количество tablet в базе данных.
ReplicaNum	Количество replica в базе данных.
UnhealthyTabletNum	Количество незавершённых (нездоровых) tablet в базе данных во время перераспределения данных.
InconsistentTabletNum	Количество несогласованных tablet в базе данных.
CloningTabletNum	Количество tablet, которые клонируются в базе данных.
ErrorStateTabletNum	В таблице типа Primary Key — количество tablet в состоянии Error.
ErrorStateTablets	В таблице типа Primary Key — ID tablet в состоянии Error.

Пример 5: Отображает общее количество всех общих задач и неудачных задач в cluster.

```
mysql> SHOW PROC '/tasks';
+-------------------------+-----------+----------+
| TaskType                | FailedNum | TotalNum |
+-------------------------+-----------+----------+
| CREATE                  | 0         | 0        |
| DROP                    | 0         | 0        |
| PUSH                    | 0         | 0        |
| CLONE                   | 0         | 0        |
| STORAGE_MEDIUM_MIGRATE  | 0         | 0        |
| ROLLUP                  | 0         | 0        |
| SCHEMA_CHANGE           | 0         | 0        |
| CANCEL_DELETE           | 0         | 0        |
| MAKE_SNAPSHOT           | 0         | 0        |
| RELEASE_SNAPSHOT        | 0         | 0        |
| CHECK_CONSISTENCY       | 0         | 0        |
| UPLOAD                  | 0         | 0        |
| DOWNLOAD                | 0         | 0        |
| CLEAR_REMOTE_FILE       | 0         | 0        |
| MOVE                    | 0         | 0        |
| REALTIME_PUSH           | 0         | 0        |
| PUBLISH_VERSION         | 0         | 0        |
| CLEAR_ALTER_TASK        | 0         | 0        |
| CLEAR_TRANSACTION_TASK  | 0         | 0        |
| RECOVER_TABLET          | 0         | 0        |
| STREAM_LOAD             | 0         | 0        |
| UPDATE_TABLET_META_INFO | 0         | 0        |
| ALTER                   | 0         | 0        |
| INSTALL_PLUGIN          | 0         | 0        |
| UNINSTALL_PLUGIN        | 0         | 0        |
| NUM_TASK_TYPE           | 0         | 0        |
| Total                   | 0         | 0        |
+-------------------------+-----------+----------+
```

Возвращаемое значение	Описание
TaskType	Тип задачи.
FailedNum	Количество неудачных задач.
TotalNum	Общее количество задач.

Пример 6: Отображает информацию об узлах FE в cluster.

```
mysql> SHOW PROC '/frontends';
+-----------------------------------+---------------+-------------+---------
-+-----------+---------+----------+------------+-------+-------+-----------
--------+---------------+----------+---------------+-----------+---------+
| Name                              | IP            | EditLogPort | HttpPort
| QueryPort | RpcPort | Role     | ClusterId  | Join  | Alive |
ReplayedJournalId | LastHeartbeat | IsHelper | ErrMsg        | StartTime |
Version |
+-----------------------------------+---------------+-------------+---------
-+-----------+---------+----------+------------+-------+-------+-----------
--------+---------------+----------+---------------+-----------+---------+
| xxx.xx.xx.xxx_9009_1600088918395 | xxx.xx.xx.xxx | 9009        | 7390
| 0         | 0       | FOLLOWER | 1747363037 | false | false | 0
| NULL          | true     | got exception | NULL      | NULL    |
+-----------------------------------+---------------+-------------+---------
-+-----------+---------+----------+------------+-------+-------+-----------
--------+---------------+----------+---------------+-----------+---------+
```

Возвращаемое значение	Описание
Name	Имя узла FE.
IP	IP-адрес узла FE.
EditLogPort	Порт для связи между узлами FE.
HttpPort	Порт HTTP Server узла FE.
QueryPort	Порт MySQL Server узла FE.
RpcPort	Порт RPC узла FE.
Role	Роль узла FE (Leader, Follower или Observer).
ClusterId	ID cluster.
Join	Присоединился ли узел FE к cluster.
Alive	Активен ли узел FE.
ReplayedJournalId	Максимальный ID метаданных, который узел FE воспроизвёл.
LastHeartbeat	Время последнего heartbeat узла FE.
IsHelper	Является ли узел FE вспомогательным узлом BDBJE.
ErrMsg	Сообщения об ошибках на узле FE.
StartTime	Время запуска узла FE.
Version	Версия Selena узла FE.

Пример 7: Отображает информацию об узлах Broker в cluster.

```
mysql> SHOW PROC '/brokers';
+-------------+---------------+------+-------+---------------+-------------
--------+--------+
| Name        | IP            | Port | Alive | LastStartTime |
LastUpdateTime      | ErrMsg |
+-------------+---------------+------+-------+---------------+-------------
--------+--------+
| hdfs_broker | xxx.xx.xx.xxx | 8500 | true  | NULL          | 2022-10-10
16:37:59 |        |
| hdfs_broker | xxx.xx.xx.xxx | 8500 | true  | NULL          | 2022-10-10
16:37:59 |        |
| hdfs_broker | xxx.xx.xx.xxx | 8500 | true  | NULL          | 2022-10-10
16:37:59 |        |
+-------------+---------------+------+-------+---------------+-------------
--------+--------+
```

Возвращаемое значение	Описание
Name	Имя узла Broker.
IP	IP-адрес узла Broker.
Port	Порт Thrift Server узла Broker. Порт используется для получения запросов.
Alive	Активен ли узел Broker.
LastStartTime	Время последнего запуска узла Broker.
LastUpdateTime	Время последнего обновления узла Broker.
ErrMsg	Сообщение об ошибке на узле Broker.

Пример 8: Отображает информацию о ресурсах в cluster.

```
mysql> SHOW PROC '/resources';
+-------------------------+--------------+---------------------+-----------
-------------------+
| Name                    | ResourceType | Key                 | Value
|
+-------------------------+--------------+---------------------+-----------
-------------------+
| hive_resource_stability | hive         | hive.metastore.uris |
thrift://xxx.xx.xxx.xxx:9083 |
| hive2                   | hive         | hive.metastore.uris |
thrift://xxx.xx.xx.xxx:9083  |
+-------------------------+--------------+---------------------+-----------
-------------------+
```

Возвращаемое значение	Описание
Name	Название ресурса.
ResourceType	Тип ресурса.
Key	Ключ ресурса.
Value	Значение ресурса.

Пример 9: Отображает информацию о транзакциях в cluster.

```
mysql> SHOW PROC '/transactions';
+-------+---------------------------------------+
| DbId  | DbName                                |
+-------+---------------------------------------+
| 10005 | default_cluster:_statistics_          |
| 12711 | default_cluster:selena_audit_db__ |
+-------+---------------------------------------+
2 rows in set (0.00 sec)

mysql> SHOW PROC '/transactions/10005';
+----------+--------+
| State    | Number |
+----------+--------+
| running  | 0      |
| finished | 4      |
+----------+--------+
2 rows in set (0.00 sec)
```

Возвращаемое значение	Описание
DbId	ID базы данных.
DbName	Название базы данных.
State	Состояние транзакции.
Number	Количество транзакций.

Пример 10: Отображает информацию мониторинга в cluster.

```
mysql> SHOW PROC '/monitor';
+------+------+
| Name | Info |
+------+------+
| jvm  |      |
+------+------+
```

Возвращаемое значение	Описание
Name	Имя JVM.
Info	Информация JVM.

Пример 11: Отображает информацию о балансировке нагрузки в cluster.

```
mysql> SHOW PROC '/cluster_balance';
+--------------------+--------+
| Item               | Number |
+--------------------+--------+
| balance_stat       | 1      |
| cluster_load_stat  | 1      |
| working_slots      | 3      |
| sched_stat         | 1      |
| priority_repair    | 0      |
| pending_tablets    | 2001   |
| running_tablets    | 0      |
| history_tablets    | 1000   |
+--------------------+--------+
```

Возвращаемое значение	Описание
Item	Подкоманда в `cluster_balance`. • balance_stat: Текущее состояние балансировки cluster. • cluster_load_stat: Текущее состояние нагрузки cluster. • working_slots: Количество доступных в данный момент рабочих слотов. • sched_stat: Текущее состояние планировщика. • priority_repair: Количество приоритетных задач по восстановлению tablet. • pending_tablets: Количество tablet, ожидающих обработки. • running_tablets: Количество tablet, которые в данный момент восстанавливаются. • history_tablets: Общее количество исторически восстановленных tablet.
Number	Количество каждой подкоманды в `cluster_balance`.

```
mysql> SHOW PROC '/cluster_balance/balance_stat';
+---------------+--------------------------------+----------+--------------
--+----------------+
| StorageMedium | BalanceType                    | Balanced |
PendingTablets | RunningTablets |
+---------------+--------------------------------+----------+--------------
--+----------------+
| HDD           | inter-node disk usage          | true     | 0
| 0              |
| HDD           | inter-node tablet distribution | true     | 0
| 0              |
| HDD           | intra-node disk usage          | true     | 0
| 0              |
| HDD           | intra-node tablet distribution | true     | 0
| 0              |
| HDD           | colocation group               | true     | 0
| 0              |
| HDD           | label-aware location           | true     | 0
| 0              |
+---------------+--------------------------------+----------+--------------
--+----------------+
```

Возвращаемое значение	Описание
StorageMedium	Носитель хранения.
BalanceType	Тип балансировки.
Balanced	Достигнуто ли сбалансированное состояние.
PendingTablets	Количество tablet со статусом задачи Pending.
RunningTablets	Количество tablet со статусом задачи Running.

Пример 12: Отображает информацию о группах Colocate JOIN в cluster.

```
mysql> SHOW PROC '/colocation_group';
+-----------------+----------------------------+---------------------------
-----------------------------------------------------------------------------
-----------------------------------------------+------------+----------------
+-------------+----------+
| GroupId         | GroupName                  | TableIds
| BucketsNum | ReplicationNum | DistCols    | IsStable |
+-----------------+----------------------------+---------------------------
-----------------------------------------------------------------------------
-----------------------------------------------+------------+----------------
+-------------+----------+
| 24010.177354    | 24010_lineitem_str_g1      | 177672
| 12         | 1              | varchar(-1) | true     |
| 24010.182146    | 24010_lineitem_str_g2      | 182144
| 192        | 1              | varchar(-1) | true     |
| 1439318.1735496 | 1439318_group_agent_uid    | 1735677, 1738390
| 12         | 2              | bigint(20)  | true     |
| 24010.37804     | 24010_gsdaf2449s9e         | 37802
| 192        | 1              | int(11)     | true     |
| 174844.175370   | 174844_groupa4             | 175368, 591307, 591362,
591389, 591416
| 12         | 1              | int(11)     | true     |
| 24010.30587     | 24010_group2               | 30585, 30669
| 12         | 1              | int(11)     | true     |
| 10005.181366    | 10005_lineorder_str_normal | 181364
| 192        | 1              | varchar(-1) | true     |
| 1904968.5973175 | 1904968_groupa2            | 5973173
| 12         | 1              | int(11)     | true     |
| 24010.182535    | 24010_lineitem_str_g3      | 182533
| 192        | 1              | varchar(-1) | true     |
+-----------------+----------------------------+---------------------------
-----------------------------------------------------------------------------
-----------------------------------------------+------------+----------------
+-------------+----------+
```

Возвращаемое значение	Описание
GroupId	ID группы Colocate JOIN.
GroupName	Название группы Colocate JOIN.
TableIds	ID таблиц в группе Colocate JOIN.
BucketsNum	Бакеты в группе Colocate JOIN.
ReplicationNum	Репликации в группе Colocate JOIN.
DistCols	Столбец распределения группы Colocate JOIN.

Возвращаемое значение	Описание
IsStable	Стабильна ли группа Colocate JOIN.

Пример 13: Отображает информацию о каталогах в cluster.

```
mysql> SHOW PROC '/catalog';
+-----------------------------------------------------------------+----------
+----------------------+
| Catalog                                                         | Type     |
Comment              |
+-----------------------------------------------------------------+----------
+----------------------+
| resource_mapping_inside_catalog_hive_hive2                      | hive     |
mapping hive catalog |
| resource_mapping_inside_catalog_hive_hive_resource_stability | hive     |
mapping hive catalog |
| default_catalog                                                 | Internal |
Internal Catalog     |
+-----------------------------------------------------------------+----------
+----------------------+
```

Возвращаемое значение	Описание
Catalog	Название каталога.
Type	Тип каталога.
Comment	Комментарии к каталогу.

Пример 14: Отображает информацию о задачах репликации в cluster.

```
mysql> SHOW PROC '/replications';
+-------------------------------------------------+------------+---------+-
------+---------------------+---------------------+-----------+----------+-
------+
| JobID                                           | DatabaseID | TableID |
TxnID | CreatedTime         | FinishedTime        | State     | Progress |
Error |
+-------------------------------------------------+------------+---------+-
------+---------------------+---------------------+-----------+----------+-
------+
| FAILOVER_GROUP_group1-11006-11010-1725593360156 | 11006      | 11010   |
99    | 2024-09-06 11:29:20 | 2024-09-06 11:29:21 | COMMITTED |          |
|
| FAILOVER_GROUP_group1-11006-11009-1725593360161 | 11006      | 11009   |
98    | 2024-09-06 11:29:20 | 2024-09-06 11:29:21 | COMMITTED |          |
|
| FAILOVER_GROUP_group1-11006-11074-1725593360161 | 11006      | 11074   |
100   | 2024-09-06 11:29:20 | 2024-09-06 11:29:21 | COMMITTED |          |
|
| FAILOVER_GROUP_group1-11006-12474-1725593360250 | 11006      | 12474   |
102   | 2024-09-06 11:29:20 | 2024-09-06 11:29:24 | COMMITTED |          |
|
| FAILOVER_GROUP_group1-11006-11024-1725593360293 | 11006      | 11024   |
101   | 2024-09-06 11:29:20 | 2024-09-06 11:29:24 | COMMITTED |          |
|
| FAILOVER_GROUP_group1-11006-13861-1725607270963 | 11006      | 13861   |
627   | 2024-09-06 15:21:10 | 2024-09-06 15:21:14 | COMMITTED |          |
|
+-------------------------------------------------+------------+---------+-
------+---------------------+---------------------+-----------+----------+-
------+
```

Возвращаемое значение	Описание
JobID	ID задания.
DatabaseID	ID базы данных.
TableID	ID таблицы.
TxnID	ID транзакции.
CreatedTime	Время создания задачи.
FinishedTime	Время завершения задачи.
State	Статус задачи. Допустимые значения: INITIALIZING, SNAPSHOTING, REPLICATING, COMMITTED, ABORTED.

Возвращаемое значение	Описание
Progress	Прогресс задачи.
Error	Сообщение об ошибке (если есть).

Пример 15: Отображает информацию о выполняемых запросах на текущем узле FE.

```
MySQL > show proc '/current_queries';
+---------------------+--------------+-----------------------------------
--+-------------+----------+------+-----------+-------------+----------
--+--------------+----------+----------+------------------+-------------
-+--------------+
| StartTime           | feIp          | QueryId
| ConnectionId | Database | User | ScanBytes  | ScanRows     | MemoryUsage
| DiskSpillSize | CPUTime  | ExecTime | Warehouse         | CustomQueryId |
ResourceGroup |
+---------------------+--------------+-----------------------------------
--+-------------+----------+------+-----------+-------------+----------
--+--------------+----------+----------+------------------+-------------
-+--------------+
| 2025-03-07 02:00:19 | 172.26.92.227 | ddbd69b9-fab4-11ef-8063-
461f20abc3f0 | 11           | tpcds_2  | root | 120.573 MB | 5859503 rows |
296.432 MB  | 0.000 B       | 27.888 s | 3.153 s  | default_warehouse |
| rg1           |
+---------------------+--------------+-----------------------------------
--+-------------+----------+------+-----------+-------------+----------
--+--------------+----------+----------+------------------+-------------
-+--------------+
```

Пример 16: Отображает информацию о выполняемых запросах на всех узлах FE в cluster.

```
MySQL > show proc '/global_current_queries';
+---------------------+---------------+-------------------------------------
--+--------------+----------+------+------------+--------------+-----------
--+---------------+---------+----------+-------------------+---------------
+---------------+
| StartTime           | feIp          | QueryId
| ConnectionId | Database | User | ScanBytes  | ScanRows     | MemoryUsage
| DiskSpillSize | CPUTime | ExecTime | Warehouse         | CustomQueryId |
ResourceGroup |
+---------------------+---------------+-------------------------------------
--+--------------+----------+------+------------+--------------+-----------
--+---------------+---------+----------+-------------------+---------------
+---------------+
| 2025-03-07 02:02:47 | 172.26.92.227 | 3603d566-fab5-11ef-8063-
461f20abc3f0 | 12           | tpcds_2  | root | 100.886 MB | 4899036 rows |
114.491 MB  | 0.000 B       | 5.700 s | 0.713 s  | default_warehouse |
| rg1           |
+---------------------+---------------+-------------------------------------
--+--------------+----------+------+------------+--------------+-----------
--+---------------+---------+----------+-------------------+---------------
+---------------+
```

SHOW PROCESSLIST

SHOW PROCESSLIST выводит список операций, выполняемых в данный момент потоками на сервере. Текущая версия Selena поддерживает только вывод списка запросов.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции не требуются привилегии.

Синтаксис

```
SHOW [FULL] PROCESSLIST
```

Параметр

Параметр	Обязательный	Описание
FULL	Нет	Если указан этот параметр, будет отображена полная SQL-команда. В противном случае отображаются только первые 100 символов команды.

Возвращаемые данные

Возвращаемое значение	Описание
Id	ID соединения.
User	Имя пользователя, выполняющего операцию.
Host	Имя хоста клиента, выполняющего операцию.
Db	Имя базы данных, в которой выполняется операция.
Command	Тип команды.
ConnectionStartTime	Время начала соединения.
Time	Время (в секундах) с момента перехода операции в текущее состояние.
State	Состояние операции.
Info	Команда, которую выполняет операция.
IsPending	Находится ли запрос в ожидании в очереди. Допустимые значения: `true` и `false`.
Warehouse	Название хранилища, в котором выполняется запрос.
CNGroup	Название группы вычислительных узлов, в которой выполняется запрос.

Примечание по использованию

Если текущий пользователь — `root`, эта команда выводит список операций всех пользователей в cluster. В противном случае выводятся только операции текущего пользователя.

Поля `IsPending`, `Warehouse` и `CNGroup` предоставляют дополнительную информацию о выполнении запросов в средах хранилищ:

- `IsPending`: Показывает, ожидает ли запрос в очереди (`true`) или активно выполняется (`false`)
- `Warehouse`: Отображает название хранилища, в котором выполняется запрос
- `CNGroup`: Показывает название группы вычислительных узлов, ответственной за выполнение запроса

Пример

Пример 1: вывод состояния операций от пользователя `root`.

```
SHOW PROCESSLIST;
+------+------+---------------------+-------+---------+-------------------
-+------+-------+------------------+-----------+-----------+---------+
| Id   | User | Host                | Db    | Command | ConnectionStartTime
| Time | State | Info             | IsPending  | Warehouse | CNGroup |
+------+------+---------------------+-------+---------+-------------------
-+------+-------+------------------+-----------+-----------+---------+
|  0   | root | x.x.x.x:xxxx        | tpcds | Query   | 2022-10-09 19:58:25
|    0 | OK    | SHOW PROCESSLIST | false      | default   |         |
+------+------+---------------------+-------+---------+-------------------
-+------+-------+------------------+-----------+-----------+---------+
```

SHOW RUNNING QUERIES

`SHOW RUNNING QUERIES` отображает информацию обо всех запросах, которые выполняются или ожидают в очереди запросов. Эта функция поддерживается начиная с версии v1.5.2.

> **ПОДСКАЗКА**
>
> Очереди запросов должны быть настроены, по умолчанию они отключены. Подробнее см. в документации по очередям запросов.

Синтаксис

```
SHOW RUNNING QUERIES
```

Возвращаемые данные

- `QueryId`: ID запроса.
- `ResourceGroupId`: ID группы ресурсов, которую задействовал запрос. Когда запрос не попал в пользовательскую группу ресурсов, отображается "-".
- `StartTime`: Время начала запроса.
- `PendingTimeout`: Время, когда ожидающий (PENDING) запрос истечёт по таймауту в очереди.
- `QueryTimeout`: Время, когда запрос истечёт по таймауту.

- `State`: Состояние запроса в очереди, где "PENDING" означает, что он находится в очереди, а "RUNNING" — что он выполняется.
- `Slots`: Количество логических ресурсов, запрошенных запросом, в настоящее время фиксировано как `1`.
- `Frontend`: Узел FE, инициировавший запрос.
- `FeStartTime`: Время запуска узла FE, инициировавшего запрос.

Пример

```
MySQL [(none)]> SHOW RUNNING QUERIES;
+---------------------------------------+-----------------+-----------------
----+---------------------+---------------------+-----------+-------+------
---------------------------+---------------------+
| QueryId                               | ResourceGroupId | StartTime
| PendingTimeout      | QueryTimeout        |   State   | Slots | Frontend
| FeStartTime         |
+---------------------------------------+-----------------+-----------------
----+---------------------+---------------------+-----------+-------+------
---------------------------+---------------------+
| a46f68c6-3b49-11ee-8b43-00163e10863a | -               | 2023-08-15
16:56:37 | 2023-08-15 17:01:37 | 2023-08-15 17:01:37 |  RUNNING  | 1     |
127.00.00.01_9010_1692069711535 | 2023-08-15 16:37:03 |
| a6935989-3b49-11ee-935a-00163e13bca3 | 12003           | 2023-08-15
16:56:40 | 2023-08-15 17:01:40 | 2023-08-15 17:01:40 |  RUNNING  | 1     |
127.00.00.02_9010_1692069658426 | 2023-08-15 16:37:03 |
| a7b5e137-3b49-11ee-8b43-00163e10863a | 12003           | 2023-08-15
16:56:42 | 2023-08-15 17:01:42 | 2023-08-15 17:01:42 |  PENDING  | 1     |
127.00.00.03_9010_1692069711535 | 2023-08-15 16:37:03 |
+---------------------------------------+-----------------+-----------------
----+---------------------+---------------------+-----------+-------+------
---------------------------+---------------------+
```

SYNC

Синхронизирует согласованность данных между различными сессиями.

В настоящее время Selena может гарантировать согласованность только в рамках одной сессии, то есть только сессия, инициировавшая операцию записи данных, может прочитать последние данные сразу после успешного завершения операции. Если вы хотите немедленно прочитать последние данные из других сессий, необходимо синхронизировать согласованность данных с помощью команды SYNC. Обычно без выполнения этой команды между сессиями будет задержка в несколько миллисекунд.

ПОДСКАЗКА

Для выполнения этой операции не требуются привилегии.

Синтаксис

```
SYNC
```

Примеры

```
mysql> SYNC;
Query OK, 0 rows affected (0.00 sec)
```

Configuration variables

ADMIN SET CONFIG

ADMIN SET CONFIG используется для установки параметров конфигурации cluster (в настоящее время с помощью этой команды можно устанавливать только динамические параметры конфигурации FE). Вы можете просмотреть эти параметры конфигурации с помощью команды ADMIN SHOW FRONTEND CONFIG.

После перезапуска FE конфигурации будут восстановлены до значений по умолчанию из файла `fe.conf`. Поэтому рекомендуется также изменить параметры конфигурации в файле `fe.conf`, чтобы предотвратить потерю изменений.

ПОДСКАЗКА

Для выполнения этой операции требуется привилегия OPERATE на уровне SYSTEM. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис

```
ADMIN SET FRONTEND CONFIG ("key" = "value")
```

Примеры

1. Установить `disable_balance` в значение `true`.

```
ADMIN SET FRONTEND CONFIG ("disable_balance" = "true");
```

ADMIN SHOW CONFIG

ADMIN SHOW CONFIG отображает конфигурацию текущего cluster (в настоящее время могут отображаться только параметры конфигурации FE). Подробное описание этих параметров конфигурации см. в разделе Конфигурация.

Если вы хотите установить или изменить параметр конфигурации, используйте ADMIN SET CONFIG.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия OPERATE на уровне SYSTEM. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис

```
ADMIN SHOW FRONTEND CONFIG [LIKE "pattern"]
```

Примечание:

Описание возвращаемых параметров:

```
1. Key:        Имя параметра конфигурации
2. Value:      Значение параметра конфигурации
3. Type:       Тип параметра конфигурации
4. IsMutable:  Можно ли установить через команду ADMIN SET CONFIG
5. MasterOnly: Применяется ли только к leader FE
6. Comment:    Описание параметра конфигурации
```

Примеры

1. Просмотр конфигурации текущего узла FE.

```
ADMIN SHOW FRONTEND CONFIG;
```

2. Поиск конфигурации текущего узла FE с использованием предиката `like`.

```
mysql> ADMIN SHOW FRONTEND CONFIG LIKE '%check_java_version%';
+--------------------+-------+---------+-----------+------------+------
---+
| Key                | Value | Type    | IsMutable | MasterOnly |
Comment |
+--------------------+-------+---------+-----------+------------+------
---+
| check_java_version | true  | boolean | false     | false      |
|
+--------------------+-------+---------+-----------+------------+------
---+
1 row in set (0.00 sec)
```

SET

Устанавливает указанные системные переменные или пользовательские переменные для Selena. Вы можете просмотреть системные переменные Selena с помощью SHOW VARIABLES. Подробнее о системных переменных см. в разделе Системные переменные. Подробнее о пользовательских переменных см. в разделе Пользовательские переменные.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции не требуются привилегии.

Синтаксис

```
SET [ GLOBAL | SESSION ] <variable_name> = <value> [, <variable_name> =
<value>] ...
```

Параметры

Параметр	Описание
Модификатор: • GLOBAL • SESSION	• С модификатором `GLOBAL` команда устанавливает переменные глобально. • С модификатором `SESSION` команда устанавливает переменные в рамках сессии. `LOCAL` является синонимом для `SESSION`. • Если модификатор не указан, по умолчанию используется `SESSION`. Подробнее о глобальных и сессионных переменных см. в разделе Системные переменные. **ПРИМЕЧАНИЕ**

Параметр	Описание
	Только пользователи с привилегией ADMIN могут устанавливать переменные глобально.
variable_name	Имя переменной.
value	Значение переменной.

Примеры

Пример 1: Установить `time_zone` в значение `Asia/Shanghai` в рамках сессии.

```
mysql> SET time_zone = "Asia/Shanghai";
Query OK, 0 rows affected (0.00 sec)
```

Пример 2: Установить `exec_mem_limit` в значение `2147483648` глобально.

```
mysql> SET GLOBAL exec_mem_limit = 2147483648;
Query OK, 0 rows affected (0.00 sec)
```

Пример 3: Установить несколько глобальных переменных. Добавьте ключевое слово `GLOBAL` перед каждой переменной.

```
mysql> SET
       GLOBAL exec_mem_limit = 2147483648,
       GLOBAL time_zone = "Asia/Shanghai";
Query OK, 0 rows affected (0.00 sec)
```

SHOW VARIABLES

SHOW VARIABLES отображает системные переменные Selena. Подробнее о системных переменных см. в разделе Системные переменные.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции не требуются привилегии.

Синтаксис

```
SHOW [ GLOBAL | SESSION ] VARIABLES
    [ LIKE <pattern> | WHERE <expr> ]
```

Параметры

Параметр	Описание
Модификатор: • GLOBAL • SESSION	• С модификатором `GLOBAL` команда отображает значения глобальных системных переменных. Эти значения используются для инициализации соответствующих сессионных переменных для новых подключений к Selena. Если переменная не имеет глобального значения, значение не отображается. • С модификатором `SESSION` команда отображает значения системных переменных, действующих для текущего подключения. Если переменная не имеет сессионного значения, отображается глобальное значение. `LOCAL` является синонимом для `SESSION`. • Если модификатор не указан, по умолчанию используется `SESSION`.
pattern	Шаблон, используемый для поиска переменной по имени с помощью условия LIKE. В этом параметре можно использовать символ подстановки %.
expr	Выражение, используемое для поиска переменной по имени `variable_name` или значению переменной `value` с помощью условия WHERE.

Возвращаемое значение

Возвращаемое значение	Описание
Variable_name	Имя переменной.
Value	Значение переменной.

Примеры

Пример 1: Показать переменную с точным совпадением имени переменной с помощью условия LIKE.

```
mysql> SHOW VARIABLES LIKE 'wait_timeout';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| wait_timeout  | 28800 |
+---------------+-------+
1 row in set (0.01 sec)
```

Пример 2: Показать переменные с приблизительным совпадением имени переменной с помощью условия LIKE и символа подстановки (%).

```
mysql> SHOW VARIABLES LIKE '%imeou%';
+-------------------------------------+-------+
| Variable_name                       | Value |
+-------------------------------------+-------+
| interactive_timeout                 | 3600  |
| net_read_timeout                    | 60    |
| net_write_timeout                   | 60    |
| new_planner_optimize_timeout        | 3000  |
| query_delivery_timeout              | 300   |
| query_queue_pending_timeout_second | 300   |
| query_timeout                       | 300   |
| tx_visible_wait_timeout             | 10    |
| wait_timeout                        | 28800 |
+-------------------------------------+-------+
9 rows in set (0.00 sec)
```

Пример 3: Показать переменную с точным совпадением имени переменной с помощью условия WHERE.

```
mysql> show variables where variable_name = 'wait_timeout';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| wait_timeout  | 28800 |
+---------------+-------+
1 row in set (0.17 sec)
```

Пример 4: Показать переменную с точным совпадением значения переменной с помощью условия WHERE.

```
mysql> show variables where value = '28800';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| wait_timeout  | 28800 |
+---------------+-------+
1 row in set (0.70 sec)
```

Resource group

ALTER RESOURCE GROUP

ALTER RESOURCE GROUP изменяет конфигурацию группы ресурсов.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия ALTER на целевую группу ресурсов. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис

```
ALTER RESOURCE GROUP resource_group_name
{  ADD CLASSIFIER1, CLASSIFIER2, ...
 | DROP (CLASSIFIER_ID_1, CLASSIFIER_ID_2, ...)
 | DROP ALL
 | WITH resource_limit
};
```

Параметры

Параметр	Описание
resource_group_name	Название группы ресурсов для изменения.
ADD	Добавление классификаторов в группу ресурсов. См. CREATE RESOURCE GROUP - Параметры для получения информации о том, как определить классификатор.
DROP	Удаление классификаторов из группы ресурсов по ID классификаторов. Вы можете проверить ID классификатора с помощью команды SHOW RESOURCE GROUP.
DROP ALL	Удаление всех классификаторов из группы ресурсов.
WITH	Изменение ограничений ресурсов группы ресурсов. См. CREATE RESOURCE GROUP - Параметры для получения информации о том, как установить ограничения ресурсов.

Примеры

Пример 1: Добавление нового классификатора в группу ресурсов `rg1`.

```
ALTER RESOURCE GROUP rg1 ADD (user='root', query_type in ('select'));
```

Пример 2: Удаление классификаторов с ID `300040`, `300041` и `300041` из группы ресурсов `rg1`.

```
ALTER RESOURCE GROUP rg1 DROP (300040, 300041, 300041);
```

Пример 3: Удаление всех классификаторов из группы ресурсов `rg1`.

```
ALTER RESOURCE GROUP rg1 DROP ALL;
```

Пример 4: Изменение ограничений ресурсов группы ресурсов `rg1`.

```
ALTER RESOURCE GROUP rg1 WITH (
    'cpu_weight' = '10'
);
```

CREATE RESOURCE GROUP

CREATE RESOURCE GROUP создаёт группу ресурсов.

Подробнее см. в Группа ресурсов.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия CREATE RESOURCE GROUP на уровне SYSTEM. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис

```
CREATE RESOURCE GROUP resource_group_name
TO CLASSIFIER1, CLASSIFIER2, ...
WITH resource_limit
```

Параметры

- `resource_group_name`: Название создаваемой группы ресурсов.

- `CLASSIFIER`: Классификаторы, используемые для фильтрации запросов, на которые накладываются ограничения ресурсов. Классификатор необходимо указывать в виде пар `"key"="value"`. Для группы ресурсов можно установить несколько классификаторов.

  Параметры для классификаторов перечислены ниже:

Параметр	Обязательный	Описание
user	Нет	Имя пользователя.
role	Нет	Роль пользователя.
query_type	Нет	Тип запроса. Поддерживаются `SELECT` и `INSERT` (с версии v1.5.2). Когда задачи INSERT попадают в группу ресурсов с `query_type` равным `insert`, узел BE резервирует указанные ресурсы CPU для задач.
source_ip	Нет	CIDR-блок, с которого инициируется запрос.
db	Нет	База данных, к которой обращается запрос. Можно указать несколько через запятую (,).
plan_cpu_cost_range	Нет	Оценочный диапазон стоимости CPU для запроса. Его значение эквивалентно полю `PlanCpuCost` в **fe.audit.log** и не имеет единицы измерения. Формат: `[DOUBLE, DOUBLE)`. Значение по умолчанию NULL означает отсутствие такого ограничения. Этот параметр поддерживается с версии v1.5.2.
plan_mem_cost_range	Нет	Оценочный диапазон стоимости памяти для запроса. Его значение эквивалентно полю `PlanMemCost` в **fe.audit.log** и не имеет единицы измерения. Формат: `[DOUBLE, DOUBLE)`. Значение по умолчанию NULL означает отсутствие такого ограничения. Этот параметр поддерживается с версии v1.5.2.

- `resource_limit`: Ограничения ресурсов, накладываемые на группу ресурсов. Ограничения необходимо указывать в виде пар `"key"="value"`. Для группы ресурсов можно установить несколько ограничений.

  Параметры для ограничений ресурсов перечислены ниже:

Параметр	Обязательный	Описание
cpu_core_limit	Нет	Мягкое ограничение на количество ядер CPU, которые могут быть выделены группе ресурсов на BE. В реальных бизнес-сценариях ядра CPU, выделяемые группе ресурсов,

Параметр	Обязательный	Описание
		пропорционально масштабируются в зависимости от доступности ядер CPU на BE. Допустимые значения: любое ненулевое положительное целое число.
cpu_weight	Нет	Обязательный параметр для создания Shared-групп ресурсов. Указывает вес планирования CPU для группы ресурсов на одном узле BE, определяя относительную долю процессорного времени, выделяемого задачам этой группы. В реальных бизнес-сценариях ядра CPU, выделяемые группе ресурсов, пропорционально масштабируются в зависимости от доступности ядер CPU на BE. Диапазон значений: (0, `avg_be_cpu_cores`], где `avg_be_cpu_cores` — среднее количество ядер CPU на всех узлах BE. Параметр действует только при значении больше 0. Только один из параметров `cpu_weight` или `exclusive_cpu_cores` может быть установлен больше 0.
exclusive_cpu_cores	Нет	Обязательный параметр для создания Exclusive-групп ресурсов (с жёстким ограничением CPU). Означает резервирование `exclusive_cpu_cores` ядер CPU исключительно для этой группы ресурсов, делая их недоступными для других групп, даже когда они простаивают, и ограничивая группу ресурсов использованием только этих зарезервированных ядер, предотвращая использование доступных ресурсов CPU других групп. Диапазон значений: (0, `min_be_cpu_cores - 1`], где `min_be_cpu_cores` — минимальное количество ядер CPU на всех узлах BE. Параметр действует только при значении больше 0. Только один из параметров `cpu_weight` или `exclusive_cpu_cores` может быть установлен больше 0.
mem_limit	Нет	Процент памяти, который может использоваться для запросов, от общей памяти, предоставляемой BE. Единица: %. Допустимые значения: (0, 1).
concurrency_limit	Нет	Верхний предел параллельных запросов в группе ресурсов. Используется для предотвращения перегрузки системы из-за слишком большого количества параллельных запросов.
max_cpu_cores	Нет	Ограничение ядер CPU для этой группы ресурсов на одном узле BE. Действует только при значении больше `0`. Диапазон: [0, `avg_be_cpu_cores`], где `avg_be_cpu_cores` — среднее количество ядер CPU на всех узлах BE. По умолчанию: 0.
big_query_cpu_second_limit	Нет	Верхний предел времени CPU для большого запроса. Параллельные запросы суммируют время. Единица измерения: секунды.
big_query_scan_r	Нет	Верхний предел количества строк, которые могут быть просканированы большим запросом.

Параметр	Обязательный	Описание
ows_limit		
big_query_mem_limit	Нет	Верхний предел использования памяти для большого запроса. Единица измерения: байты.

> **ПРИМЕЧАНИЕ**
>
> До версии v1.5.2 Selena позволяла устанавливать `type` группы ресурсов в `short_query`. Однако параметр `type` устарел и заменён на `exclusive_cpu_cores`. Для любых существующих групп ресурсов этого типа система автоматически преобразует их в Exclusive-группу ресурсов, где значение `exclusive_cpu_cores` равно `cpu_weight` после обновления до версии v1.5.2.

Пример

Пример 1: Создание Shared-группы ресурсов `rg1` на основе нескольких классификаторов.

```
CREATE RESOURCE GROUP rg1
TO
    (user='rg1_user1', role='rg1_role1', query_type in ('select'),
source_ip='192.168.x.x/24'),
    (user='rg1_user2', query_type in ('select'),
source_ip='192.168.x.x/24'),
    (user='rg1_user3', source_ip='192.168.x.x/24'),
    (user='rg1_user4'),
    (db='db1')
WITH ('cpu_weight' = '10',
      'mem_limit' = '20%',
      'big_query_cpu_second_limit' = '100',
      'big_query_scan_rows_limit' = '100000',
      'big_query_mem_limit' = '1073741824'
);
```

Пример 2: Создание Exclusive-группы ресурсов `rg2` на основе нескольких классификаторов.

```
CREATE RESOURCE GROUP rg2
TO
    (user='rg1_user5', role='rg1_role5', query_type in ('select'),
source_ip='192.168.x.x/24'),
    (user='rg1_user6', query_type in ('select'),
source_ip='192.168.x.x/24'),
    (user='rg1_user7', source_ip='192.168.x.x/24'),
    (user='rg1_user8'),
    (db='db2')
WITH ('exclusive_cpu_cores' = '10',
      'mem_limit' = '20%',
      'type' = 'normal',
      'big_query_cpu_second_limit' = '100',
      'big_query_scan_rows_limit' = '100000',
      'big_query_mem_limit' = '1073741824'
);
```

DROP RESOURCE GROUP

DROP RESOURCE GROUP [IF EXISTS] удаляет указанную группу ресурсов.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия DROP на целевую группу ресурсов. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис

```
DROP RESOURCE GROUP <resource_group_name>
```

Параметры

Параметр	Описание
resource_group_name	Название группы ресурсов для удаления.

Пример

Пример 1: Удаление группы ресурсов `rg1`.

```
DROP RESOURCE GROUP rg1;
```

SHOW RESOURCE GROUP

SHOW RESOURCE GROUP показывает информацию о группах ресурсов.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции не требуются привилегии.

Синтаксис

```
SHOW { RESOURCE GROUPS [ALL] | RESOURCE GROUP <resource_group_name> }
```

Параметры

Параметр	Описание
RESOURCE GROUPS	Показывает группы ресурсов, соответствующие текущему пользователю.
ALL	Укажите этот параметр, чтобы показать все группы ресурсов в cluster.
RESOURCE GROUP	Показывает указанную группу ресурсов.
resource_group_name	Название группы ресурсов для отображения.

Возвращаемые данные

Возвращаемое значение	Описание
name	Название группы ресурсов.
id	ID группы ресурсов.
cpu_weight	Вес планирования CPU этой группы ресурсов на узле BE.
exclusive_cpu_cores	Параметр жёсткой изоляции CPU для этой группы ресурсов.
mem_limit	Ограничение памяти группы ресурсов.
big_query_cpu_second_limit	Верхний предел времени CPU для больших запросов.
big_query_scan_rows_limit	Ограничение сканирования строк для больших запросов.

Возвращаемое значение	Описание
big_query_mem_limit	Ограничение памяти для больших запросов.
concurrency_limit	Ограничение параллелизма группы ресурсов.
spill_mem_limit_threshold	Порог использования памяти, запускающий сброс на диск.
classifiers	Классификаторы, связанные с группой ресурсов. `id` — это ID классификатора, а `weight` — степень соответствия.

Примеры

Пример 1: Показать все группы ресурсов в cluster.

```
mysql> SHOW RESOURCE GROUPS ALL;
+---------------+------+------------+---------------------+-----------+----
------------------------+---------------------------+---------------------
+-------------------+---------------------------+--------------------+
| name          | id   | cpu_weight | exclusive_cpu_cores | mem_limit |
big_query_cpu_second_limit | big_query_scan_rows_limit |
big_query_mem_limit | concurrency_limit | spill_mem_limit_threshold |
classifiers        |
+---------------+------+------------+---------------------+-----------+----
------------------------+---------------------------+---------------------
+-------------------+---------------------------+--------------------+
| default_mv_wg | 3    | 1          | 0                   | 80.0%     | 0
| 0                         | 0                   | null              | 80%
| (id=0, weight=0.0) |
| default_wg    | 2    | 32         | 0                   | 100.0%    | 0
| 0                         | 0                   | null              |
100%                      | (id=0, weight=0.0) |
+---------------+------+------------+---------------------+-----------+----
------------------------+---------------------------+---------------------
+-------------------+---------------------------+--------------------+
```

SHOW USAGE RESOURCE GROUPS

SHOW USAGE RESOURCE GROUPS показывает информацию об использовании групп ресурсов. Эта функция поддерживается с версии v1.5.2.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции не требуются привилегии.

Синтаксис

```
SHOW USAGE RESOURCE GROUPS
```

Возвращаемые данные

- `Name`: Название группы ресурсов.
- `Id`: ID группы ресурсов.
- `Backend`: IP-адрес или FQDN узла BE.
- `BEInUseCpuCores`: Количество ядер CPU, используемых в данный момент этой группой ресурсов на этом BE. Это значение является приблизительной оценкой.
- `BEInUseMemBytes`: Количество байт памяти, используемых в данный момент этой группой ресурсов на этом BE.
- `BERunningQueries`: Количество запросов от этой группы ресурсов, которые всё ещё выполняются на этом BE.

Примечания по использованию

- BE периодически отправляют информацию об использовании ресурсов на Leader FE с интервалом, указанным в `report_resource_usage_interval_ms`, который по умолчанию равен 1 секунде.
- Результаты будут показывать только строки, в которых хотя бы один из параметров `BEInUseCpuCores`/`BEInUseMemBytes`/`BERunningQueries` является положительным числом. Другими словами, информация отображается только тогда, когда группа ресурсов активно использует какие-либо ресурсы на BE.

Пример

```
MySQL [(none)]> SHOW USAGE RESOURCE GROUPS;
+------------+----+-----------+-----------------+-----------------+--------
----------+
| Name       | Id | Backend   | BEInUseCpuCores | BEInUseMemBytes |
BERunningQueries |
+------------+----+-----------+-----------------+-----------------+--------
----------+
| default_wg | 0  | 127.0.0.1 | 0.100           | 1               | 5
|
+------------+----+-----------+-----------------+-----------------+--------
----------+
| default_wg | 0  | 127.0.0.2 | 0.200           | 2               | 6
|
+------------+----+-----------+-----------------+-----------------+--------
----------+
| wg1        | 0  | 127.0.0.1 | 0.300           | 3               | 7
|
+------------+----+-----------+-----------------+-----------------+--------
----------+
| wg2        | 0  | 127.0.0.1 | 0.400           | 4               | 8
|
+------------+----+-----------+-----------------+-----------------+--------
----------+
```

Storage volume

ALTER STORAGE VOLUME

ALTER STORAGE VOLUME изменяет свойства учётных данных, комментарий или статус (`enabled`) тома хранения. Подробнее о свойствах тома хранения см. в CREATE STORAGE VOLUME. Эта функция поддерживается начиная с v1.5.2.

> **ВНИМАНИЕ**
>
> - Только пользователи с привилегией ALTER на определённый том хранения могут выполнять эту операцию.
> - Свойства `TYPE`, `LOCATIONS` и другие свойства, связанные с путём, существующего тома хранения изменить нельзя. Вы можете изменить только свойства, связанные с учётными данными. Если вы изменили элементы конфигурации, связанные с путём, базы данных и таблицы, созданные до

> изменения, станут доступны только для чтения, и вы не сможете загружать в них данные.
> - Когда `enabled` имеет значение `false`, соответствующий том хранения не может быть использован.

Синтаксис

```
ALTER STORAGE VOLUME [ IF EXISTS ] <storage_volume_name>
{ COMMENT = '<comment_string>'
| SET ("key" = "value"[,...]) }
```

Параметры

Параметр	Описание
storage_volume_name	Название тома хранения для изменения.
COMMENT	Комментарий к тому хранения.

Подробную информацию о свойствах, которые можно изменить или добавить, см. в CREATE STORAGE VOLUME - PROPERTIES.

Примеры

Пример 1: Отключение тома хранения `my_s3_volume`.

```
MySQL > ALTER STORAGE VOLUME my_s3_volume
    -> SET ("enabled" = "false");
Query OK, 0 rows affected (0.01 sec)
```

Пример 2: Изменение информации об учётных данных тома хранения `my_s3_volume`.

```
MySQL > ALTER STORAGE VOLUME my_s3_volume
    -> SET (
    ->     "aws.s3.use_instance_profile" = "true"
    -> );
Query OK, 0 rows affected (0.00 sec)
```

Связанные SQL-команды

- CREATE STORAGE VOLUME

- DROP STORAGE VOLUME
- SET DEFAULT STORAGE VOLUME
- DESC STORAGE VOLUME
- SHOW STORAGE VOLUMES

CREATE STORAGE VOLUME

CREATE STORAGE VOLUME создаёт том хранения для удалённой системы хранения. Эта функция поддерживается начиная с v1.5.2.

Том хранения состоит из свойств и информации об учётных данных удалённого хранилища данных. Вы можете ссылаться на том хранения при создании баз данных и cloud-native таблиц в cluster Selena с разделяемыми данными.

> **ВНИМАНИЕ**
>
> Только пользователи с привилегией CREATE STORAGE VOLUME на уровне SYSTEM могут выполнять эту операцию.

Синтаксис

```
CREATE STORAGE VOLUME [IF NOT EXISTS] <storage_volume_name>
TYPE = { S3 | HDFS | AZBLOB | ADLS2 | GS }
LOCATIONS = ('<remote_storage_path>')
[ COMMENT '<comment_string>' ]
PROPERTIES
("key" = "value",...)
```

Параметры

Параметр	Описание
storage_volume_name	Название тома хранения. Обратите внимание, что нельзя создать том хранения с именем `builtin_storage_volume`, так как оно зарезервировано для встроенного тома хранения. Соглашения об именовании см. в Системные ограничения.
TYPE	Тип удалённой системы хранения. Допустимые значения: `S3`, `HDFS`, `AZBLOB`, `ADLS2` и `GS`. `S3` означает AWS S3 или S3-совместимые системы хранения. `AZBLOB` означает Azure Blob Storage (поддерживается с v1.5.2). `ADLS2` означает Azure Data Lake Storage Gen2 (поддерживается с v1.5.2). `HDFS` означает cluster HDFS. `GS` означает Google Storage (доступ через нативный SDK, поддерживается с v1.5.2).

Параметр	Описание
LOCATIONS	Расположения хранения. Формат следующий: • Для AWS S3 или S3-совместимых систем хранения: `s3://<s3_path>`. `<s3_path>` должен быть абсолютным путём, например, `s3://testbucket/subpath`. Обратите внимание, что если вы хотите включить функцию Partitioned Prefix для тома хранения, вы можете указать только имя бакета, указание подпути не допускается. • Для Azure Blob Storage: `azblob://<azblob_path>`. `<azblob_path>` должен быть абсолютным путём, например, `azblob://testcontainer/subpath`. • Для Azure Data Lake Storage Gen2: `adls2://<file_system_name>/<dir_name>`. Пример: `adls2://testfilesystem/selena`. • Для GS с нативным SDK: `gs://<gs_path>`. `<gs_path>` должен быть абсолютным путём, например, `gs://testcbucket/subpath`. • Для HDFS: `hdfs://<host>:<port>/<hdfs_path>`. `<hdfs_path>` должен быть абсолютным путём, например, `hdfs://127.0.0.1:9000/user/xxx/selena`. • Для WebHDFS: `webhdfs://<host>:<http_port>/<hdfs_path>`, где `<http_port>` — HTTP-порт NameNode. `<hdfs_path>` должен быть абсолютным путём, например, `webhdfs://127.0.0.1:50070/user/xxx/selena`. • Для ViewFS: `viewfs://<ViewFS_cluster>/<viewfs_path>`, где `<ViewFS_cluster>` — имя cluster ViewFS. `<viewfs_path>` должен быть абсолютным путём, например, `viewfs://myviewfscluster/user/xxx/selena`.
COMMENT	Комментарий к тому хранения.
PROPERTIES	Параметры в парах `"key" = "value"`, используемые для указания свойств и информации об учётных данных для доступа к удалённой системе хранения. Подробную информацию см. в PROPERTIES.

PROPERTIES

В таблице ниже перечислены все доступные свойства томов хранения. После таблицы приведены инструкции по использованию этих свойств, разделённые по различным сценариям с точки зрения Информации об учётных данных и Функций.

Свойство	Описание
enabled	Включить ли этот том хранения. По умолчанию: `false`. Отключённый том хранения не может быть использован.
aws.s3.region	Регион, в котором находится ваш бакет S3, например, `us-west-2`.
aws.s3.endpoint	URL конечной точки для доступа к вашему бакету S3, например, `https://s3.us-west-2.amazonaws.com`. [Превью] Начиная с v1.5.2 поддерживается класс хранения Amazon S3 Express One Zone, например, `https://s3express.us-west-2.amazonaws.com`. *Beta* **BETA FEATURE** Рекомендации по использованию бета-функций

Свойство	Описание
aws.s3.use_aws_sdk_default_behavior	Использовать ли учётные данные аутентификации по умолчанию AWS SDK. Допустимые значения: `true` и `false` (по умолчанию).
aws.s3.use_instance_profile	Использовать ли Instance Profile и Assumed Role в качестве методов аутентификации для доступа к S3. Допустимые значения: `true` и `false` (по умолчанию). • Если вы используете учётные данные на основе IAM-пользователя (Access Key и Secret Key) для доступа к S3, вы должны установить этот параметр в `false` и указать `aws.s3.access_key` и `aws.s3.secret_key`. • Если вы используете Instance Profile для доступа к S3, вы должны установить этот параметр в `true`. • Если вы используете Assumed Role для доступа к S3, вы должны установить этот параметр в `true` и указать `aws.s3.iam_role_arn`. • Если вы используете внешнюю учётную запись AWS, вы должны установить этот параметр в `true` и указать `aws.s3.iam_role_arn` и `aws.s3.external_id`.
aws.s3.access_key	Access Key ID для доступа к вашему бакету S3.
aws.s3.secret_key	Secret Access Key для доступа к вашему бакету S3.
aws.s3.iam_role_arn	ARN роли IAM с привилегиями на ваш бакет S3, в котором хранятся ваши файлы данных.
aws.s3.external_id	Внешний ID учётной записи AWS, который используется для межаккаунтного доступа к вашему бакету S3.
aws.s3.enable_path_style_access	Использовать ли Path Style для доступа к S3. Допустимые значения: `true` и `false`. По умолчанию: `false`.
azure.blob.endpoint	Конечная точка вашей учётной записи Azure Blob Storage, например, `https://test.blob.core.windows.net`.
azure.blob.shared_key	Shared Key для авторизации запросов к Azure Blob Storage.
azure.blob.sas_token	Подписи общего доступа (SAS) для авторизации запросов к Azure Blob Storage.
azure.adls2.endpoint	Конечная точка вашей учётной записи Azure Data Lake Storage Gen2, например, `https://test.dfs.core.windows.net`.
azure.adls2.shared_key	Shared Key для авторизации запросов к Azure Data Lake Storage Gen2.
azure.adls2.sas_token	Подписи общего доступа (SAS) для авторизации запросов к Azure Data Lake Storage Gen2.
azure.adls2.oauth2_use_managed_identity	Использовать ли Managed Identity для авторизации запросов к Azure Data Lake Storage Gen2. По умолчанию: `false`.
azure.adls2.oauth2_tenant_id	Tenant ID управляемого удостоверения для авторизации запросов к Azure Data Lake Storage Gen2.
azure.adls2.oauth2_client_id	Client ID управляемого удостоверения для авторизации запросов к Azure Data Lake Storage Gen2.

Свойство	Описание
gcp.gcs.service_account_email	Адрес электронной почты в JSON-файле, созданном при создании Service Account, например, `user@hello.iam.gserviceaccount.com`.
gcp.gcs.service_account_private_key_id	Private Key ID в JSON-файле, созданном при создании Service Account.
gcp.gcs.service_account_private_key	Private Key в JSON-файле, созданном при создании Service Account, например, `-----BEGIN PRIVATE KEY----xxxx-----END PRIVATE KEY-----\n`.
gcp.gcs.impersonation_service_account	Service Account, который вы хотите имперсонировать, если используете аутентификацию на основе имперсонации.
gcp.gcs.use_compute_engine_service_account	Использовать ли Service Account, привязанный к вашему Compute Engine.
hadoop.security.authentication	Метод аутентификации. Допустимые значения: `simple` (по умолчанию) и `kerberos`. `simple` означает простую аутентификацию, то есть по имени пользователя. `kerberos` означает аутентификацию Kerberos.
username	Имя пользователя для доступа к NameNode в cluster HDFS.
hadoop.security.kerberos.ticket.cache.path	Путь к Ticket Cache, сгенерированному kinit.
dfs.nameservices	Имя cluster HDFS.
dfs.ha.namenodes.`<ha_cluster_name>`	Имя NameNode. Несколько имён должны быть разделены запятыми (,). Пробелы в кавычках не допускаются. `<ha_cluster_name>` — имя службы HDFS, указанное в `dfs.nameservices`.
dfs.namenode.rpc-address.`<ha_cluster_name>`.`<NameNode>`	Информация об адресе RPC NameNode. `<NameNode>` — имя NameNode, указанное в `dfs.ha.namenodes.<ha_cluster_name>`.
dfs.client.failover.proxy.provider	Провайдер NameNode для подключения клиента. Значение по умолчанию: `org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider`.
fs.viewfs.mounttable.`<ViewFS_cluster>`.link./`<viewfs_path>`	Путь к монтируемому cluster ViewFS. Несколько путей должны быть разделены запятыми (,). `<ViewFS_cluster>` — имя cluster ViewFS, указанное в `LOCATIONS`.
aws.s3.enable_partitioned_prefix	Включить ли функцию Partitioned Prefix для тома хранения. По умолчанию: `false`. Подробнее об этой функции см. в Partitioned Prefix.
aws.s3.num_partitioned_prefix	Количество префиксов для создания для тома хранения. По умолчанию: `256`. Допустимый диапазон: [4, 1024].

Информация об учётных данных

AWS S3

- Если вы используете учётные данные аутентификации по умолчанию AWS SDK для доступа к S3, установите следующие свойства:

```
"enabled" = "{ true | false }",
"aws.s3.region" = "<region>",
"aws.s3.endpoint" = "<endpoint_url>",
"aws.s3.use_aws_sdk_default_behavior" = "true"
```

- Если вы используете учётные данные на основе IAM-пользователя (Access Key и Secret Key) для доступа к S3, установите следующие свойства:

```
"enabled" = "{ true | false }",
"aws.s3.region" = "<region>",
"aws.s3.endpoint" = "<endpoint_url>",
"aws.s3.use_aws_sdk_default_behavior" = "false",
"aws.s3.use_instance_profile" = "false",
"aws.s3.access_key" = "<access_key>",
"aws.s3.secret_key" = "<secrete_key>"
```

- Если вы используете Instance Profile для доступа к S3, установите следующие свойства:

```
"enabled" = "{ true | false }",
"aws.s3.region" = "<region>",
"aws.s3.endpoint" = "<endpoint_url>",
"aws.s3.use_aws_sdk_default_behavior" = "false",
"aws.s3.use_instance_profile" = "true"
```

- Если вы используете Assumed Role для доступа к S3, установите следующие свойства:

```
"enabled" = "{ true | false }",
"aws.s3.region" = "<region>",
"aws.s3.endpoint" = "<endpoint_url>",
"aws.s3.use_aws_sdk_default_behavior" = "false",
"aws.s3.use_instance_profile" = "true",
"aws.s3.iam_role_arn" = "<role_arn>"
```

- Если вы используете Assumed Role для доступа к S3 из внешней учётной записи AWS, установите следующие свойства:

```
"enabled" = "{ true | false }",
"aws.s3.region" = "<region>",
"aws.s3.endpoint" = "<endpoint_url>",
"aws.s3.use_aws_sdk_default_behavior" = "false",
"aws.s3.use_instance_profile" = "true",
"aws.s3.iam_role_arn" = "<role_arn>",
"aws.s3.external_id" = "<external_id>"
```

MinIO

Если вы используете MinIO, установите следующие свойства:

```
"enabled" = "{ true | false }",

-- Например: us-east-1
"aws.s3.region" = "<region>",

-- Например: http://172.26.xx.xxx:39000
"aws.s3.endpoint" = "<endpoint_url>",

"aws.s3.access_key" = "<access_key>",
"aws.s3.secret_key" = "<secrete_key>"
```

Azure Blob Storage

Создание тома хранения в Azure Blob Storage поддерживается начиная с v1.5.2.

- Если вы используете Shared Key для доступа к Azure Blob Storage, установите следующие свойства:

```
"enabled" = "{ true | false }",
"azure.blob.endpoint" = "<endpoint_url>",
"azure.blob.shared_key" = "<shared_key>"
```

- Если вы используете подписи общего доступа (SAS) для доступа к Azure Blob Storage, установите следующие свойства:

```
"enabled" = "{ true | false }",
"azure.blob.endpoint" = "<endpoint_url>",
"azure.blob.sas_token" = "<sas_token>"
```

> **ПРИМЕЧАНИЕ**
>
> Иерархическое пространство имён должно быть отключено при создании учётной записи Azure Blob Storage.

Azure Data Lake Storage Gen2

Создание тома хранения в Azure Data Lake Storage Gen2 поддерживается начиная с v1.5.2.

- Если вы используете Shared Key для доступа к Azure Data Lake Storage Gen2, установите следующие свойства:

```
"enabled" = "{ true | false }",
"azure.adls2.endpoint" = "<endpoint_url>",
"azure.adls2.shared_key" = "<shared_key>"
```

- Если вы используете подписи общего доступа (SAS) для доступа к Azure Data Lake Storage Gen2, установите следующие свойства:

```
"enabled" = "{ true | false }",
"azure.adls2.endpoint" = "<endpoint_url>",
"azure.adls2.sas_token" = "<sas_token>"
```

- Если вы используете Managed Identity для доступа к Azure Data Lake Storage Gen2, установите следующие свойства:

```
"enabled" = "{ true | false }",
"azure.adls2.endpoint" = "<endpoint_url>",
"azure.adls2.oauth2_use_managed_identity" = "true",
"azure.adls2.oauth2_tenant_id" = "<tenant_id>",
"azure.adls2.oauth2_client_id" = "<client_id>"
```

> **ПРИМЕЧАНИЕ**
>
> Azure Data Lake Storage Gen1 не поддерживается.

Google Storage

- Если вы используете Service Account, привязанный к вашему Compute Engine, для доступа к Google Storage (поддерживается с v1.5.2), установите следующие свойства:

```
"enabled" = "{ true | false }",
"gcp.gcs.use_compute_engine_service_account" = "true"
```

- Если вы используете метод аутентификации на основе Service Account для доступа к Google Storage (поддерживается с v1.5.2), установите следующие свойства:

```
"enabled" = "{ true | false }",
"gcp.gcs.use_compute_engine_service_account" = "false",
"gcp.gcs.service_account_email" = "<google_service_account_email>",
"gcp.gcs.service_account_private_key_id" = "
<google_service_private_key_id>",
"gcp.gcs.service_account_private_key" = "<google_service_private_key>"
```

- Если вы используете аутентификацию на основе имперсонации для доступа к Google Storage (поддерживается с v1.5.2), установите следующие свойства:

```
"enabled" = "{ true | false }",
"gcp.gcs.use_compute_engine_service_account" = "false",
"gcp.gcs.service_account_email" = "<google_service_account_email>",
"gcp.gcs.service_account_private_key_id" = "
<google_service_private_key_id>",
"gcp.gcs.service_account_private_key" = "<google_service_private_key>",
"gcp.gcs.impersonation_service_account" = "
<assumed_google_service_account_email>"
```

- Если вы используете протокол S3 с аутентификацией на основе IAM-пользователя для доступа к Google Storage, установите следующие свойства:

> **ПОДСКАЗКА**
>
> Google Storage поддерживается с использованием XML API, и настройки используют синтаксис AWS S3. В этом случае вы должны установить `TYPE` как `S3` и `LOCATIONS` как S3-совместимое расположение хранения.

```
"enabled" = "{ true | false }",

-- Например: us-east1
"aws.s3.region" = "<region>",

-- Например: https://storage.googleapis.com
"aws.s3.endpoint" = "<endpoint_url>",

"aws.s3.access_key" = "<access_key>",
"aws.s3.secret_key" = "<secrete_key>"
```

HDFS

- Если вы не используете аутентификацию для доступа к HDFS, установите следующие свойства:

```
"enabled" = "{ true | false }"
```

- Если вы используете простую аутентификацию (поддерживается с v1.5.2) для доступа к HDFS, установите следующие свойства:

```
"enabled" = "{ true | false }",
"hadoop.security.authentication" = "simple",
"username" = "<hdfs_username>"
```

- Если вы используете аутентификацию Kerberos Ticket Cache (поддерживается с v1.5.2) для доступа к HDFS, установите следующие свойства:

```
"enabled" = "{ true | false }",
"hadoop.security.authentication" = "kerberos",
"hadoop.security.kerberos.ticket.cache.path" = "<ticket_cache_path>"
```

> **ВНИМАНИЕ**
>
> - Эта настройка только заставляет систему использовать KeyTab для доступа к HDFS через Kerberos. Убедитесь, что каждый узел BE или CN имеет доступ к файлам KeyTab. Также убедитесь, что файл **/etc/krb5.conf** настроен правильно.
> - Ticket cache генерируется внешним инструментом kinit. Убедитесь, что у вас есть crontab или аналогичная периодическая задача для обновления тикетов.

- Если в вашем cluster HDFS включена конфигурация NameNode HA (поддерживается с v1.5.2), дополнительно установите следующие свойства:

```
"dfs.nameservices" = "<ha_cluster_name>",
"dfs.ha.namenodes.<ha_cluster_name>" = "<NameNode1>,<NameNode2> [, ...]",
"dfs.namenode.rpc-address.<ha_cluster_name>.<NameNode1>" = "<hdfs_host>:
<hdfs_port>",
"dfs.namenode.rpc-address.<ha_cluster_name>.<NameNode2>" = "<hdfs_host>:
<hdfs_port>",
[...]
"dfs.client.failover.proxy.provider.<ha_cluster_name>" =
"org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvide
```

Подробнее см. в документации HDFS HA.

- Если вы используете WebHDFS (поддерживается с v1.5.2), установите следующие свойства:

```
"enabled" = "{ true | false }"
```

Подробнее см. в документации WebHDFS.

- Если вы используете Hadoop ViewFS (поддерживается с v1.5.2), установите следующие свойства:

```
-- Замените <ViewFS_cluster> на имя cluster ViewFS.
"fs.viewfs.mounttable.<ViewFS_cluster>.link./<viewfs_path_1>" =
"hdfs://<hdfs_host_1>:<hdfs_port_1>/<hdfs_path_1>",
"fs.viewfs.mounttable.<ViewFS_cluster>.link./<viewfs_path_2>" =
"hdfs://<hdfs_host_2>:<hdfs_port_2>/<hdfs_path_2>",
[, ...]
```

Подробнее см. в документации ViewFS.

Функции

Partitioned Prefix

Начиная с v1.5.2, Selena поддерживает создание томов хранения с функцией Partitioned Prefix для S3-совместимых систем объектного хранения. Когда эта функция включена, Selena сохраняет данные в несколько равномерно префиксированных partition (подпутей) в бакете. Это может легко увеличить производительность чтения и записи Selena для файлов данных, хранящихся в бакете, потому что лимит QPS или пропускной способности бакета установлен для каждой partition.

Чтобы включить эту функцию, установите следующие свойства в дополнение к приведённым выше параметрам учётных данных:

```
"aws.s3.enable_partitioned_prefix" = "{ true | false }",
"aws.s3.num_partitioned_prefix" = "<INT>"
```

> **ПРИМЕЧАНИЕ**
>
> - Функция Partitioned Prefix поддерживается только для S3-совместимых систем объектного хранения, то есть `TYPE` тома хранения должен быть `S3`.
> - `LOCATIONS` тома хранения должен содержать только имя бакета, например, `s3://testbucket`. Указание подпути после имени бакета не допускается.
> - Оба свойства неизменяемы после создания тома хранения.
> - Вы не можете включить эту функцию при создании тома хранения с помощью файла конфигурации FE **fe.conf**.

Примеры

Пример 1: Создание тома хранения `my_s3_volume` для бакета AWS S3 `defaultbucket`, использование учётных данных на основе IAM-пользователя (Access Key и Secret Key) для доступа к S3 и его включение.

```
CREATE STORAGE VOLUME my_s3_volume
TYPE = S3
LOCATIONS = ("s3://defaultbucket/test/")
PROPERTIES
(
    "aws.s3.region" = "us-west-2",
    "aws.s3.endpoint" = "https://s3.us-west-2.amazonaws.com",
    "aws.s3.use_aws_sdk_default_behavior" = "false",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.access_key" = "xxxxxxxxxx",
    "aws.s3.secret_key" = "yyyyyyyyyy"
);
```

Пример 2: Создание тома хранения `my_hdfs_volume` для HDFS и его включение.

```
CREATE STORAGE VOLUME my_hdfs_volume
TYPE = HDFS
LOCATIONS = ("hdfs://127.0.0.1:9000/sr/test/")
PROPERTIES
(
    "enabled" = "true"
);
```

Пример 3: Создание тома хранения `hdfsvolumehadoop` для HDFS с использованием простой аутентификации.

```
CREATE STORAGE VOLUME hdfsvolumehadoop
TYPE = HDFS
LOCATIONS = ("hdfs://127.0.0.1:9000/sr/test/")
PROPERTIES(
    "hadoop.security.authentication" = "simple",
    "username" = "selena"
);
```

Пример 4: Использование аутентификации Kerberos Ticket Cache для доступа к HDFS и создание тома хранения `hdfsvolkerberos`.

```
CREATE STORAGE VOLUME hdfsvolkerberos
TYPE = HDFS
LOCATIONS = ("hdfs://127.0.0.1:9000/sr/test/")
PROPERTIES(
    "hadoop.security.authentication" = "kerberos",
    "hadoop.security.kerberos.ticket.cache.path" =
"/path/to/ticket/cache/path"
);
```

Пример 5: Создание тома хранения `hdfsvolha` для cluster HDFS с включённой конфигурацией NameNode HA.

```
CREATE STORAGE VOLUME hdfsvolha
TYPE = HDFS
LOCATIONS = ("hdfs://myhacluster/data/sr")
PROPERTIES(
    "dfs.nameservices" = "myhacluster",
    "dfs.ha.namenodes.myhacluster" = "nn1,nn2,nn3",
    "dfs.namenode.rpc-address.myhacluster.nn1" =
"machine1.example.com:8020",
    "dfs.namenode.rpc-address.myhacluster.nn2" =
"machine2.example.com:8020",
    "dfs.namenode.rpc-address.myhacluster.nn3" =
"machine3.example.com:8020",
    "dfs.namenode.http-address.myhacluster.nn1" =
"machine1.example.com:9870",
    "dfs.namenode.http-address.myhacluster.nn2" =
"machine2.example.com:9870",
    "dfs.namenode.http-address.myhacluster.nn3" =
"machine3.example.com:9870",
    "dfs.client.failover.proxy.provider.myhacluster" =
"org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider"
);
```

Пример 6: Создание тома хранения `webhdfsvol` для WebHDFS.

```
CREATE STORAGE VOLUME webhdfsvol
TYPE = HDFS
LOCATIONS = ("webhdfs://namenode:9870/data/sr");
```

Пример 7: Создание тома хранения `viewfsvol` с использованием Hadoop ViewFS.

```
CREATE STORAGE VOLUME viewfsvol
TYPE = HDFS
LOCATIONS = ("viewfs://clusterX/data/sr")
PROPERTIES(
    "fs.viewfs.mounttable.clusterX.link./data" = "hdfs://nn1-
clusterx.example.com:8020/data",
    "fs.viewfs.mounttable.clusterX.link./project" = "hdfs://nn2-
clusterx.example.com:8020/project"
);
```

Пример 8: Создание тома хранения `adls2` для Azure Data Lake Storage Gen2 с использованием SAS-токена.

```
CREATE STORAGE VOLUME adls2
TYPE = ADLS2
LOCATIONS = ("adls2://testfilesystem/selena")
PROPERTIES (
    "azure.adls2.endpoint" = "https://test.dfs.core.windows.net",
    "azure.adls2.sas_token" = "xxx"
);
```

Пример 9: Создание тома хранения `gs` для Google Storage с использованием имперсонированного Service Account.

```
CREATE STORAGE VOLUME gs
TYPE = GS
LOCATIONS = ("gs://testbucket/selena")
PROPERTIES (
    "gcp.gcs.use_compute_engine_service_account" = "false",
    "gcp.gcs.service_account_email" = "user@hello.iam.gserviceaccount.com",
    "gcp.gcs.service_account_private_key_id" =
"61d257bd847xxxxxxxxxxxxxxx4f0b9b6b9ca07af3b7ea",
    "gcp.gcs.service_account_private_key" = "-----BEGIN PRIVATE KEY----
xxxx-----END PRIVATE KEY-----\n",
    "gcp.gcs.impersonation_service_account" =
"admin@hello.iam.gserviceaccount.com"
);
```

Связанные SQL-команды

- ALTER STORAGE VOLUME
- DROP STORAGE VOLUME
- SET DEFAULT STORAGE VOLUME
- DESC STORAGE VOLUME
- SHOW STORAGE VOLUMES

DESC STORAGE VOLUME

DESC STORAGE VOLUME описывает том хранения. Эта функция поддерживается начиная с v1.5.2.

> **ВНИМАНИЕ**
>
> Только пользователи с привилегией USAGE на определённый том хранения могут выполнять эту операцию.

Синтаксис

```
DESC[RIBE] STORAGE VOLUME <storage_volume_name>
```

Параметры

Параметр	Описание
storage_volume_name	Название тома хранения для описания.

Возвращаемые данные

Возвращаемое значение	Описание
Name	Название тома хранения.
Type	Тип удалённой системы хранения. Допустимые значения: `S3` и `AZBLOB`.
IsDefault	Является ли том хранения томом хранения по умолчанию.
Location	Расположение удалённой системы хранения.
Params	Информация об учётных данных для доступа к удалённой системе хранения.
Enabled	Включён ли том хранения.
Comment	Комментарий к тому хранения.

Примеры

Пример 1: Описание тома хранения `my_s3_volume`.

```
MySQL > DESCRIBE STORAGE VOLUME my_s3_volume\G
*************************** 1. row ***************************
     Name: my_s3_volume
     Type: S3
IsDefault: false
 Location: s3://defaultbucket/test/
   Params:
{"aws.s3.access_key":"xxxxxxxxxx","aws.s3.secret_key":"yyyyyyyyyy","aws.s3.en
west-2.amazonaws.com","aws.s3.region":"us-west-
2","aws.s3.use_instance_profile":"true","aws.s3.use_aws_sdk_default_behavior"
  Enabled: false
  Comment:
1 row in set (0.00 sec)
```

Связанные SQL-команды

- CREATE STORAGE VOLUME
- ALTER STORAGE VOLUME
- DROP STORAGE VOLUME
- SET DEFAULT STORAGE VOLUME
- SHOW STORAGE VOLUMES

DROP STORAGE VOLUME

DROP STORAGE VOLUME удаляет том хранения. Удалённые тома хранения больше не могут использоваться. Эта функция поддерживается начиная с v1.5.2.

> **ВНИМАНИЕ**
>
> - Только пользователи с привилегией DROP на определённый том хранения могут выполнять эту операцию.
> - Том хранения по умолчанию и встроенный том хранения `builtin_storage_volume` не могут быть удалены. Вы можете использовать DESC STORAGE VOLUME, чтобы проверить, является ли том хранения томом по умолчанию.
> - Тома хранения, на которые ссылаются существующие базы данных или cloud-native таблицы, не могут быть удалены.

Синтаксис

```
DROP STORAGE VOLUME [ IF EXISTS ] <storage_volume_name>
```

Параметры

Параметр	Описание
storage_volume_name	Название тома хранения для удаления.

Примеры

Пример 1: Удаление тома хранения `my_s3_volume`.

```
MySQL > DROP STORAGE VOLUME my_s3_volume;
Query OK, 0 rows affected (0.01 sec)
```

Связанные SQL-команды

- CREATE STORAGE VOLUME
- ALTER STORAGE VOLUME
- SET DEFAULT STORAGE VOLUME
- DESC STORAGE VOLUME
- SHOW STORAGE VOLUMES

SET DEFAULT STORAGE VOLUME

SET DEFAULT STORAGE VOLUME устанавливает том хранения в качестве тома хранения по умолчанию. После создания тома хранения для внешнего источника данных вы можете установить его в качестве тома хранения по умолчанию для вашего cluster Selena. Эта функция поддерживается начиная с v1.5.2.

> **ВНИМАНИЕ**
>
> - Только пользователи с привилегией USAGE на определённый том хранения могут выполнять эту операцию.
> - Том хранения по умолчанию не может быть удалён или отключён.
> - Вы должны установить том хранения по умолчанию для cluster Selena с разделяемыми данными, поскольку Selena хранит системную статистическую информацию в томе хранения по умолчанию.

Синтаксис

```
SET <storage_volume_name> AS DEFAULT STORAGE VOLUME
```

Параметры

Параметр	Описание
storage_volume_name	Название тома хранения, который будет установлен в качестве тома хранения по умолчанию.

Примеры

Пример 1: Установка тома хранения `my_s3_volume` в качестве тома хранения по умолчанию.

```
MySQL > SET my_s3_volume AS DEFAULT STORAGE VOLUME;
Query OK, 0 rows affected (0.01 sec)
```

Связанные SQL-команды

- CREATE STORAGE VOLUME
- ALTER STORAGE VOLUME
- DROP STORAGE VOLUME
- DESC STORAGE VOLUME
- SHOW STORAGE VOLUMES

SHOW STORAGE VOLUMES

SHOW STORAGE VOLUMES показывает тома хранения в вашем cluster Selena. Эта функция поддерживается начиная с v1.5.2.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции не требуются привилегии.

Синтаксис

```
SHOW STORAGE VOLUMES [ LIKE '<pattern>' ]
```

Параметры

Параметр	Описание
pattern	Шаблон для сопоставления томов хранения.

Возвращаемые данные

Возвращаемое значение	Описание
Storage Volume	Название тома хранения.

Примеры

Пример 1: Показать все тома хранения в cluster Selena.

```
MySQL > SHOW STORAGE VOLUMES;
+----------------+
| Storage Volume |
+----------------+
| my_s3_volume   |
+----------------+
1 row in set (0.01 sec)
```

Связанные SQL-команды

- CREATE STORAGE VOLUME
- ALTER STORAGE VOLUME
- DROP STORAGE VOLUME
- SET DEFAULT STORAGE VOLUME
- DESC STORAGE VOLUME

Tablet and replica

ADMIN CANCEL REPAIR

ADMIN CANCEL REPAIR используется для отмены приоритетного восстановления указанных таблиц или partition.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия OPERATE на уровне SYSTEM. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис

```
ADMIN CANCEL REPAIR TABLE table_name[ PARTITION (p1,...)]
```

Примечание

> Эта команда только указывает, что система больше не будет приоритетно восстанавливать replica указанных таблиц или partition. Система продолжит восстанавливать эти копии в рамках обычного планирования.

Примеры

1. Отмена приоритетного восстановления

```
ADMIN CANCEL REPAIR TABLE tbl PARTITION(p1);
```

ADMIN CHECK TABLET

ADMIN CHECK TABLET используется для проверки группы tablet.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия OPERATE на уровне SYSTEM. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис

```
ADMIN CHECK TABLE (tablet_id1, tablet_id2, ...)
PROPERTIES("type" = "...")
```

Примечание:

1. Необходимо указать ID tablet и свойство "type" в PROPERTIES.

2. В настоящее время "type" поддерживает только:

   Consistency: Проверка согласованности replica tablet. Эта команда асинхронная. После её отправки Selena начнёт проверку согласованности между

соответствующими tablet. Окончательные результаты будут показаны в столбце InconsistentTabletNum в результате команды SHOW PROC "/statistic".

Примеры

1. Проверка согласованности replica для указанной группы tablet

```
ADMIN CHECK TABLET (10000, 10001)
PROPERTIES("type" = "consistency");
```

ADMIN REPAIR

ADMIN REPAIR используется для приоритетного восстановления указанных таблиц или partition.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия OPERATE на уровне SYSTEM. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис

```
ADMIN REPAIR TABLE table_name[ PARTITION (p1,...)]
```

Примечание:

1. Эта команда только означает, что система попытается приоритетно восстановить replica указанных таблиц или partition без гарантии успешного восстановления. Пользователи могут просматривать статус восстановления с помощью команды ADMIN SHOW REPLICA STATUS.
2. Таймаут по умолчанию составляет 14400 секунд (4 часа). Таймаут означает, что система перестанет приоритетно восстанавливать replica указанных таблиц или partition. В случае таймаута необходимо повторно использовать команду для желаемых настроек.

Примеры

1. Попытка восстановить указанные таблицы

```
ADMIN REPAIR TABLE tbl1;
```

2. Попытка восстановить указанные partition

```
ADMIN REPAIR TABLE tbl1 PARTITION (p1, p2);
```

ADMIN SET PARTITION VERSION

ADMIN SET PARTITION VERSION устанавливает для partition определённую версию данных.

Обратите внимание, что ручная установка версии partition является высокорисковой операцией и рекомендуется только при возникновении проблем с метаданными cluster. В обычных условиях версия partition согласована с версией tablet внутри неё.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия OPERATE на уровне SYSTEM. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис

```
ADMIN SET TABLE <table_name> PARTITION ( <partition_name> | <partition_id>
)
VERSION TO <version>
```

Параметры

- `table_name`: Название таблицы, которой принадлежит partition.
- `partition_name`: Название partition. Вы должны указать partition, используя либо `partition_name`, либо `partition_id`. Для непартиционированных таблиц `partition_name` совпадает с названием таблицы.
- `partition_id`: ID partition. Вы должны указать partition, используя либо `partition_name`, либо `partition_id`. Для таблиц со стратегией случайного бакетирования вы можете использовать только `partition_id` для указания partition.
- `version`: Версия, которую вы хотите установить для partition.

Примеры

1. Установить версию непартиционированной таблицы `t1` в `10`.

```
ADMIN SET TABLE t1 PARTITION(t1) VERSION TO 10;
```

2. Установить версию partition `p1` в таблице `t2` в `10`.

```
ADMIN SET TABLE t2 PARTITION(p1) VERSION TO 10;
```

3. Установить версию partition с ID `123456` в `10`. `t3` — таблица со стратегией случайного бакетирования.

```
ADMIN SET TABLE t3 PARTITION('123456') VERSION TO 10;
```

ADMIN SET REPLICA STATUS

ADMIN SET REPLICA STATUS используется для установки статуса указанных replica.

Эта команда в настоящее время используется только для ручной установки статуса некоторых replica в BAD или OK, позволяя системе автоматически восстанавливать эти replica.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия OPERATE на уровне SYSTEM. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис

```
ADMIN SET REPLICA STATUS
PROPERTIES ("key" = "value", ...)
```

В настоящее время поддерживаются следующие атрибуты:

"table_id": обязательный. Указывает ID tablet.

"backend_id": обязательный. Указывает ID узла BE.

"status": обязательный. Указывает статус. В настоящее время поддерживаются только "bad" и "ok".

Если указанная replica не существует или её статус уже bad, replica будет проигнорирована.

Примечание:

Replica, установленные в статус Bad, могут быть немедленно удалены, действуйте с осторожностью.

Примеры

1. Установить статус replica tablet 10003 на BE 10001 в bad.

```
ADMIN SET REPLICA STATUS PROPERTIES("tablet_id" = "10003", "backend_id"
= "10001", "status" = "bad");
```

2. Установить статус replica tablet 10003 на BE 10001 в ok.

```
ADMIN SET REPLICA STATUS PROPERTIES("tablet_id" = "10003", "backend_id"
= "10001", "status" = "ok");
```

ADMIN SHOW REPLICA DISTRIBUTION

ADMIN SHOW REPLICA DISTRIBUTION используется для отображения статуса распределения replica таблицы или partition.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия OPERATE на уровне SYSTEM. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис

```
ADMIN SHOW REPLICA DISTRIBUTION FROM [db_name.]tbl_name [PARTITION (p1,
...)]
```

Примечание:

Столбец Graph в результате показывает соотношение распределения replica в графическом виде.

Примеры

1. Просмотр распределения replica таблиц

```
ADMIN SHOW REPLICA DISTRIBUTION FROM tbl1;
```

2. Просмотр распределения replica partition в таблице

```
ADMIN SHOW REPLICA DISTRIBUTION FROM db1.tbl1 PARTITION(p1, p2);
```

ADMIN SHOW REPLICA STATUS

ADMIN SHOW REPLICA STATUS используется для отображения статуса replica для таблицы или partition.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия OPERATE на уровне SYSTEM. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис

```
ADMIN SHOW REPLICA STATUS FROM [db_name.]tbl_name [PARTITION (p1, ...)]
[where_clause]
```

```
where_clause:
WHERE STATUS [!]= "replica_status"
```

```
replica_status:
OK:            Replica исправна
DEAD:          Backend replica недоступен
VERSION_ERROR: Версия данных replica отсутствует
SCHEMA_ERROR:  Schema hash replica некорректен
MISSING:       Replica не существует
```

Примеры

1. Просмотр статуса всех replica таблицы.

```
ADMIN SHOW REPLICA STATUS FROM db1.tbl1;
```

2. Просмотр replica partition со статусом VERSION_ERROR.

```
ADMIN SHOW REPLICA STATUS FROM tbl1 PARTITION (p1, p2)
WHERE STATUS = "VERSION_ERROR";
```

3. Просмотр всех неисправных replica таблицы.

```
ADMIN SHOW REPLICA STATUS FROM tbl1
WHERE STATUS != "OK";
```

SHOW TABLE STATUS

SHOW TABLE STATUS используется для просмотра части информации о таблице.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции не требуются привилегии.

Синтаксис

```
SHOW TABLE STATUS
[FROM db] [LIKE "pattern"]
```

> Примечание
>
> Этот оператор в основном совместим с синтаксисом MySQL. В настоящее время он показывает лишь небольшую часть информации, например Comment.

Примеры

1. Просмотр всей информации о таблицах в текущей базе данных.

   ```
   SHOW TABLE STATUS;
   ```

2. Просмотр всей информации о таблицах, имена которых содержат example, в указанной базе данных.

   ```
   SHOW TABLE STATUS FROM db LIKE "%example%";
   ```

File

CREATE FILE

Вы можете выполнить команду CREATE FILE для создания файла. После создания файл загружается и сохраняется в Selena. В базе данных только администратор может создавать и удалять файлы, а все пользователи, имеющие права доступа к базе данных, могут использовать файлы, принадлежащие этой базе данных.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия FILE на уровне SYSTEM. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Основные понятия

File: файл, который создается и сохраняется в Selena. После создания и сохранения файла в Selena система присваивает файлу уникальный ID. Вы можете найти файл по имени базы данных, каталогу и имени файла.

Синтаксис

```
CREATE FILE "file_name" [IN database]
[properties]
```

Параметры

Параметр	Обязательный	Описание
file_name	Да	Имя файла.
database	Нет	База данных, которой принадлежит файл. Если этот параметр не указан, по умолчанию используется имя базы данных, к которой вы обращаетесь в текущей сессии.
properties	Да	Свойства файла. В следующей таблице описаны параметры конфигурации свойств.

Параметры конфигурации `properties`

Параметр конфигурации	Обязательный	Описание
url	Да	URL-адрес для загрузки файла. Поддерживается только HTTP URL без аутентификации. После сохранения файла в Selena URL больше не нужен.
catalog	Да	Категория, к которой принадлежит файл. Вы можете указать каталог в соответствии с вашими бизнес-требованиями. Однако в некоторых ситуациях необходимо установить этот параметр в определённый каталог. Например, при загрузке данных из Kafka Selena ищет файлы в каталоге из источника данных Kafka.
MD5	Нет	Алгоритм вычисления контрольной суммы для проверки файла. Если вы укажете этот параметр, Selena проверит файл после его загрузки.

Примеры

- Создать файл с именем **test.pem** в категории kafka.

```
CREATE FILE "test.pem"
PROPERTIES
(
    "url" = "https://selena-public.oss-cn-xxxx.aliyuncs.com/key/test.pem",
    "catalog" = "kafka"
);
```

- Создать файл с именем **client.key** в категории my_catalog.

```
CREATE FILE "client.key"
IN my_database
PROPERTIES
(
    "url" = "http://test.bj.bcebos.com/kafka-key/client.key",
    "catalog" = "my_catalog",
    "md5" = "b5bb901bf10f99205b39a46ac3557dd9"
);
```

DROP FILE

Используйте команду DROP FILE для удаления файла. При использовании этой команды файл удаляется как из памяти frontend (FE), так и из Berkeley DB Java Edition (BDBJE).

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия FILE на уровне SYSTEM. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис

```
DROP FILE "file_name" [FROM database]
[properties]
```

Параметры

Параметр	Обязательный	Описание
file_name	Да	Имя файла.
database	Нет	База данных, которой принадлежит файл.

Параметр	Обязательный	Описание
properties	Да	Свойства файла. В следующей таблице описаны параметры конфигурации свойств.

Параметры конфигурации `properties`

Параметры конфигурации	Обязательный	Описание
catalog	Да	Категория, к которой принадлежит файл.

Примеры

Удалить файл с именем **ca.pem**.

```
DROP FILE "ca.pem" properties("catalog" = "kafka");
```

SHOW FILE

Вы можете выполнить команду SHOW FILE для просмотра информации о файле, хранящемся в базе данных.

> **ПОДСКАЗКА**
>
> Пользователи с любой привилегией на базу данных, которой принадлежит файл, могут выполнять эту операцию. Если файл принадлежит базе данных, все пользователи, имеющие доступ к этой базе данных, могут использовать этот файл.

Синтаксис

```
SHOW FILE [FROM database]
```

Информация о файле, возвращаемая этой командой:

- `FileId`: глобально уникальный ID файла.
- `DbName`: база данных, которой принадлежит файл.
- `Catalog`: категория, к которой принадлежит файл.

- `FileName`: имя файла.
- `FileSize`: размер файла. Единица измерения — байты.
- `MD5`: алгоритм вычисления контрольной суммы для проверки файла.

Примеры

Просмотр файла, хранящегося в `my_database`.

```
SHOW FILE FROM my_database;
```

SQL blacklist

ADD SQLBLACKLIST

ADD SQLBLACKLIST добавляет регулярное выражение в чёрный список SQL для запрета определённых шаблонов SQL. Когда функция чёрного списка SQL включена, Selena сравнивает все выполняемые SQL-запросы с регулярными выражениями SQL в чёрном списке. Selena не выполняет SQL-запросы, соответствующие любому регулярному выражению в чёрном списке, и возвращает ошибку. Это предотвращает запуск определённых SQL-запросов, которые могут вызвать сбои cluster или непредвиденное поведение.

Подробнее о чёрном списке SQL см. в Управление чёрным списком SQL.

ПОДСКАЗКА

- Для выполнения этой операции требуется привилегия BLACKLIST на уровне SYSTEM. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.
- Чёрный список предназначен только для запросов SELECT, запросов INSERT (с версии v1.5.2) и запросов CTAS (с версии v1.5.2).

Синтаксис

```
ADD SQLBLACKLIST "<sql_reg_expr>"
```

Параметр

`sql_reg_expr`: регулярное выражение, используемое для указания определённого шаблона SQL. Чтобы различать специальные символы в SQL-запросе и в регулярном выражении, необходимо использовать escape-символ `\` в качестве префикса для специальных символов в SQL-запросе, таких как `(`, `)` и `+`. Поскольку `(` и `)` часто используются в SQL-запросах, Selena может распознавать `(` и `)` в SQL-запросах напрямую. Вам не нужно использовать escape-символ для `(` и `)`.

Примеры

Пример 1: Добавление `count(\*)` в чёрный список SQL.

```
mysql> ADD SQLBLACKLIST "select count(\\*) from .+";
```

Пример 2: Добавление `count(distinct )` в чёрный список SQL.

```
mysql> ADD SQLBLACKLIST "select count(distinct .+) from .+";
```

Пример 3: Добавление `order by limit x, y, 1 <= x <=7, 5 <=y <=7` в чёрный список SQL.

```
mysql> ADD SQLBLACKLIST "select id_int from test_all_type_select1
    order by id_int
    limit [1-7], [5-7]";
```

Пример 4: Добавление сложного регулярного выражения SQL в чёрный список SQL. Этот пример демонстрирует использование escape-символов для `*` и `-` в SQL-запросах.

```
mysql> ADD SQLBLACKLIST
    "select id_int \\* 4, id_tinyint, id_varchar
        from test_all_type_nullable
    except select id_int, id_tinyint, id_varchar
        from test_basic
    except select (id_int \\* 9 \\- 8) \\/ 2, id_tinyint, id_varchar
        from test_all_type_nullable2
    except select id_int, id_tinyint, id_varchar
        from test_basic_nullable";
```

Пример 5: Запрет всех запросов INSERT INTO:

```
ADD SQLBLACKLIST "(?i)^insert\\s+into\\s+.*";
```

Пример 6: Запрет всех запросов INSERT INTO ... VALUES:

```
ADD SQLBLACKLIST "(?i)^insert\\s+into\\s+.*values\\s*\\(";
```

Пример 7: Запрет всех запросов INSERT INTO ... VALUES, кроме запросов к системному представлению `_statistics_.column_statistics`:

```
ADD SQLBLACKLIST "(?i)^insert\\s+into\\s+
(?!column_statistics\\b).*values\\s*\\(";
```

DELETE SQLBLACKLIST

DELETE SQLBLACKLIST удаляет регулярное выражение SQL из чёрного списка SQL.

Подробнее о чёрном списке SQL см. в Управление чёрным списком SQL.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия BLACKLIST на уровне SYSTEM. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис

```
DELETE SQLBLACKLIST <sql_index_number>
```

Параметр

`sql_index_number`: номер индекса регулярного выражения SQL в чёрном списке. Несколько номеров индексов разделяются запятой (,) и пробелом. Получить номер индекса можно с помощью SHOW SQLBLACKLIST.

Примеры

```
mysql> DELETE SQLBLACKLIST 3, 4;

mysql> SHOW SQLBLACKLIST;
+-------+-----------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----+
| Index | Forbidden SQL
|
+-------+-----------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----+
| 1     | select count\(\*\) from .+
|
| 2     | select id_int \* 4, id_tinyint, id_varchar from
test_all_type_nullable except select id_int, id_tinyint, id_varchar from
test_basic except select \(id_int \* 9 \- 8\) \/ 2, id_tinyint, id_varchar
from test_all_type_nullable2 except select id_int, id_tinyint, id_varchar
from test_basic_nullable |
+-------+-----------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----+
```

SHOW SQLBLACKLIST

SHOW SQLBLACKLIST показывает регулярные выражения SQL в чёрном списке SQL.

Подробнее о чёрном списке SQL см. в Управление чёрным списком SQL.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия BLACKLIST на уровне SYSTEM. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис

```
SHOW SQLBLACKLIST
```

Возвращаемые данные

Возвращаемое значение	Описание
Index	Номер индекса регулярного выражения SQL, добавленного в чёрный список.
Forbidden SQL	Регулярное выражение SQL, добавленное в чёрный список.

Примеры

```
mysql> SHOW SQLBLACKLIST;
+-------+-------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----+
| Index | Forbidden SQL
|
+-------+-------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----+
| 1     | select count\(\*\) from .+
|
| 2     | select id_int \* 4, id_tinyint, id_varchar from
test_all_type_nullable except select id_int, id_tinyint, id_varchar from
test_basic except select \(id_int \* 9 \- 8\) \/ 2, id_tinyint, id_varchar
from test_all_type_nullable2 except select id_int, id_tinyint, id_varchar
from test_basic_nullable |
| 3     | select id_int from test_all_type_select1 order by id_int limit
[1-7], [5-7]
|
| 4     | select count\(distinct .+\) from .+
|
+-------+-------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----+
```

Plugin

INSTALL PLUGIN

INSTALL PLUGIN используется для установки плагина.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия PLUGIN на уровне SYSTEM.
> Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис

```
INSTALL PLUGIN FROM [source] [PROPERTIES ("key"="value", ...)]
```

Поддерживаются 3 типа источников:

```
1. Абсолютный путь к zip-файлу
2. Абсолютный путь к директории плагина
3. HTTP или HTTPS ссылка для скачивания zip-файла
```

PROPERTIES поддерживает установку некоторых конфигураций плагинов, например установку значения md5sum для zip-файла и т.д.

Примеры

1. Установка плагина из локального zip-файла:

```
INSTALL PLUGIN FROM "/home/users/selena/auditdemo.zip";
```

2. Установка плагина из локальной директории:

```
INSTALL PLUGIN FROM "/home/users/selena/auditdemo/";
```

3. Скачивание и установка плагина:

```
INSTALL PLUGIN FROM "http://mywebsite.com/plugin.zip";
```

4. Скачивание и установка плагина с указанием значения md5sum для zip-файла:

```
INSTALL PLUGIN FROM "http://mywebsite.com/plugin.zip"
PROPERTIES("md5sum" = "73877f6029216f4314d712086a146570");
```

SHOW PLUGINS

SHOW PLUGINS используется для просмотра установленных плагинов.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия PLUGIN на уровне SYSTEM. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис

```
SHOW PLUGINS
```

Эта команда отобразит все встроенные и пользовательские плагины.

Примеры

1. Показать установленные плагины:

```
SHOW PLUGINS;
```

UNINSTALL PLUGIN

UNINSTALL PLUGIN используется для удаления плагина.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия PLUGIN на уровне SYSTEM. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис

```
UNINSTALL PLUGIN <plugin_name>
```

plugin_name можно просмотреть с помощью команды SHOW PLUGINS.

Удалить можно только не встроенные плагины.

Примеры

1. Удаление плагина:

```
UNINSTALL PLUGIN auditdemo;
```

Execution plan and profile

ANALYZE PROFILE

ANALYZE PROFILE анализирует определённый профиль запроса на основе фрагментов и отображает его в древовидной структуре. Для получения дополнительной информации о профиле запроса см. Обзор профиля запроса.

Эта функция поддерживается начиная с версии v1.5.2.

> **ВНИМАНИЕ**
>
> Только пользователи с привилегией SELECT на определённую таблицу могут выполнять эту операцию.

Синтаксис

```
ANALYZE PROFILE FROM '<query_id>', [<plan_node_id>[, ...] ]
```

Параметры

Параметр	Описание
query_id	ID запроса. Вы можете получить его с помощью SHOW PROFILELIST.
plan_node_id	ID узла плана в профиле. Вы можете указать этот параметр для просмотра детальных метрик соответствующего узла (узлов) плана. Если этот параметр не указан, отображаются только сводные метрики всех узлов плана.

Примеры

Пример 1: Запрос профиля запроса без указания ID узла.

```
MySQL root@172.26.81.178:tpch> analyze profile from '35fc9ef0-8442-11ee-844d-00163e0f59de';
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Explain String                                                                                                                                                        |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Summary                                                                                                                                                               |
|     QueryId: 35fc9ef0-8442-11ee-844d-00163e0f59de                                                                                                                     |
|     Version: profile-bba710c                                                                                                                                          |
|     State: Finished                                                                                                                                                   |
|     TotalTime: 26ms                                                                                                                                                   |
|         ExecutionTime: 11.747ms [Scan: 4.489ms (38.21%), Network: 1.874ms (15.95%), ResultDeliverTime: 0ns (0.00%), ScheduleTime: 912.743us (7.77%)]                   |
|         CollectProfileTime: 6ms                                                                                                                                       |
|         FrontendProfileMergeTime: 5.049ms                                                                                                                             |
|     QueryPeakMemoryUsage: 710.030 KB, QueryAllocatedMemoryUsage: 115.099 MB                                                                                           |
|     Top Most Time-consuming Nodes:                                                                                                                                    |
|         1. OLAP_SCAN (id=0) : 5.745ms (68.87%)                                                                                                                        |
|         2. EXCHANGE (id=3) [GATHER]: 1.986ms (23.81%)                                                                                                                 |
|         3. PROJECT (id=1) : 343.661us (4.12%)                                                                                                                         |
|         4. AGGREGATION (id=2) [serialize, update]: 172.182us (2.06%)                                                                                                  |
|         5. AGGREGATION (id=4) [finalize, merge]: 61.624us (0.74%)                                                                                                     |
|         6. RESULT_SINK: 32.340us (0.39%)                                                                                                                              |
|     Top Most Memory-consuming Nodes:                                                                                                                                  |
|         1. OLAP_SCAN (id=0) : 2.525 MB                                                                                                                                |
|         2. EXCHANGE (id=3) [GATHER]: 7.234 KB                                                                                                                         |
|         3. AGGREGATION (id=2) [serialize, update]: 3.671 KB                                                                                                           |
|         4. AGGREGATION (id=4) [finalize, merge]: 2.991 KB                                                                                                             |
|     NonDefaultVariables:                                                                                                                                              |
|         enable_adaptive_sink_dop: false -> true                                                                                                                       |
|         enable_async_profile: true -> false                                                                                                                           |
|         enable_profile: false -> true                                                                                                                                 |
|         prefer_compute_node: false -> true                                                                                                                            |
|         sql_mode_v2: 32 -> 34                                                                                                                                         |
| Fragment 0                                                                                                                                                            |
| │   BackendNum: 1                                                                                                                                                     |
| │   InstancePeakMemoryUsage: 22.194 KB, InstanceAllocatedMemoryUsage: 53.422 KB                                                                                      |
| │   PrepareTime: 599.073us                                                                                                                                            |
| └──RESULT_SINK                                                                                                                                                        |
| │     TotalTime: 32.340us (0.39%) [CPUTime: 32.340us]                                                                                                               |
| │     OutputRows: 1                                                                                                                                                  |
| │     SinkType: MYSQL_PROTOCAL                                                                                                                                       |
| │   └──AGGREGATION (id=4) [finalize, merge]                                                                                                                          |
| │       │   Estimates: [row: 1, cpu: 8.00, memory: 8.00, network: 0.00, cost: 16503378.85]                                                                          |
| │       │   TotalTime: 61.624us (0.74%) [CPUTime: 61.624us]                                                                                                          |
| │       │   OutputRows: 1                                                                                                                                            |
| │       │   PeakMemory: 2.991 KB, AllocatedMemory: 4.632 KB                                                                                                          |
| │       │   AggExprs: [count(17: count)]                                                                                                                             |
| │       │   SubordinateOperators:                                                                                                                                    |
| │       │       LOCAL_EXCHANGE [Passthrough]                                                                                                                         |
| │       └──EXCHANGE (id=3) [GATHER]                                                                                                                                  |
| │           Estimates: [row: 1, cpu: 8.00, memory: 0.00, network: 8.00, cost: 16503358.85]                                                                          |
| │           TotalTime: 1.986ms (23.81%) [CPUTime: 112.324us, NetworkTime: 1.874ms]                                                                                   |
| │           OutputRows: 10                                                                                                                                           |
| │           PeakMemory: 7.234 KB, AllocatedMemory: 60.896 KB                                                                                                         |
| │           Detail Timers:                                                                                                                                           |
| │               OverallTime: 1.290ms [min=1.125ms, max=1.406ms]                                                                                                      |
| │               WaitTime: 743.468us [min=230.363us, max=1.125ms]                                                                                                     |
| │                                                                                                                                                                   |
| Fragment 1                                                                                                                                                            |
| │   BackendNum: 3                                                                                                                                                     |
| │   InstancePeakMemoryUsage: 535.105 KB, InstanceAllocatedMemoryUsage: 115.046 MB                                                                                    |
| │   PrepareTime: 1.055ms                                                                                                                                              |
| └──DATA_STREAM_SINK (id=3)                                                                                                                                            |
| │     PartitionType: UNPARTITIONED                                                                                                                                   |
| │   └──AGGREGATION (id=2) [serialize, update]                                                                                                                        |
| │       │   Estimates: [row: 1, cpu: 3000607.50, memory: 0.80, network: 0.00, cost: 16503342.85]                                                                     |
| │       │   TotalTime: 172.182us (2.06%) [CPUTime: 172.182us]                                                                                                        |
| │       │   OutputRows: 10                                                                                                                                           |
| │       │   PeakMemory: 3.671 KB, AllocatedMemory: 46.327 KB                                                                                                         |
| │       │   AggExprs: [count(*)]                                                                                                                                     |
| │       └──PROJECT (id=1)                                                                                                                                            |
| │           │   Estimates: [row: ?, cpu: ?, memory: ?, network: ?, cost: ?]                                                                                          |
| │           │   TotalTime: 343.661us (4.12%) [CPUTime: 343.661us]                                                                                                    |
| │           │   OutputRows: 6.001M (6001215)                                                                                                                         |
| │           │   Expression: [1]                                                                                                                                      |
| │           └──OLAP_SCAN (id=0)                                                                                                                                       |
| │               Estimates: [row: 6001215, cpu: 30006075.00, memory: 0.00, network: 0.00, cost: 15003037.50]                                                          |
| │               TotalTime: 5.745ms (68.87%) [CPUTime: 1.256ms, ScanTime: 4.489ms]                                                                                     |
| │               OutputRows: 6.001M (6001215)                                                                                                                         |
| │               PeakMemory: 2.525 MB, AllocatedMemory: 108.166 MB                                                                                                    |
| │               Table: : lineitem                                                                                                                                    |
| │               SubordinateOperators:                                                                                                                                |
| │                   NOOP                                                                                                                                           |
| │                   OLAP_SCAN_PREPARE                                                                                                                              |
| │               Detail Timers: [ScanTime = IOTaskExecTime + IOTaskWaitTime]                                                                                         |
| │                   IOTaskExecTime: 3.703ms [min=3.224ms, max=4.434ms]                                                                                             |
| │                       SegmentRead: 3.110ms [min=2.646ms, max=3.808ms]                                                                                            |
| │                           BlockFetch: 2.944ms [min=2.493ms, max=3.616ms]                                                                                         |
| │                   IOTaskWaitTime: 106.066us [min=41.862us, max=484.711us]                                                                                        |
| │                                                                                                                                                                   |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------+
84 rows in set
Time: 0.018s
MySQL root@172.26.81.178:tpch>
```

Пример 2: Запрос профиля запроса с указанием ID узла `0`. Selena возвращает все детальные метрики для Node ID `0` и выделяет метрики с высоким использованием для упрощения выявления проблем.

```
MySQL root@172.26.81.178:tpch> analyze profile from '35fc9ef0-8442-11ee-844d-00163e0f59de', 0;
+-----------------------
```

```
|                                GetDelVec: 0ns                                                                                |
|                                GetDeltaColumnGroup: 6.124us [min=1.295us, max=10.804us]                                      |
|                                GetRowsets: 305ns [min=223ns, max=396ns]                                                      |
|                                IOTime: 160.874us [min=148.729us, max=171.534us]                                              |
|                                ReadPKIndex: 0ns                                                                              |
|                                SegmentInit: 45.173us [min=37.247us, max=51.996us]                                            |
|                                    BitmapIndexFilter: 0ns                                                                    |
|                                    BitmapIndexFilterRows: 0                                                                  |
|                                    BitmapIndexIteratorInit: 974ns [min=810ns, max=1.335us]                                   |
|                                    BloomFilterFilter: 0ns                                                                    |
|                                    BloomFilterFilterRows: 0                                                                  |
|                                    ColumnIteratorInit: 18.537us [min=11.481us, max=24.101us]                                 |
|                                    RemainingRowsAfterShortKeyFilter: 6.001M (6001215) [min=595.164K (595164), max=604.362K (604362)]|
|                                    SegmentRuntimeZoneMapFilterRows: 0                                                        |
|                                    SegmentZoneMapFilterRows: 0                                                               |
|                                    ShortKeyFilter: 1.012us [min=749ns, max=1.348us]                                          |
|                                    ShortKeyFilterRows: 0                                                                     |
|                                    ShortKeyRangeNumber: 0                                                                    |
|                                    ZoneMapIndexFilterRows: 0                                                                 |
|                                    ZoneMapIndexFiter: 727ns [min=528ns, max=1.096us]                                         |
|                                SegmentRead: 3.110ms [min=2.646ms, max=3.808ms]                                               |
|                                    BlockFetch: 2.944ms [min=2.493ms, max=3.616ms]                                            |
|                                    BlockFetchCount: 1.470K (1470) [min=146, max=148]                                         |
|                                    BlockSeek: 122.611us [min=101.321us, max=145.219us]                                       |
|                                    BlockSeekCount: 370 [min=37, max=37]                                                      |
|                                    ChunkCopy: 0ns                                                                            |
|                                    DecompressT: 0ns                                                                          |
|                                    DelVecFilterRows: 0                                                                       |
|                                    PredFilter: 0ns                                                                           |
|                                    PredFilterRows: 0                                                                         |
|                                    RowsetsReadCount: 20 [min=2, max=2]                                                       |
|                                    SegmentsReadCount: 10 [min=1, max=1]                                                      |
|                                    TotalColumnsDataPageCount: 370 [min=37, max=37]                                           |
|                            IOTaskWaitTime: 106.066us [min=41.862us, max=484.711us]                                           |
|                            MorselsCount: 10 [min=1, max=1]                                                                   |
|                            PeakChunkBufferMemoryUsage: 128.000 KB                                                            |
|                            PeakChunkBufferSize: 11                                                                           |
|                            PeakIOTasks: 1                                                                                    |
|                            PeakScanTaskQueueSize: 6 [min=0, max=2]                                                           |
|                            PrepareChunkSourceTime: 169.748us [min=151.592us, max=184.497us]                                  |
|                            PushdownPredicates: 0                                                                             |
|                            RawRowsRead: 6.001M (6001215) [min=595.164K (595164), max=604.362K (604362)]                     |
|                            ReadPagesNum: 370 [min=37, max=37]                                                                |
|                            RowsRead: 6.001M (6001215) [min=595.164K (595164), max=604.362K (604362)]                         |
|                            SubmitTaskCount: 30 [min=3, max=3]                                                                |
|                            SubmitTaskTime: 24.403us [min=14.188us, max=41.869us]                                             |
|                            TabletCount: 10 [min=3, max=4]                                                                    |
|                            UncompressedBytesRead: 5.939 MB [min=604.067 KB, max=611.704 KB]                                  |
|                                                                                                                              |
+------------------------------------------------------------------------------------------------------------------------------+
141 rows in set
Time: 0.025s
MySQL root@172.26.81.178:tpch>
```

Связанные SQL-команды

- SHOW PROFILELIST
- EXPLAIN ANALYZE

EXPLAIN

EXPLAIN показывает логические или физические планы выполнения запроса. Инструкции по анализу плана запроса см. в разделе Анализ плана.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции не требуются привилегии.

Синтаксис

```
EXPLAIN [ LOGICAL | VERBOSE | COSTS ] <query>
```

ПОДСКАЗКА

Уровень детализации возвращаемого плана выполнения: EXPLAIN LOGICAL < EXPLAIN < EXPLAIN VERBOSE < EXPLAIN COSTS.

Начиная с версии v1.5.2, уровень детализации по умолчанию изменён на `COSTS`. Вы можете настроить уровень детализации, возвращаемой командой EXPLAIN, с помощью динамического параметра FE `query_detail_explain_level`.

В большинстве случаев достаточно использовать EXPLAIN. EXPLAIN VERBOSE и EXPLAIN COSTS выводят много внутренней информации, которая в основном используется для отладки плана выполнения.

Параметры

Параметр	Описание
LOGICAL	Показывает упрощённые логические планы выполнения.
VERBOSE	Показывает детальные логические планы выполнения, включая типы данных, информацию о допустимости null-значений и стратегию оптимизации.
COSTS (По умолчанию)	Показывает детальные логические планы выполнения, включая статистику столбцов.
query	Запрос, план выполнения которого вы хотите просмотреть.

ПРИМЕЧАНИЕ

Вы можете настроить уровень детализации, возвращаемой командой EXPLAIN, с помощью динамического параметра FE `query_detail_explain_level`.

Возвращаемое значение

Возвращает план выполнения входного запроса.

Возвращаемое значение	Описание
avgRowSize	Средний размер сканируемых строк данных.
cardinality	Общее количество строк данных в сканируемой таблице.
colocate	Находится ли таблица в режиме colocate.
numNodes	Количество узлов для сканирования.
rollup	Используемое материализованное представление Rollup.

Возвращаемое значение	Описание
preaggregation	Статус предварительной агрегации.
predicates	Предикаты в запросе.
column statistics	Статистика столбца, последовательность: MIN, MAX, количество NULL, средний размер и кардинальность.

Примеры

EXPLAIN

```
MySQL tpcds> explain select
    sr_customer_sk as ctr_customer_sk,
    sr_store_sk as ctr_store_sk,
    sum(SR_RETURN_AMT) as ctr_total_return
  from
    store_returns,
    date_dim
  where
    sr_returned_date_sk = d_date_sk and d_year = 2000
  group by
    sr_customer_sk,
    sr_store_sk
+-----------------------------------------------------------------------------
---+
| Explain String
|
+-----------------------------------------------------------------------------
---+
| PLAN FRAGMENT 0
|
|  OUTPUT EXPRS:5: sr_customer_sk | 9: sr_store_sk | 49: sum
|
|   PARTITION: UNPARTITIONED
|
|
|
|   RESULT SINK
|
|
|
|   9:EXCHANGE
|
|
|
| PLAN FRAGMENT 1
|
|  OUTPUT EXPRS:
|
|   PARTITION: HASH_PARTITIONED: 5: sr_customer_sk, 9: sr_store_sk
|
|
|
```

```
|   STREAM DATA SINK
|
|     EXCHANGE ID: 09
|
|     UNPARTITIONED
|
|
|
|   8:AGGREGATE (merge finalize)
|
|   |  output: sum(49: sum)
|
|   |  group by: 5: sr_customer_sk, 9: sr_store_sk
|
|   |
|
|   7:EXCHANGE
|
|
|
| PLAN FRAGMENT 2
|
|  OUTPUT EXPRS:
|
|   PARTITION: RANDOM
|
|
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 07
|
|     HASH_PARTITIONED: 5: sr_customer_sk, 9: sr_store_sk
|
|
|
|   6:AGGREGATE (update serialize)
|
|   |  STREAMING
|
|   |  output: sum(12: sr_return_amt)
|
|   |  group by: 5: sr_customer_sk, 9: sr_store_sk
|
|   |
|
```

```
|   5:Project
|
|   |  <slot 5> : 5: sr_customer_sk
|
|   |  <slot 9> : 9: sr_store_sk
|
|   |  <slot 12> : 12: sr_return_amt
|
|   |
|
|   4:HASH JOIN
|
|   |  join op: INNER JOIN (BROADCAST)
|
|   |  colocate: false, reason:
|
|   |  equal join conjunct: 3: sr_returned_date_sk = 21: d_date_sk
|
|   |
|
|   |----3:EXCHANGE
|
|   |
|
|   0:OlapScanNode
|
|      TABLE: store_returns
|
|      PREAGGREGATION: ON
|
|      PREDICATES: 3: sr_returned_date_sk IS NOT NULL
|
|      partitions=1/1
|
|      rollup: store_returns
|
|      tabletRatio=10/10
|
|
tabletList=10241,10243,10245,10247,10249,10251,10253,10255,10257,10259 |
|      cardinality=277502
|
|      avgRowSize=20.0
|
|
|
```

```
| PLAN FRAGMENT 3
|
|  OUTPUT EXPRS:
|
|   PARTITION: RANDOM
|
|
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 03
|
|     UNPARTITIONED
|
|
|
|   2:Project
|
|   |  <slot 21> : 21: d_date_sk
|
|   |
|
|   1:OlapScanNode
|
|      TABLE: date_dim
|
|      PREAGGREGATION: ON
|
|      PREDICATES: 27: d_year = 2000
|
|      partitions=1/1
|
|      rollup: date_dim
|
|      tabletRatio=5/5
|
|      tabletList=11543,11545,11547,11549,11551
|
|      cardinality=362
|
|      avgRowSize=8.0
|
+-----------------------------------------------------------------------------
---+
79 rows in set
```

EXPLAIN LOGICAL

```
MySQL tpcds> explain logical select
    sr_customer_sk as ctr_customer_sk,
    sr_store_sk as ctr_store_sk,
    sum(SR_RETURN_AMT) as ctr_total_return
  from
    store_returns,
    date_dim
  where
    sr_returned_date_sk = d_date_sk and d_year = 2000
  group by
    sr_customer_sk,
    sr_store_sk

+-------------------------------------------------------------------------
-------------------------------------------------------+
| Explain String
|
+-------------------------------------------------------------------------
-------------------------------------------------------+
| - Output => [5:sr_customer_sk, 9:sr_store_sk, 49:sum]
|
|     - AGGREGATE(GLOBAL) [5:sr_customer_sk, 9:sr_store_sk]
|
|             Estimates: {row: 49728, cpu: 795660.58, memory: 795660.58,
network: 0.00, cost: 8405375.45}                          |
|             49:sum := sum(49:sum)
|
|         - EXCHANGE(SHUFFLE) [5, 9]
|
|                 Estimates: {row: 49728, cpu: 79566.06, memory: 0.00,
network: 0.00, cost: 6416223.99}                           |
|             - AGGREGATE(LOCAL) [5:sr_customer_sk, 9:sr_store_sk]
|
|                     Estimates: {row: 49728, cpu: 119349.09, memory:
79566.06, network: 0.00, cost: 6376440.96}                  |
|                     49:sum := sum(12:sr_return_amt)
|
|                 - HASH/INNER JOIN [3:sr_returned_date_sk = 21:d_date_sk]
=> [5:sr_customer_sk, 9:sr_store_sk, 12:sr_return_amt] |
|                         Estimates: {row: 49728, cpu: 6744977.39, memory:
1446.51, network: 0.00, cost: 6157634.30}              |
|                     - SCAN [store_returns] => [3:sr_returned_date_sk,
5:sr_customer_sk, 9:sr_store_sk, 12:sr_return_amt]        |
|                             Estimates: {row: 277502, cpu: 5550040.00,
```

```
memory: 0.00, network: 0.00, cost: 2775020.00}            |
|                              partitionRatio: 1/1, tabletRatio: 10/10
|
|                              predicate: 3:sr_returned_date_sk IS NOT NULL
|
|                      - EXCHANGE(BROADCAST)
|
|                              Estimates: {row: 361, cpu: 1446.51, memory:
1446.51, network: 1446.51, cost: 7232.57}               |
|                          - SCAN [date_dim] => [21:d_date_sk]
|
|                                  Estimates: {row: 361, cpu: 2893.03,
memory: 0.00, network: 0.00, cost: 1446.51}                 |
|                                  partitionRatio: 1/1, tabletRatio: 5/5
|
|                                  predicate: 27:d_year = 2000
|
+-------------------------------------------------------------------------
-------------------------------------------------------+
21 rows in set
```

EXPLAIN VERBOSE

```
MySQL tpcds> explain verbose select
    sr_customer_sk as ctr_customer_sk,
    sr_store_sk as ctr_store_sk,
    sum(SR_RETURN_AMT) as ctr_total_return
  from
    store_returns,
    date_dim
  where
    sr_returned_date_sk = d_date_sk and d_year = 2000
  group by
    sr_customer_sk,
    sr_store_sk

+-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
----+
| Explain String
|
+-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
----+
| RESOURCE GROUP: default_wg
|
|
|
| PLAN FRAGMENT 0(F04)
|
|   Fragment Cost: 0.0
|
|   Output Exprs:5: sr_customer_sk | 9: sr_store_sk | 49: sum
|
|   Input Partition: UNPARTITIONED
|
|   RESULT SINK
|
|
|
|   9:EXCHANGE
|
|      cardinality: 49729
|
|
|
| PLAN FRAGMENT 1(F03)
```

```
|
|   Fragment Cost: 2028934.4876271961
|
|
|
|   Input Partition: HASH_PARTITIONED: 5: sr_customer_sk, 9: sr_store_sk
|
|   OutPut Partition: UNPARTITIONED
|
|   OutPut Exchange Id: 09
|
|
|
|   8:AGGREGATE (merge finalize)
|
|   |  aggregate: sum[([49: sum, DECIMAL128(38,2), true]); args: DECIMAL64;
result: DECIMAL128(38,2); args nullable: true; result nullable: true]
|
|   |  group by: [5: sr_customer_sk, INT, true], [9: sr_store_sk, INT,
true]
|
|   |  cardinality: 49729
|
|   |
|
|   7:EXCHANGE
|
|      distribution type: SHUFFLE
|
|      partition exprs: [5: sr_customer_sk, INT, true], [9: sr_store_sk,
INT, true]
|
|      cardinality: 49729
|
|
|
| PLAN FRAGMENT 2(F00)
|
|   Fragment Cost: 6376440.959353934
|
|
|
|   Input Partition: RANDOM
|
|   OutPut Partition: HASH_PARTITIONED: 5: sr_customer_sk, 9: sr_store_sk
|
```

```
|   OutPut Exchange Id: 07
|
|
|
|   6:AGGREGATE (update serialize)
|
|   |  STREAMING
|
|   |  aggregate: sum[([12: sr_return_amt, DECIMAL64(7,2), true]); args:
DECIMAL64; result: DECIMAL128(38,2); args nullable: true; result nullable:
true] |
|   |  group by: [5: sr_customer_sk, INT, true], [9: sr_store_sk, INT,
true]
|
|   |  cardinality: 49729
|
|   |
|
|   5:Project
|
|   |  output columns:
|
|   |  5 <-> [5: sr_customer_sk, INT, true]
|
|   |  9 <-> [9: sr_store_sk, INT, true]
|
|   |  12 <-> [12: sr_return_amt, DECIMAL64(7,2), true]
|
|   |  cardinality: 49729
|
|   |
|
|   4:HASH JOIN
|
|   |  join op: INNER JOIN (BROADCAST)
|
|   |  equal join conjunct: [3: sr_returned_date_sk, INT, true] = [21:
d_date_sk, INT, false]
|
|   |  build runtime filters:
|
|   |  - filter_id = 0, build_expr = (21: d_date_sk), remote = false
|
|   |  output columns: 5, 9, 12
|
|   |  can local shuffle: true
```

```
|
|   |  cardinality: 49729
|
|   |
|
|   |----3:EXCHANGE
|
|   |       distribution type: BROADCAST
|
|   |       cardinality: 362
|
|   |
|
|   0:OlapScanNode
|
|      table: store_returns, rollup: store_returns
|
|      preAggregation: on
|
|      Predicates: 3: sr_returned_date_sk IS NOT NULL
|
|      partitionsRatio=1/1, tabletsRatio=10/10
|
|
tabletList=10241,10243,10245,10247,10249,10251,10253,10255,10257,10259
|
|      actualRows=287514, avgRowSize=20.0
|
|      cardinality: 277502
|
|      probe runtime filters:
|
|      - filter_id = 0, probe_expr = (3: sr_returned_date_sk)
|
|
|
| PLAN FRAGMENT 3(F01)
|
|   Fragment Cost: 1446.5148514851485
|
|
|
|   Input Partition: RANDOM
|
|   OutPut Partition: UNPARTITIONED
|
```

```
|   OutPut Exchange Id: 03
|
|
|
|   2:Project
|
|   |  output columns:
|
|   |  21 <-> [21: d_date_sk, INT, false]
|
|   |  cardinality: 362
|
|   |
|
|   1:OlapScanNode
|
|      table: date_dim, rollup: date_dim
|
|      preAggregation: on
|
|      Predicates: [27: d_year, INT, false] = 2000
|
|      partitionsRatio=1/1, tabletsRatio=5/5
|
|      tabletList=11543,11545,11547,11549,11551
|
|      actualRows=73049, avgRowSize=8.0
|
|      cardinality: 362
|
+-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
----+
```

EXPLAIN COSTS

```
+-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
----+
| Explain String
|
+-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
----+
| PLAN FRAGMENT 0(F04)
|
|   Output Exprs:5: sr_customer_sk | 9: sr_store_sk | 49: sum
|
|   Input Partition: UNPARTITIONED
|
|   RESULT SINK
|
|
|
|   9:EXCHANGE
|
|      cardinality: 49729
|
|
|
| PLAN FRAGMENT 1(F03)
|
|
|
|   Input Partition: HASH_PARTITIONED: 5: sr_customer_sk, 9: sr_store_sk
|
|   OutPut Partition: UNPARTITIONED
|
|   OutPut Exchange Id: 09
|
|
|
|   8:AGGREGATE (merge finalize)
|
|   |  aggregate: sum[([49: sum, DECIMAL128(38,2), true]); args: DECIMAL64;
result: DECIMAL128(38,2); args nullable: true; result nullable: true]
|
|   |  group by: [5: sr_customer_sk, INT, true], [9: sr_store_sk, INT,
true]
|
```

```
|   |  cardinality: 49729
|
|   |  column statistics:
|
|   |  * sr_customer_sk-->[1.0, 100000.0, 2.010867271218169E-5, 4.0,
49728.786461450894] ESTIMATE
|
|   |  * sr_store_sk-->[1.0, 10.0, 0.14285714285714285, 4.0, 6.0] ESTIMATE
|
|   |  * sum-->[0.0, 16917.12, 0.034878301578357924, 8.0,
49728.786461450894] ESTIMATE
|
|   |
|
|   7:EXCHANGE
|
|      distribution type: SHUFFLE
|
|      partition exprs: [5: sr_customer_sk, INT, true], [9: sr_store_sk,
INT, true]
|
|      cardinality: 49729
|
|
|
| PLAN FRAGMENT 2(F00)
|
|
|
|   Input Partition: RANDOM
|
|   OutPut Partition: HASH_PARTITIONED: 5: sr_customer_sk, 9: sr_store_sk
|
|   OutPut Exchange Id: 07
|
|
|
|   6:AGGREGATE (update serialize)
|
|   |  STREAMING
|
|   |  aggregate: sum[([12: sr_return_amt, DECIMAL64(7,2), true]); args:
DECIMAL64; result: DECIMAL128(38,2); args nullable: true; result nullable:
true] |
|   |  group by: [5: sr_customer_sk, INT, true], [9: sr_store_sk, INT,
true]
```

```
|
|   |  cardinality: 49729
|
|   |  column statistics:
|
|   |  * sr_customer_sk-->[1.0, 100000.0, 2.010867271218169E-5, 4.0,
49728.786461450894] ESTIMATE
|
|   |  * sr_store_sk-->[1.0, 10.0, 0.14285714285714285, 4.0, 6.0] ESTIMATE
|
|   |  * sum-->[0.0, 16917.12, 0.034878301578357924, 8.0,
49728.786461450894] ESTIMATE
|
|   |
|
|   5:Project
|
|   |  output columns:
|
|   |  5 <-> [5: sr_customer_sk, INT, true]
|
|   |  9 <-> [9: sr_store_sk, INT, true]
|
|   |  12 <-> [12: sr_return_amt, DECIMAL64(7,2), true]
|
|   |  cardinality: 49729
|
|   |  column statistics:
|
|   |  * sr_customer_sk-->[1.0, 100000.0, 0.034836564480338346, 4.0,
49728.786461450894] ESTIMATE
|
|   |  * sr_store_sk-->[1.0, 10.0, 0.034996556689413386, 4.0, 6.0] ESTIMATE
|
|   |  * sr_return_amt-->[0.0, 16917.12, 0.034878301578357924, 8.0,
49728.786461450894] ESTIMATE
|
|   |
|
|   4:HASH JOIN
|
|   |  join op: INNER JOIN (BROADCAST)
|
|   |  equal join conjunct: [3: sr_returned_date_sk, INT, true] = [21:
d_date_sk, INT, false]
|
```

```
|   |  build runtime filters:
|
|   |  - filter_id = 0, build_expr = (21: d_date_sk), remote = false
|
|   |  output columns: 5, 9, 12
|
|   |  can local shuffle: true
|
|   |  cardinality: 49729
|
|   |  column statistics:
|
|   |  * sr_returned_date_sk-->[2450820.0, 2452822.0, 0.0, 4.0,
361.6287128712871] ESTIMATE
|
|   |  * sr_customer_sk-->[1.0, 100000.0, 0.034836564480338346, 4.0,
49728.786461450894] ESTIMATE
|
|   |  * sr_store_sk-->[1.0, 10.0, 0.034996556689413386, 4.0, 6.0] ESTIMATE
|
|   |  * sr_return_amt-->[0.0, 16917.12, 0.034878301578357924, 8.0,
49728.786461450894] ESTIMATE
|
|   |  * d_date_sk-->[2450820.0, 2452822.0, 0.0, 4.0, 361.6287128712871]
ESTIMATE
|
|   |
|
|   |----3:EXCHANGE
|
|   |       distribution type: BROADCAST
|
|   |       cardinality: 362
|
|   |
|
|   0:OlapScanNode
|
|      table: store_returns, rollup: store_returns
|
|      preAggregation: on
|
|      Predicates: 3: sr_returned_date_sk IS NOT NULL
|
|      partitionsRatio=1/1, tabletsRatio=10/10
|
```

```
|
tabletList=10241,10243,10245,10247,10249,10251,10253,10255,10257,10259
|
|      actualRows=287514, avgRowSize=20.0
|
|      cardinality: 277502
|
|      probe runtime filters:
|
|      - filter_id = 0, probe_expr = (3: sr_returned_date_sk)
|
|      column statistics:
|
|      * sr_returned_date_sk-->[2450820.0, 2452822.0, 0.0, 4.0, 2018.0]
ESTIMATE
|
|      * sr_customer_sk-->[1.0, 100000.0, 0.034836564480338346, 4.0,
86524.0] ESTIMATE
|
|      * sr_store_sk-->[1.0, 10.0, 0.034996556689413386, 4.0, 6.0] ESTIMATE
|
|      * sr_return_amt-->[0.0, 16917.12, 0.034878301578357924, 8.0,
113968.0] ESTIMATE
|
|
|
| PLAN FRAGMENT 3(F01)
|
|
|
|   Input Partition: RANDOM
|
|   OutPut Partition: UNPARTITIONED
|
|   OutPut Exchange Id: 03
|
|
|
|   2:Project
|
|   |  output columns:
|
|   |  21 <-> [21: d_date_sk, INT, false]
|
|   |  cardinality: 362
|
```

```
|   |  column statistics:
|
|   |  * d_date_sk-->[2415022.0, 2488070.0, 0.0, 4.0, 361.6287128712871]
ESTIMATE
|
|   |
|
|   1:OlapScanNode
|
|      table: date_dim, rollup: date_dim
|
|      preAggregation: on
|
|      Predicates: [27: d_year, INT, false] = 2000
|
|      partitionsRatio=1/1, tabletsRatio=5/5
|
|      tabletList=11543,11545,11547,11549,11551
|
|      actualRows=73049, avgRowSize=8.0
|
|      cardinality: 362
|
|      column statistics:
|
|      * d_date_sk-->[2415022.0, 2488070.0, 0.0, 4.0, 361.6287128712871]
ESTIMATE
|
|      * d_year-->[2000.0, 2000.0, 0.0, 4.0, 202.0] ESTIMATE
|
+-------------------------------------------------------------------------
-------------------------------------------------------------------------
----+
114 rows in set
```

EXPLAIN ANALYZE

EXPLAIN ANALYZE выполняет указанный SQL-запрос и показывает профиль этого запроса. Подробнее о профиле запроса см. в Обзор профиля запроса.

Эта функция поддерживается начиная с версии v1.5.2.

> **ВНИМАНИЕ**

> Только пользователи с привилегией SELECT или INSERT на определённую таблицу могут выполнять эту операцию.

Синтаксис

```
EXPLAIN ANALYZE <statement>
```

Параметры

`statement`: SQL-запрос, профиль которого вы хотите просмотреть. Поддерживаются как SELECT, так и INSERT INTO.

Примечания по использованию

Обратите внимание, что при анализе профилей команды INSERT INTO данные фактически не будут вставлены. По умолчанию транзакция отменяется, что гарантирует отсутствие непреднамеренных изменений данных в процессе анализа профиля.

Примеры

Пример 1: Симуляция и анализ запроса SELECT. Результаты запроса отбрасываются.

```
MySQL root@172.26.81.178:tpch> explain analyze select count(*) from lineitem;
+-----------------------
```

Пример 2: Симуляция и анализ запроса INSERT INTO. Транзакция загрузки будет отменена.

```
MySQL root@172.26.81.178:tpch> select count(*) from lineitem;
+----------+
| count(*) |
+----------+
| 6001215  |
+----------+
1 row in set
Time: 0.018s
MySQL root@172.26.81.178:tpch> explain analyze insert into lineitem select * from lineitem;
+-------------------
```

SHOW PROFILELIST

SHOW PROFILELIST выводит список записей профилей запросов, кэшированных в вашем cluster Selena. Подробнее о профиле запроса см. в Обзор профиля запроса.

Эта функция поддерживается начиная с версии v1.5.2.

Для выполнения этой операции не требуются привилегии.

Синтаксис

```
SHOW PROFILELIST [LIMIT n]
```

Параметры

`LIMIT n`: выводит n последних записей.

Возвращаемые данные

Возвращаемое значение	Описание
QueryId	ID запроса.
StartTime	Время начала запроса.
Time	Задержка выполнения запроса.
State	Статус запроса, включая: `Error`: запрос завершился с ошибкой. `Finished`: запрос завершён. `Running`: запрос выполняется.
Statement	SQL-запрос.

Примеры

Пример 1: Показать пять последних записей профилей запросов.

```
SHOW PROFILELIST LIMIT 5;
```

Связанные SQL-команды

- ANALYZE PROFILE
- EXPLAIN ANALYZE

3.3 Loading and unloading

ALTER LOAD

ALTER LOAD изменяет приоритет задания Broker Load, находящегося в состоянии **QUEUEING** или **LOADING**. Этот оператор поддерживается начиная с версии 2.5.

> **ПРИМЕЧАНИЕ**
>
> Изменение приоритета задания Broker Load, находящегося в состоянии **LOADING**, не влияет на выполнение задания.

Синтаксис

```
ALTER LOAD FOR <label_name>
PROPERTIES
(
    'priority'='{LOWEST | LOW | NORMAL | HIGH | HIGHEST}'
)
```

Параметры

Параметр	Обязательный	Описание
label_name	Да	Метка задания загрузки. Формат: `[<database_name>.]<label_name>`. См. BROKER LOAD.
priority	Да	Новый приоритет, который вы хотите указать для задания загрузки. Допустимые значения: `LOWEST`, `LOW`, `NORMAL`, `HIGH` и `HIGHEST`. См. BROKER LOAD.

Примеры

Предположим, у вас есть задание Broker Load с меткой `test_db.label1`, и задание находится в состоянии **QUEUEING**. Если вы хотите запустить задание как можно скорее, вы можете выполнить следующую команду, чтобы изменить приоритет задания на `HIGHEST`:

```
ALTER LOAD FOR test_db.label1
PROPERTIES
(
    'priority'='HIGHEST'
);
```

BROKER LOAD

Selena предоставляет метод загрузки данных на основе MySQL - Broker Load. После отправки задачи загрузки Selena выполняет её асинхронно. Вы можете использовать `SELECT * FROM information_schema.loads` для запроса результата задачи. Эта функция поддерживается начиная с версии v1.5.2. Для получения дополнительной информации о фоновой информации, принципах, поддерживаемых форматах файлов данных, способах выполнения загрузки в одну и несколько таблиц, а также о том, как просматривать результаты задач, см. loading overview.

Вы можете загружать данные в таблицы Selena только как пользователь, имеющий привилегию INSERT на эти таблицы Selena. Если у вас нет привилегии INSERT, следуйте инструкциям в разделе GRANT, чтобы предоставить привилегию INSERT пользователю, которого вы используете для подключения к вашему cluster Selena. Синтаксис: `GRANT INSERT ON TABLE <table_name> IN DATABASE <database_name> TO { ROLE <role_name> | USER <user_identity>}`.

Синтаксис

```
LOAD LABEL [<database_name>.]<label_name>
(
    data_desc[, data_desc ...]
)
WITH BROKER
(
    StorageCredentialParams
)
[PROPERTIES
(
    opt_properties
)
]
```

Обратите внимание, что в Selena некоторые литералы используются в качестве зарезервированных ключевых слов SQL. Не используйте эти ключевые слова напрямую в SQL-выражениях. Если вы хотите использовать такое ключевое слово в SQL-выражении, заключите его в обратные кавычки (`). См. Keywords.

Параметры

database_name и label_name

`label_name` указывает метку задачи загрузки. Для соглашений об именовании см. System limits.

`database_name` опционально указывает имя базы данных, к которой принадлежит целевая таблица.

Каждая задача загрузки имеет метку, которая является уникальной в пределах всей базы данных. Вы можете использовать метку задачи загрузки для просмотра статуса выполнения задачи и предотвращения повторной загрузки тех же данных. Когда задача загрузки переходит в состояние **FINISHED**, её метка не может быть использована повторно. Только метка задачи загрузки, которая перешла в состояние **CANCELLED**, может быть использована повторно. В большинстве случаев метка задачи загрузки используется повторно для повтора этой задачи и загрузки тех же данных, тем самым реализуя семантику Exactly-Once.

Для соглашений об именовании меток см. System limits.

data_desc

Описание пакета данных для загрузки. Каждый дескриптор `data_desc` объявляет информацию, такую как источник данных, ETL-функции, целевая таблица Selena и целевые партиции.

Broker Load поддерживает загрузку нескольких файлов данных за один раз. В одной задаче загрузки вы можете использовать несколько дескрипторов `data_desc` для объявления нескольких файлов данных, которые вы хотите загрузить, или использовать один дескриптор `data_desc` для объявления одного пути к файлу, из которого вы хотите загрузить все файлы данных. Broker Load также может гарантировать транзакционную атомарность каждой задачи загрузки, которая выполняется для загрузки нескольких файлов данных. Атомарность означает, что загрузка нескольких файлов данных в одной

задаче загрузки должна либо успешно завершиться, либо завершиться с ошибкой. Никогда не происходит так, что загрузка некоторых файлов данных успешна, в то время как загрузка других файлов завершается с ошибкой.

`data_desc` поддерживает следующий синтаксис:

```
DATA INFILE ("<file_path>"[, "<file_path>" ...])
[NEGATIVE]
INTO TABLE <table_name>
[PARTITION (<partition1_name>[, <partition2_name> ...])]
[TEMPORARY PARTITION (<temporary_partition1_name>[,
<temporary_partition2_name> ...])]
[COLUMNS TERMINATED BY "<column_separator>"]
[ROWS TERMINATED BY "<row_separator>"]
[FORMAT AS "CSV | Parquet | ORC"]
[(format_type_options)]
[(column_list)]
[COLUMNS FROM PATH AS (<partition_field_name>[, <partition_field_name>
...])]
[SET <k1=f1(v1)>[, <k2=f2(v2)> ...]]
[WHERE predicate]
```

`data_desc` должен включать следующие параметры:

- `file_path`

  Указывает путь сохранения одного или нескольких файлов данных, которые вы хотите загрузить.

  Вы можете указать этот параметр как путь сохранения одного файла данных. Например, вы можете указать этот параметр как `"hdfs://<hdfs_host>:<hdfs_port>/user/data/tablename/20210411"` для загрузки файла данных с именем `20210411` из пути `/user/data/tablename` на HDFS-сервере.

  Вы также можете указать этот параметр как путь сохранения нескольких файлов данных, используя подстановочные символы `?`, `*`, `[]`, `{}` или `^`. См. Wildcard reference. Например, вы можете указать этот параметр как `"hdfs://<hdfs_host>:<hdfs_port>/user/data/tablename/*/*"` или `"hdfs://<hdfs_host>:<hdfs_port>/user/data/tablename/dt=202104*/*"` для загрузки файлов данных из

всех партиций или только партиций `202104` в пути `/user/data/tablename` на HDFS-сервере.

> **NOTE**
>
> Подстановочные символы также могут использоваться для указания промежуточных путей.

В приведенных выше примерах параметры `hdfs_host` и `hdfs_port` описаны следующим образом:

- `hdfs_host`: IP-адрес хоста NameNode в HDFS-кластере.
- `hdfs_host`: порт FS хоста NameNode в HDFS-кластере. По умолчанию номер порта - `9000`.

> **NOTICE**
>
> - Broker Load поддерживает доступ к AWS S3 по протоколу S3 или S3A. Поэтому при загрузке данных из AWS S3 вы можете включить `s3://` или `s3a://` в качестве префикса в URI S3, который вы передаете в качестве пути к файлу.
> - Broker Load поддерживает доступ к Google GCS только по протоколу gs. Поэтому при загрузке данных из Google GCS вы должны включить `gs://` в качестве префикса в URI GCS, который вы передаете в качестве пути к файлу.
> - При загрузке данных из Blob Storage вы должны использовать протокол wasb или wasbs для доступа к вашим данным:
> - Если ваша учетная запись хранилища разрешает доступ по HTTP, используйте протокол wasb и запишите путь к файлу как `wasb://<container_name>@<storage_account_name>.blob.core.windows.net/<path>/<file_name>/*`.
> - Если ваша учетная запись хранилища разрешает доступ по HTTPS, используйте протокол wasbs и запишите путь к файлу как `wasbs://<container_name>@<storage_account_name>.blob.core.windows.net/<path>/<file_name>/*`.
> - При загрузке данных из Data Lake Storage Gen2 вы должны использовать протокол abfs или abfss для доступа к вашим данным:

> - Если ваша учетная запись хранилища разрешает доступ по HTTP, используйте протокол abfs и запишите путь к файлу как `abfs://<container_name>@<storage_account_name>.dfs.core.windows.net/<file_name>`.
> - Если ваша учетная запись хранилища разрешает доступ по HTTPS, используйте протокол abfss и запишите путь к файлу как `abfss://<container_name>@<storage_account_name>.dfs.core.windows.net/<file_name>`.
>
> - При загрузке данных из Data Lake Storage Gen1 вы должны использовать протокол adl для доступа к вашим данным и записать путь к файлу как `adl://<data_lake_storage_gen1_name>.azuredatalakestore.net/<path>/<file_name>`.

- `INTO TABLE`

  Указывает имя целевой таблицы Selena.

`data_desc` также может опционально включать следующие параметры:

- `NEGATIVE`

  Отменяет загрузку определенного пакета данных. Для этого вам нужно загрузить тот же пакет данных с указанным ключевым словом `NEGATIVE`.

> **NOTE**
>
> Этот параметр действителен только когда таблица Selena является Aggregate-таблицей и все её столбцы значений вычисляются функцией `sum`.

- `PARTITION`

  Указывает партиции, в которые вы хотите загрузить данные. По умолчанию, если вы не указываете этот параметр, исходные данные будут загружены во все партиции таблицы Selena.

- `TEMPORARY PARTITION`

  Указывает имя временной партиции, в которую вы хотите загрузить данные. Вы можете указать несколько временных партиций, которые должны быть разделены

запятыми (,).

- `COLUMNS TERMINATED BY`

  Указывает разделитель столбцов, используемый в файле данных. По умолчанию, если вы не указываете этот параметр, он по умолчанию имеет значение `\t`, что означает табуляцию. Разделитель столбцов, который вы указываете с помощью этого параметра, должен быть таким же, как разделитель столбцов, который фактически используется в файле данных. В противном случае задача загрузки завершится неудачей из-за недостаточного качества данных, и её `State` будет `CANCELLED`.

  Задачи Broker Load отправляются по протоколу MySQL. Selena и MySQL оба экранируют символы в запросах на загрузку. Поэтому, если разделитель столбцов является невидимым символом, таким как табуляция, вы должны добавить обратную косую черту () перед разделителем столбцов. Например, вы должны ввести `\\t`, если разделитель столбцов - `\t`, и вы должны ввести `\\n`, если разделитель столбцов - `\n`. Файлы Apache Hive™ используют `\x01` в качестве разделителя столбцов, поэтому вы должны ввести `\\x01`, если файл данных из Hive.

  > **NOTE**
  >
  > - Для CSV-данных вы можете использовать строку UTF-8, такую как запятая (,), табуляция или вертикальная черта (|), длина которой не превышает 50 байт, в качестве текстового разделителя.
  > - Null-значения обозначаются с помощью `\N`. Например, файл данных состоит из трех столбцов, и запись из этого файла данных содержит данные в первом и третьем столбцах, но не содержит данных во втором столбце. В этой ситуации вам нужно использовать `\N` во втором столбце для обозначения null-значения. Это означает, что запись должна быть составлена как `a,\N,b` вместо `a,,b`. `a,,b` обозначает, что второй столбец записи содержит пустую строку.

- `ROWS TERMINATED BY`

  Указывает разделитель строк, используемый в файле данных. По умолчанию, если вы не указываете этот параметр, он по умолчанию имеет значение `\n`, что означает перевод строки. Разделитель строк, который вы указываете с помощью этого параметра, должен быть таким же, как разделитель строк, который фактически

используется в файле данных. В противном случае задача загрузки завершится неудачей из-за недостаточного качества данных, и её `State` будет `CANCELLED`. Этот параметр поддерживается начиная с версии v1.5.2.

Для заметок об использовании разделителя строк см. заметки об использовании для предыдущего параметра `COLUMNS TERMINATED BY`.

- `FORMAT AS`

  Указывает формат файла данных. Допустимые значения: `CSV`, `Parquet` и `ORC`. По умолчанию, если вы не указываете этот параметр, Selena определяет формат файла данных на основе расширения имени файла **.csv**, **.parquet** или **.orc**, указанного в параметре `file_path`.

- `format_type_options`

  Указывает параметры формата CSV, когда `FORMAT AS` установлен в `CSV`. Синтаксис:

```
(
    key = value
    key = value
    ...
)
```

> **NOTE**
>
> `format_type_options` поддерживается в версии v1.5.2 и выше.

В следующей таблице описаны параметры.

Параметр	Описание
skip_header	Указывает, пропускать ли первые строки файла данных, когда файл данных в формате CSV. Тип: INTEGER. Значение по умолчанию: `0`. В некоторых CSV-форматированных файлах данных первые строки в начале используются для определения метаданных, таких как имена столбцов и типы данных столбцов. Установив параметр `skip_header`, вы можете позволить Selena пропустить первые строки файла данных во время загрузки данных. Например, если вы установите этот параметр в `1`, Selena пропустит первую строку файла данных во время загрузки данных.

Параметр	Описание
	Первые строки в начале файла данных должны быть разделены с использованием разделителя строк, который вы указываете в выражении загрузки.

trim_space	Указывает, удалять ли пробелы перед и после разделителей столбцов из файла данных, когда файл данных в формате CSV. Тип: BOOLEAN. Значение по умолчанию: `false`.
	Для некоторых баз данных пробелы добавляются к разделителям столбцов при экспорте данных в виде CSV-форматированного файла данных. Такие пробелы называются ведущими пробелами или завершающими пробелами в зависимости от их расположения. Установив параметр `trim_space`, вы можете позволить Selena удалять такие ненужные пробелы во время загрузки данных.
	Обратите внимание, что Selena не удаляет пробелы (включая ведущие пробелы и завершающие пробелы) внутри поля, заключенного в пару символов, указанных в `enclose`. Например, следующие значения полей используют вертикальную черту (`|`) в качестве разделителя столбцов и двойные кавычки (`"`) в качестве символа `enclose`:
	`|"Love Selena"|`
	`|" Love Selena "|`
	`| "Love Selena" |`
	Если вы установите `trim_space` в `true`, Selena обработает предыдущие значения полей следующим образом:
	`|"Love Selena"|`
	`|" Love Selena "|`
	`|"Love Selena"|`

enclose	Указывает символ, который используется для обертывания значений полей в файле данных в соответствии с RFC4180, когда файл данных в формате CSV. Тип: однобайтовый символ. Значение по умолчанию: `NONE`. Наиболее распространенные символы - одинарная кавычка (`'`) и двойная кавычка (`"`). Все специальные символы (включая разделители строк и разделители столбцов), заключенные с использованием символа, указанного в `enclose`, считаются обычными символами. Selena может делать больше, чем RFC4180, так как позволяет вам указать любой однобайтовый символ в качестве символа `enclose`. Если значение поля содержит символ, указанный в `enclose`, вы можете использовать тот же символ для экранирования этого символа `enclose`. Например, вы устанавливаете `enclose` в `"`, и значение поля - `a "quoted" c`. В этом случае вы можете ввести значение поля как `"a ""quoted"" c"` в файл данных.
escape	Указывает символ, который используется для экранирования различных специальных символов, таких как разделители строк, разделители столбцов, символы экранирования и символы, указанные в `enclose`, которые затем рассматриваются Selena как обычные символы и анализируются как часть значений полей, в которых они находятся. Тип: однобайтовый символ. Значение по умолчанию: `NONE`. Наиболее распространенный символ - косая черта (`\`), которая должна быть записана как двойная косая черта (`\\`) в SQL-выражениях. **NOTE** Символ, указанный в `escape`, применяется как внутри, так и снаружи каждой пары символов,

Параметр	Описание
	указанных в `enclose`. Два примера следующие: • Когда вы устанавливаете `enclose` в `"` и `escape` в `\`, Selena анализирует `"say \"Hello world\""` в `say "Hello world"`. • Предположим, что разделитель столбцов - запятая (`,`). Когда вы устанавливаете `escape` в `\`, Selena анализирует `a, b\, c` в два отдельных значения полей: `a` и `b, c`.

- `column_list`

  Указывает сопоставление столбцов между файлом данных и таблицей Selena. Синтаксис: `(<column_name>[, <column_name> ...])`. Столбцы, объявленные в `column_list`, сопоставляются по имени со столбцами таблицы Selena.

  > **NOTE**
  >
  > Если столбцы файла данных сопоставляются последовательно со столбцами таблицы Selena, вам не нужно указывать `column_list`.

  Если вы хотите пропустить определенный столбец файла данных, вам нужно только временно назвать этот столбец иначе, чем любой из столбцов таблицы Selena. Для получения дополнительной информации см. loading overview.

- `COLUMNS FROM PATH AS`

  Извлекает информацию об одном или нескольких полях партиции из пути к файлу, который вы указываете. Этот параметр действителен только когда путь к файлу содержит поля партиции.

  Например, если файл данных хранится в пути `/path/col_name=col_value/file1`, где `col_name` - поле партиции и может быть сопоставлено со столбцом таблицы Selena, вы можете указать этот параметр как `col_name`. Таким образом, Selena извлекает значения `col_value` из пути и загружает их в столбец таблицы Selena, на который сопоставлен `col_name`.

  > **NOTE**
  >
  > Этот параметр доступен только при загрузке данных из HDFS.

- `SET`

  Указывает одну или несколько функций, которые вы хотите использовать для преобразования столбца файла данных. Примеры:

  - Таблица Selena состоит из трех столбцов, которые представляют собой `col1`, `col2` и `col3` последовательно. Файл данных состоит из четырех столбцов, среди которых первые два столбца сопоставляются последовательно с `col1` и `col2` таблицы Selena, а сумма последних двух столбцов сопоставляется с `col3` таблицы Selena. В этом случае вам нужно указать `column_list` как `(col1,col2,tmp_col3,tmp_col4)` и указать `(col3=tmp_col3+tmp_col4)` в предложении SET для реализации преобразования данных.
  - Таблица Selena состоит из трех столбцов, которые представляют собой `year`, `month` и `day` последовательно. Файл данных состоит только из одного столбца, который содержит значения даты и времени в формате `yyyy-mm-dd hh:mm:ss`. В этом случае вам нужно указать `column_list` как `(tmp_time)` и указать `(year = year(tmp_time), month=month(tmp_time), day=day(tmp_time))` в предложении SET для реализации преобразования данных.
- `WHERE`

  Указывает условия, на основе которых вы хотите фильтровать исходные данные. Selena загружает только исходные данные, которые соответствуют условиям фильтрации, указанным в предложении WHERE.

WITH BROKER

В версии v1.5.2 и более ранних введите `WITH BROKER "<broker_name>"`, чтобы указать broker, который вы хотите использовать. Начиная с версии v1.5.2, вам больше не нужно указывать broker, но вам все равно нужно сохранить ключевое слово `WITH BROKER`.

StorageCredentialParams

Информация аутентификации, используемая Selena для доступа к вашей системе хранения.

HDFS

HDFS с открытым исходным кодом поддерживает два метода аутентификации: простая аутентификация и аутентификация Kerberos. Broker Load использует простую аутентификацию по умолчанию. HDFS с открытым исходным кодом также поддерживает

настройку механизма HA для NameNode. Если вы выбираете HDFS с открытым исходным кодом в качестве вашей системы хранения, вы можете указать конфигурацию аутентификации и конфигурацию HA следующим образом:

- Конфигурация аутентификации
  - Если вы используете простую аутентификацию, настройте `StorageCredentialParams` следующим образом:

```
"hadoop.security.authentication" = "simple",
"username" = "<hdfs_username>",
"password" = "<hdfs_password>"
```

В следующей таблице описаны параметры в `StorageCredentialParams`.

Параметр	Описание
hadoop.security.authentication	Метод аутентификации. Допустимые значения: `simple` и `kerberos`. Значение по умолчанию: `simple`. `simple` представляет простую аутентификацию, то есть без аутентификации, а `kerberos` представляет аутентификацию Kerberos.
username	Имя пользователя учетной записи, которую вы хотите использовать для доступа к NameNode HDFS-кластера.
password	Пароль учетной записи, которую вы хотите использовать для доступа к NameNode HDFS-кластера.

  - Если вы используете аутентификацию Kerberos, настройте `StorageCredentialParams` следующим образом:

```
"hadoop.security.authentication" = "kerberos",
"kerberos_principal" = "nn/zelda1@ZELDA.COM",
"kerberos_keytab" = "/keytab/hive.keytab",
"kerberos_keytab_content" = "YWFhYWFh"
```

В следующей таблице описаны параметры в `StorageCredentialParams`.

Параметр	Описание
hadoop.security.aut	Метод аутентификации. Допустимые значения: `simple` и `kerberos`. Значение по умолчанию: `simple`. `simple` представляет простую аутентификацию, то есть без аутентификации, а `kerberos` представляет аутентификацию Kerberos.

Параметр	Описание
hentication	
kerberos_principal	Принципал Kerberos для аутентификации. Каждый принципал состоит из следующих трех частей, чтобы гарантировать, что он уникален в HDFS-кластере: ▪ `username` или `servicename`: имя принципала. ▪ `instance`: имя сервера, на котором размещен узел для аутентификации в HDFS-кластере. Имя сервера помогает гарантировать уникальность принципала, например, когда HDFS-кластер состоит из нескольких DataNode, которые каждый независимо аутентифицируются. ▪ `realm`: имя области. Имя области должно быть в верхнем регистре. Пример: `nn/zelda1@ZELDA.COM`.
kerberos_keytab	Путь сохранения файла keytab Kerberos.
kerberos_keytab_content	Содержимое файла keytab Kerberos в кодировке Base64. Вы можете выбрать указание либо `kerberos_keytab`, либо `kerberos_keytab_content`.

- Конфигурация HA

  Вы можете настроить механизм HA для NameNode HDFS-кластера. Таким образом, если NameNode переключается на другой узел, Selena может автоматически идентифицировать новый узел, который служит NameNode. Это включает следующие сценарии:

  - Если вы загружаете данные из одного HDFS-кластера с одним настроенным пользователем Kerberos, поддерживается как загрузка на основе broker, так и загрузка без broker.

    - Для выполнения загрузки на основе broker убедитесь, что развернута хотя бы одна независимая группа broker, и поместите файл `hdfs-site.xml` в путь `{deploy}/conf` на узле broker, обслуживающем HDFS-кластер. Selena добавит путь `{deploy}/conf` в переменную окружения `CLASSPATH` при запуске broker, позволяя broker читать информацию об узлах HDFS-кластера.

    - Для выполнения загрузки без broker вам нужно только установить `hadoop.security.authentication = kerberos` в `conf/core-site.xml` в каталогах развертывания всех узлов FE, BE и CN в вашем кластере и использовать команду `kinit` для настройки учетной записи Kerberos.

- Если вы загружаете данные из одного HDFS-кластера с несколькими настроенными пользователями Kerberos, поддерживается только загрузка на основе broker. Убедитесь, что развернута хотя бы одна независимая группа broker, и поместите файл `hdfs-site.xml` в путь `{deploy}/conf` на узле broker, обслуживающем HDFS-кластер. Selena добавит путь `{deploy}/conf` в переменную окружения `CLASSPATH` при запуске broker, позволяя broker читать информацию об узлах HDFS-кластера.

- Если вы загружаете данные из нескольких HDFS-кластеров (независимо от того, настроен один или несколько пользователей Kerberos), поддерживается только загрузка на основе broker. Убедитесь, что развернута хотя бы одна независимая группа broker для каждого из этих HDFS-кластеров, и выполните одно из следующих действий, чтобы позволить broker читать информацию об узлах HDFS-кластера:

  - Поместите файл `hdfs-site.xml` в путь `{deploy}/conf` на узле broker, обслуживающем каждый HDFS-кластер. Selena добавит путь `{deploy}/conf` в переменную окружения `CLASSPATH` при запуске broker, позволяя broker читать информацию об узлах этого HDFS-кластера.

  - Добавьте следующую конфигурацию HA при создании задачи:

```
"dfs.nameservices" = "ha_cluster",
"dfs.ha.namenodes.ha_cluster" = "ha_n1,ha_n2",
"dfs.namenode.rpc-address.ha_cluster.ha_n1" = "<hdfs_host>:<hdfs_p
"dfs.namenode.rpc-address.ha_cluster.ha_n2" = "<hdfs_host>:<hdfs_p
"dfs.client.failover.proxy.provider" =
"org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxy
```

    В следующей таблице описаны параметры в конфигурации HA.

Параметр	Описание
dfs.nameservices	Имя HDFS-кластера.
dfs.ha.namenodes.XXX	Имя NameNode в HDFS-кластере. Если вы указываете несколько имен NameNode, разделите их запятыми (`,`). `xxx` - это имя HDFS-кластера, которое вы указали в `dfs.nameservices`.

Параметр	Описание
dfs.namenode.rpc-address.XXX.NN	RPC-адрес NameNode в HDFS-кластере. `NN` - это имя NameNode, которое вы указали в `dfs.ha.namenodes.XXX`.
dfs.client.failover.proxy.provider	Провайдер NameNode, к которому будет подключаться клиент. Значение по умолчанию: `org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider`.

> **NOTE**
>
> Вы можете использовать выражение SHOW BROKER для проверки broker, развернутых в вашем кластере Selena.

AWS S3

Если вы выбираете AWS S3 в качестве вашей системы хранения, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации на основе instance profile, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.region" = "<aws_s3_region>"
```

- Чтобы выбрать метод аутентификации на основе assumed role, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.iam_role_arn" = "<iam_role_arn>",
"aws.s3.region" = "<aws_s3_region>"
```

- Чтобы выбрать метод аутентификации на основе IAM user, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "false",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>",
"aws.s3.region" = "<aws_s3_region>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательно	Описание
aws.s3.use_instance_profile	Да	Указывает, включать ли методы учетных данных instance profile и assumed role. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`.
aws.s3.iam_role_arn	Нет	ARN IAM-роли, которая имеет привилегии на вашем AWS S3 bucket. Если вы выбираете assumed role в качестве метода учетных данных для доступа к AWS S3, вы должны указать этот параметр.
aws.s3.region	Да	Регион, в котором находится ваш AWS S3 bucket. Пример: `us-west-1`.
aws.s3.access_key	Нет	Access key вашего IAM-пользователя. Если вы выбираете IAM user в качестве метода учетных данных для доступа к AWS S3, вы должны указать этот параметр.
aws.s3.secret_key	Нет	Secret key вашего IAM-пользователя. Если вы выбираете IAM user в качестве метода учетных данных для доступа к AWS S3, вы должны указать этот параметр.

Для получения информации о том, как выбрать метод аутентификации для доступа к AWS S3 и как настроить политику управления доступом в AWS IAM Console, см. Authentication parameters for accessing AWS S3.

Google GCS

Если вы выбираете Google GCS в качестве вашей системы хранения, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации на основе VM, настройте `StorageCredentialParams` следующим образом:

  ```
  "gcp.gcs.use_compute_engine_service_account" = "true"
  ```

  В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.use_compute_engine_service_account	false	true	Указывает, использовать ли непосредственно служебную учетную запись, привязанную к вашему Compute Engine.

- Чтобы выбрать метод аутентификации на основе service account, настройте `StorageCredentialParams` следующим образом:

```
"gcp.gcs.service_account_email" = "<google_service_account_email>",
"gcp.gcs.service_account_private_key_id" = "
<google_service_private_key_id>",
"gcp.gcs.service_account_private_key" = "<google_service_private_key>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.service_account_email	""	`"user@hello.iam.gserviceaccount.com"`	Адрес электронной почты в JSON-файле, созданном при создании служебной учетной записи.
gcp.gcs.service_account_private_key_id	""	"61d257bd8479547cb3e04f0b9b6b9ca07af3b7ea"	ID приватного ключа в JSON-файле, созданном при создании служебной учетной записи.
gcp.gcs.service_account_private_key	""	"-----BEGIN PRIVATE KEY-----xxxx-----END PRIVATE KEY-----\n"	Приватный ключ в JSON-файле, созданном при создании служебной учетной записи.

- Чтобы выбрать метод аутентификации на основе impersonation, настройте `StorageCredentialParams` следующим образом:
  - Заставить VM instance имитировать служебную учетную запись:

```
"gcp.gcs.use_compute_engine_service_account" = "true",
"gcp.gcs.impersonation_service_account" = "
<assumed_google_service_account_email>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.use_compute_engine_service_account	false	true	Указывает, использовать ли непосредственно служебную учетную запись, привязанную к вашему Compute Engine.
gcp.gcs.impersonation_service_account	""	"hello"	Служебная учетная запись, которую вы хотите имитировать.

- Заставить служебную учетную запись (названную мета-служебной учетной записью) имитировать другую служебную учетную запись (названную данными служебной учетной записью):

```
"gcp.gcs.service_account_email" = "<google_service_account_email>",
"gcp.gcs.service_account_private_key_id" = "
<meta_google_service_account_email>",
"gcp.gcs.service_account_private_key" = "
<meta_google_service_account_email>",
"gcp.gcs.impersonation_service_account" = "
<data_google_service_account_email>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.service_account_email	""	`"user@hello.iam.gserviceaccount.com"`	Адрес электронной почты в JSON-файле, созданном при создании мета-служебной учетной записи.
gcp.gcs.service_account_private_key_id	""	"61d257bd8479547cb3e04f0b9b6b9ca07af3b7ea"	ID приватного ключа в JSON-файле, созданном при создании мета-служебной учетной записи.
gcp.gcs.service_account_private_key	""	"-----BEGIN PRIVATE KEY----xxxx-----END PRIVATE KEY-----\n"	Приватный ключ в JSON-файле, созданном при создании мета-служебной учетной записи.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.impersonation_service_account	""	"hello"	Данные служебной учетной записи, которую вы хотите имитировать.

Другая S3-совместимая система хранения

Если вы выбираете другую S3-совместимую систему хранения, такую как MinIO, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.enable_ssl" = "false",
"aws.s3.enable_path_style_access" = "true",
"aws.s3.endpoint" = "<s3_endpoint>",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательно	Описание
aws.s3.enable_ssl	Да	Указывает, включать ли SSL-соединение. Допустимые значения: `true` и `false`. Значение по умолчанию: `true`.
aws.s3.enable_path_style_access	Да	Указывает, включать ли доступ к URL в стиле пути. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`. Для MinIO вы должны установить значение `true`.
aws.s3.endpoint	Да	Конечная точка, которая используется для подключения к вашей S3-совместимой системе хранения вместо AWS S3.
aws.s3.access_key	Да	Access key вашего IAM-пользователя.
aws.s3.secret_key	Да	Secret key вашего IAM-пользователя.

Microsoft Azure Storage

Azure Blob Storage

Если вы выбираете Blob Storage в качестве вашей системы хранения, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации Shared Key, настройте `StorageCredentialParams` следующим образом:

```
"azure.blob.storage_account" = "<storage_account_name>",
"azure.blob.shared_key" = "<storage_account_shared_key>"
```

  В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательно	Описание
azure.blob.storage_account	Да	Имя пользователя вашей учетной записи Blob Storage.
azure.blob.shared_key	Да	Shared key вашей учетной записи Blob Storage.

- Чтобы выбрать метод аутентификации SAS Token, настройте `StorageCredentialParams` следующим образом:

```
"azure.blob.storage_account" = "<storage_account_name>",
"azure.blob.container" = "<container_name>",
"azure.blob.sas_token" = "<storage_account_SAS_token>"
```

  В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательно	Описание
azure.blob.storage_account	Да	Имя пользователя вашей учетной записи Blob Storage.
azure.blob.container	Да	Имя blob container, который хранит ваши данные.
azure.blob.sas_token	Да	SAS-токен, который используется для доступа к вашей учетной записи Blob Storage.

Azure Data Lake Storage Gen2

Если вы выбираете Data Lake Storage Gen2 в качестве вашей системы хранения, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации Managed Identity, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls2.oauth2_use_managed_identity" = "true",
"azure.adls2.oauth2_tenant_id" = "<service_principal_tenant_id>",
"azure.adls2.oauth2_client_id" = "<service_client_id>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательно	Описание
azure.adls2.oauth2_use_managed_identity	Да	Указывает, включать ли метод аутентификации Managed Identity. Установите значение `true`.
azure.adls2.oauth2_tenant_id	Да	ID tenant, данные которого вы хотите получить.
azure.adls2.oauth2_client_id	Да	Client (application) ID управляемой идентичности.

- Чтобы выбрать метод аутентификации Shared Key, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls2.storage_account" = "<storage_account_name>",
"azure.adls2.shared_key" = "<storage_account_shared_key>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательно	Описание
azure.adls2.storage_account	Да	Имя пользователя вашей учетной записи хранения Data Lake Storage Gen2.
azure.adls2.shared_key	Да	Shared key вашей учетной записи хранения Data Lake Storage Gen2.

- Чтобы выбрать метод аутентификации Service Principal, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls2.oauth2_client_id" = "<service_client_id>",
"azure.adls2.oauth2_client_secret" = "
<service_principal_client_secret>",
"azure.adls2.oauth2_client_endpoint" = "
<service_principal_client_endpoint>"
```

В следующей таблице описаны параметры, которые вам нужно настроить `in StorageCredentialParams`.

Параметр	Обязательно	Описание
azure.adls2.oauth2_client_id	Да	Client (application) ID service principal.
azure.adls2.oauth2_client_secret	Да	Значение нового client (application) secret, созданного.
azure.adls2.oauth2_client_endpoint	Да	Конечная точка токена OAuth 2.0 (v1) service principal или application.

Azure Data Lake Storage Gen1

Если вы выбираете Data Lake Storage Gen1 в качестве вашей системы хранения, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации Managed Service Identity, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls1.use_managed_service_identity" = "true"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательно	Описание
azure.adls1.use_managed_service_identity	Да	Указывает, включать ли метод аутентификации Managed Service Identity. Установите значение `true`.

- Чтобы выбрать метод аутентификации Service Principal, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls1.oauth2_client_id" = "<application_client_id>",
"azure.adls1.oauth2_credential" = "<application_client_credential>",
"azure.adls1.oauth2_endpoint" = "<OAuth_2.0_authorization_endpoint_v2>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательно	Описание
azure.adls1.oauth2_client_id	Да	Client (application) ID .
azure.adls1.oauth2_credential	Да	Значение нового client (application) secret, созданного.
azure.adls1.oauth2_endpoint	Да	Конечная точка токена OAuth 2.0 (v1) service principal или application.

opt_properties

Указывает некоторые дополнительные параметры, настройки которых применяются ко всей задаче загрузки. Синтаксис:

```
PROPERTIES ("<key1>" = "<value1>"[, "<key2>" = "<value2>" ...])
```

Поддерживаются следующие параметры:

- `timeout`

  Указывает период тайм-аута задачи загрузки. Единица измерения: секунда. Период тайм-аута по умолчанию составляет 4 часа. Мы рекомендуем указывать период тайм-аута короче 6 часов. Если задача загрузки не завершается в течение периода тайм-аута, Selena отменяет задачу загрузки, и статус задачи загрузки становится **CANCELLED**.

  > **NOTE**
  >
  > В большинстве случаев вам не нужно устанавливать период тайм-аута. Мы рекомендуем устанавливать период тайм-аута только когда задача загрузки не может завершиться в течение периода тайм-аута по умолчанию.

  Используйте следующую формулу для определения периода тайм-аута:

  Период тайм-аута > (Общий размер файлов данных для загрузки x Общее количество файлов данных для загрузки и материализованных представлений, созданных на файлах данных)/Средняя скорость загрузки

  > **NOTE**

> "Средняя скорость загрузки" - это средняя скорость загрузки для всего вашего кластера Selena. Средняя скорость загрузки варьируется для каждого кластера в зависимости от конфигурации сервера и максимального количества разрешенных параллельных задач запроса для кластера. Вы можете определить среднюю скорость загрузки на основе скоростей загрузки исторических задач загрузки.

Предположим, что вы хотите загрузить файл данных размером 1 ГБ, на котором созданы два материализованных представления, в кластер Selena, средняя скорость загрузки которого составляет 10 МБ/с. Время, необходимое для загрузки данных, составляет примерно 102 секунды.

(1 x 1024 x 3)/10 = 307.2 (секунда)

Для этого примера мы рекомендуем установить период тайм-аута в значение больше 308 секунд.

- `max_filter_ratio`

  Указывает максимальную допустимую ошибку задачи загрузки. Максимальная допустимая ошибка - это максимальный процент строк, которые могут быть отфильтрованы в результате недостаточного качества данных. Допустимые значения: `0`~`1`. Значение по умолчанию: `0`.

  - Если вы установите этот параметр в `0`, Selena не игнорирует неквалифицированные строки во время загрузки. Таким образом, если исходные данные содержат неквалифицированные строки, задача загрузки завершается с ошибкой. Это помогает гарантировать правильность данных, загруженных в Selena.

  - Если вы установите этот параметр в значение больше `0`, Selena может игнорировать неквалифицированные строки во время загрузки. Таким образом, задача загрузки может завершиться успешно, даже если исходные данные содержат неквалифицированные строки.

    > **NOTE**
    >
    > Строки, которые отфильтровываются из-за недостаточного качества данных, не включают строки, которые отфильтровываются предложением WHERE.

Если задача загрузки завершается с ошибкой, потому что максимальная допустимая ошибка установлена в `0`, вы можете использовать SHOW LOAD для просмотра результата задачи. Затем определите, могут ли неквалифицированные строки быть отфильтрованы. Если неквалифицированные строки могут быть отфильтрованы, вычислите максимальную допустимую ошибку на основе значений, возвращенных для `dpp.abnorm.ALL` и `dpp.norm.ALL` в результате задачи, настройте максимальную допустимую ошибку и снова отправьте задачу загрузки. Формула для вычисления максимальной допустимой ошибки следующая:

`max_filter_ratio` = [`dpp.abnorm.ALL`/(`dpp.abnorm.ALL` + `dpp.norm.ALL`)]

Сумма значений, возвращенных для `dpp.abnorm.ALL` и `dpp.norm.ALL`, - это общее количество строк для загрузки.

- `log_rejected_record_num`

  Указывает максимальное количество неквалифицированных строк данных, которые могут быть зарегистрированы. Этот параметр поддерживается начиная с версии v1.5.2. Допустимые значения: `0`, `-1` и любое ненулевое положительное целое число. Значение по умолчанию: `0`.

  - Значение `0` указывает, что строки данных, которые отфильтровываются, не будут зарегистрированы.
  - Значение `-1` указывает, что все строки данных, которые отфильтровываются, будут зарегистрированы.
  - Ненулевое положительное целое число, такое как `n`, указывает, что до `n` строк данных, которые отфильтровываются, могут быть зарегистрированы на каждом BE.
- `load_mem_limit`

  Указывает максимальный объем памяти, который может быть предоставлен задаче загрузки. Значение этого параметра не может превышать верхний предел памяти, поддерживаемый каждым узлом BE или CN. Единица измерения: байты: лимит памяти по умолчанию составляет 2 ГБ.

- `strict_mode`

Указывает, включать ли строгий режим. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`. `true` указывает включить строгий режим, а `false` указывает отключить строгий режим.

- `timezone`

  Указывает часовой пояс задачи загрузки. Значение по умолчанию: `Asia/Shanghai`. Настройка часового пояса влияет на результаты, возвращаемые такими функциями, как strftime, alignment_timestamp и from_unixtime. Для получения дополнительной информации см. Configure a time zone. Часовой пояс, указанный в параметре `timezone`, является часовым поясом на уровне сессии.

- `priority`

  Указывает приоритет задачи загрузки. Допустимые значения: `LOWEST`, `LOW`, `NORMAL`, `HIGH` и `HIGHEST`. Значение по умолчанию: `NORMAL`. Broker Load предоставляет параметр FE `max_broker_load_job_concurrency`, который определяет максимальное количество задач Broker Load, которые могут выполняться одновременно в вашем кластере Selena. Если количество задач Broker Load, которые отправляются в указанный период времени, превышает максимальное количество, избыточные задачи будут ожидать планирования на основе их приоритетов.

  Вы можете использовать выражение ALTER LOAD для изменения приоритета существующей задачи загрузки, которая находится в состоянии `QUEUEING` или `LOADING`.

  Selena позволяет устанавливать параметр `priority` для задачи Broker Load начиная с версии v1.5.2.

- `partial_update`

  Использовать ли частичные обновления. Допустимые значения: `TRUE` и `FALSE`. Значение по умолчанию: `FALSE`, что указывает на отключение этой функции.

- `partial_update_mode`

  Указывает режим для частичных обновлений. Допустимые значения: `row` и `column`.

  - Значение `row` (по умолчанию) означает частичные обновления в режиме строк, который более подходит для обновлений в реальном времени с большим количеством столбцов и небольшими пакетами.
  - Значение `column` означает частичные обновления в режиме столбцов, который более подходит для пакетных обновлений с небольшим количеством столбцов и большим количеством строк. В таких сценариях включение режима столбцов обеспечивает более высокую скорость обновления. Например, в таблице со 100 столбцами, если обновляются только 10 столбцов (10% от общего количества) для всех строк, скорость обновления в режиме столбцов в 10 раз быстрее.

- `merge_condition`

  Указывает имя столбца, который вы хотите использовать в качестве условия для определения того, могут ли обновления вступить в силу. Обновление из исходной записи в целевую запись вступает в силу только когда исходная запись данных имеет большее или равное значение, чем целевая запись данных в указанном столбце.

  > **NOTE**
  >
  > Столбец, который вы указываете, не может быть столбцом первичного ключа. Кроме того, только таблицы, использующие Primary Key table, поддерживают условные обновления.

Selena поддерживает загрузку JSON-данных начиная с версии v1.5.2. Параметры следующие:

- jsonpaths

  Имена ключей, которые вы хотите загрузить из JSON-файла данных. Вам нужно указать этот параметр только когда вы загружаете JSON-данные, используя режим соответствия. Значение этого параметра в формате JSON. См. Configure column mapping for JSON data loading.

- strip_outer_array

  Указывает, удалять ли самую внешнюю структуру массива. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`.

В реальных бизнес-сценариях JSON-данные могут иметь самую внешнюю структуру массива, обозначенную парой квадратных скобок `[]`. В этой ситуации мы рекомендуем установить этот параметр в `true`, чтобы Selena удалила самые внешние квадратные скобки `[]` и загрузила каждый внутренний массив как отдельную запись данных. Если вы установите этот параметр в `false`, Selena анализирует весь JSON-файл данных в один массив и загружает массив как одну запись данных. Например, JSON-данные - `[ {"category" : 1, "author" : 2}, {"category" : 3, "author" : 4} ]`. Если вы установите этот параметр в `true`, `{"category" : 1, "author" : 2}` и `{"category" : 3, "author" : 4}` анализируются в отдельные записи данных, которые загружаются в отдельные строки таблицы Selena.

- json_root

  Корневой элемент JSON-данных, которые вы хотите загрузить из JSON-файла данных. Вам нужно указать этот параметр только когда вы загружаете JSON-данные, используя режим соответствия. Значение этого параметра - допустимая строка JsonPath. По умолчанию значение этого параметра пустое, что указывает, что все данные JSON-файла данных будут загружены. Для получения дополнительной информации см. раздел "Load JSON data using matched mode with root element specified" этой темы.

При загрузке JSON-данных также обратите внимание, что размер каждого JSON-объекта не может превышать 4 ГБ. Если отдельный JSON-объект в JSON-файле данных превышает 4 ГБ, выдается ошибка "This parser can't support a document that big.".

Сопоставление столбцов

Настройка сопоставления столбцов для загрузки CSV-данных

Если столбцы файла данных могут быть сопоставлены один к одному последовательно со столбцами таблицы Selena, вам не нужно настраивать сопоставление столбцов между файлом данных и таблицей Selena.

Если столбцы файла данных не могут быть сопоставлены один к одному последовательно со столбцами таблицы Selena, вам нужно использовать параметр `columns` для настройки сопоставления столбцов между файлом данных и таблицей Selena. Это включает следующие два случая использования:

- **Одинаковое количество столбцов, но разная последовательность столбцов. Кроме того, данные из файла данных не нужно вычислять функциями перед загрузкой в соответствующие столбцы таблицы Selena.**

  В параметре `columns` вам нужно указать имена столбцов таблицы Selena в той же последовательности, в которой расположены столбцы файла данных.

  Например, таблица Selena состоит из трех столбцов, которые представляют собой `col1`, `col2` и `col3` последовательно, а файл данных также состоит из трех столбцов, которые могут быть сопоставлены со столбцами таблицы Selena `col3`, `col2` и `col1` последовательно. В этом случае вам нужно указать `"columns: col3, col2, col1"`.

- **Разное количество столбцов и разная последовательность столбцов. Кроме того, данные из файла данных нужно вычислить функциями перед загрузкой в соответствующие столбцы таблицы Selena.**

  В параметре `columns` вам нужно указать имена столбцов таблицы Selena в той же последовательности, в которой расположены столбцы файла данных, и указать функции, которые вы хотите использовать для вычисления данных. Два примера следующие:

  - Таблица Selena состоит из трех столбцов, которые представляют собой `col1`, `col2` и `col3` последовательно. Файл данных состоит из четырех столбцов, среди которых первые три столбца могут быть сопоставлены последовательно со столбцами таблицы Selena `col1`, `col2` и `col3`, а четвертый столбец не может быть сопоставлен ни с одним из столбцов таблицы Selena. В этом случае вам нужно временно указать имя для четвертого столбца файла данных, и временное имя должно отличаться от любого из имен столбцов таблицы Selena. Например, вы можете указать `"columns: col1, col2, col3, temp"`, где четвертый столбец файла данных временно назван `temp`.
  - Таблица Selena состоит из трех столбцов, которые представляют собой `year`, `month` и `day` последовательно. Файл данных состоит только из одного столбца, который содержит значения даты и времени в формате `yyyy-mm-dd hh:mm:ss`. В этом случае вы можете указать `"columns: col, year = year(col), month=month(col), day=day(col)"`, где `col` - временное имя столбца файла данных, а функции `year = year(col)`, `month=month(col)` и `day=day(col)` используются для извлечения данных из столбца файла данных `col` и загрузки данных в соответствующие столбцы таблицы Selena. Например, `year =`

`year(col)` используется для извлечения данных `yyyy` из столбца файла данных `col` и загрузки данных в столбец таблицы Selena `year`.

Для подробных примеров см. Configure column mapping.

Настройка сопоставления столбцов для загрузки JSON-данных

Если ключи JSON-документа имеют те же имена, что и столбцы таблицы Selena, вы можете загрузить JSON-форматированные данные, используя простой режим. В простом режиме вам не нужно указывать параметр `jsonpaths`. Этот режим требует, чтобы JSON-форматированные данные были объектом, обозначенным фигурными скобками `{}`, например `{"category": 1, "author": 2, "price": "3"}`. В этом примере `category`, `author` и `price` - имена ключей, и эти ключи могут быть сопоставлены один к одному по имени со столбцами `category`, `author` и `price` таблицы Selena.

Если ключи JSON-документа имеют разные имена, чем столбцы таблицы Selena, вы можете загрузить JSON-форматированные данные, используя режим соответствия. В режиме соответствия вам нужно использовать параметры `jsonpaths` и `COLUMNS` для указания сопоставления столбцов между JSON-документом и таблицей Selena:

- В параметре `jsonpaths` укажите ключи JSON в последовательности, в которой они расположены в JSON-документе.
- В параметре `COLUMNS` укажите сопоставление между ключами JSON и столбцами таблицы Selena:
  - Имена столбцов, указанные в параметре `COLUMNS`, сопоставляются один к одному последовательно с ключами JSON.
  - Имена столбцов, указанные в параметре `COLUMNS`, сопоставляются один к одному по имени со столбцами таблицы Selena.

Для примеров загрузки JSON-форматированных данных с использованием режима соответствия см. Load JSON data using matched mode.

Связанные параметры конфигурации

Параметр конфигурации FE `max_broker_load_job_concurrency` указывает максимальное количество задач Broker Load, которые могут выполняться одновременно в вашем кластере Selena.

В Selena v1.5.2 и более ранних, если общее количество задач Broker Load, которые отправляются в течение определенного периода времени, превышает максимальное количество, избыточные задачи будут поставлены в очередь и запланированы на основе времени их отправки.

Начиная с Selena v1.5.2, если общее количество задач Broker Load, которые отправляются в течение определенного периода времени, превышает максимальное количество, избыточные задачи ставятся в очередь и планируются на основе их приоритетов. Вы можете указать приоритет для задачи, используя параметр `priority`, описанный выше. Вы можете использовать ALTER LOAD для изменения приоритета существующей задачи, которая находится в состоянии **QUEUEING** или **LOADING**.

Разделение и параллельное выполнение задач

Задача Broker Load может быть разделена на одну или несколько подзадач, которые выполняются параллельно. Подзадачи в рамках задачи загрузки выполняются в рамках одной транзакции. Они все должны либо успешно завершиться, либо завершиться с ошибкой. Selena разделяет каждую задачу загрузки на основе того, как вы объявляете `data_desc` в выражении `LOAD`:

- Если вы объявляете несколько параметров `data_desc`, каждый из которых указывает отдельную таблицу, создается подзадача для загрузки данных каждой таблицы.
- Если вы объявляете несколько параметров `data_desc`, каждый из которых указывает отдельную партицию для одной и той же таблицы, создается подзадача для загрузки данных каждой партиции.

Кроме того, каждая подзадача может быть дополнительно разделена на один или несколько экземпляров, которые равномерно распределяются и параллельно выполняются на узлах BE или CN вашего кластера Selena. Selena разделяет каждую подзадачу на основе параметра FE `min_bytes_per_broker_scanner` и количества узлов BE или CN. Вы можете использовать следующую формулу для расчета количества экземпляров в отдельной подзадаче:

Количество экземпляров в отдельной подзадаче = min(Объем данных для загрузки отдельной подзадачей/`min_bytes_per_broker_scanner`, Количество узлов BE/CN)

В большинстве случаев для каждой задачи загрузки объявляется только один `data_desc`, каждая задача загрузки разделяется только на одну подзадачу, а подзадача разделяется

на такое же количество экземпляров, как количество узлов BE или CN.

Примеры

Этот раздел использует HDFS в качестве примера для описания различных конфигураций загрузки.

Загрузка CSV-данных

Этот раздел использует CSV в качестве примера для объяснения различных конфигураций параметров, которые вы можете использовать для удовлетворения ваших разнообразных требований к загрузке.

Установка периода тайм-аута

Ваша база данных Selena `test_db` содержит таблицу с именем `table1`. Таблица состоит из трех столбцов, которые представляют собой `col1`, `col2` и `col3` последовательно.

Ваш файл данных `example1.csv` также состоит из трех столбцов, которые сопоставляются последовательно с `col1`, `col2` и `col3` таблицы `table1`.

Если вы хотите загрузить все данные из `example1.csv` в `table1` в течение до 3600 секунд, выполните следующую команду:

```
LOAD LABEL test_db.label1
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/selena/data/input/example1.csv")
    INTO TABLE table1
)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
)
PROPERTIES
(
    "timeout" = "3600"
);
```

Установка максимальной допустимой ошибки

Ваша база данных Selena `test_db` содержит таблицу с именем `table2`. Таблица состоит из трех столбцов, которые представляют собой `col1`, `col2` и `col3` последовательно.

Ваш файл данных `example2.csv` также состоит из трех столбцов, которые сопоставляются последовательно с `col1`, `col2` и `col3` таблицы `table2`.

Если вы хотите загрузить все данные из `example2.csv` в `table2` с максимальной допустимой ошибкой `0.1`, выполните следующую команду:

```
LOAD LABEL test_db.label2
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/selena/data/input/example2.csv")
    INTO TABLE table2

)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
)
PROPERTIES
(
    "max_filter_ratio" = "0.1"
);
```

Загрузка всех файлов данных из пути к файлу

Ваша база данных Selena `test_db` содержит таблицу с именем `table3`. Таблица состоит из трех столбцов, которые представляют собой `col1`, `col2` и `col3` последовательно.

Все файлы данных, хранящиеся в пути `/user/selena/data/input/` вашего HDFS-кластера, также каждый состоят из трех столбцов, которые сопоставляются последовательно с `col1`, `col2` и `col3` таблицы `table3`. Разделитель столбцов, используемый в этих файлах данных, - `\x01`.

Если вы хотите загрузить данные из всех этих файлов данных, хранящихся в пути `hdfs://<hdfs_host>:<hdfs_port>/user/selena/data/input/`, в `table3`, выполните следующую команду:

```
LOAD LABEL test_db.label3
(
    DATA INFILE("hdfs://<hdfs_host>:<hdfs_port>/user/selena/data/input/*")
    INTO TABLE table3
    COLUMNS TERMINATED BY "\\x01"
)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
);
```

Установка механизма HA для NameNode

Ваша база данных Selena `test_db` содержит таблицу с именем `table4`. Таблица состоит из трех столбцов, которые представляют собой `col1`, `col2` и `col3` последовательно.

Ваш файл данных `example4.csv` также состоит из трех столбцов, которые сопоставляются с `col1`, `col2` и `col3` таблицы `table4`.

Если вы хотите загрузить все данные из `example4.csv` в `table4` с настроенным механизмом HA для NameNode, выполните следующую команду:

```
LOAD LABEL test_db.label4
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/selena/data/input/example4.csv")
    INTO TABLE table4
)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>",
    "dfs.nameservices" = "my_ha",
    "dfs.ha.namenodes.my_ha" = "my_namenode1,
my_namenode2","dfs.namenode.rpc-address.my_ha.my_namenode1" =
"nn1_host:rpc_port",
    "dfs.namenode.rpc-address.my_ha.my_namenode2" = "nn2_host:rpc_port",
    "dfs.client.failover.proxy.provider" =
"org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider"
);
```

Установка аутентификации Kerberos

Ваша база данных Selena `test_db` содержит таблицу с именем `table5`. Таблица состоит из трех столбцов, которые представляют собой `col1`, `col2` и `col3` последовательно.

Ваш файл данных `example5.csv` также состоит из трех столбцов, которые сопоставляются последовательно с `col1`, `col2` и `col3` таблицы `table5`.

Если вы хотите загрузить все данные из `example5.csv` в `table5` с настроенной аутентификацией Kerberos и указанным путем к файлу keytab, выполните следующую команду:

```
LOAD LABEL test_db.label5
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/selena/data/input/example5.csv")
    INTO TABLE table5
    COLUMNS TERMINATED BY "\t"
)
WITH BROKER
(
    "hadoop.security.authentication" = "kerberos",
    "kerberos_principal" = "selena@YOUR.COM",
    "kerberos_keytab" = "/home/selena/selena.keytab"
);
```

Отмена загрузки данных

Ваша база данных Selena `test_db` содержит таблицу с именем `table6`. Таблица состоит из трех столбцов, которые представляют собой `col1`, `col2` и `col3` последовательно.

Ваш файл данных `example6.csv` также состоит из трех столбцов, которые сопоставляются последовательно с `col1`, `col2` и `col3` таблицы `table6`.

Вы загрузили все данные из `example6.csv` в `table6`, запустив задачу Broker Load.

Если вы хотите отменить загруженные данные, выполните следующую команду:

```
LOAD LABEL test_db.label6
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/selena/data/input/example6.csv")
    NEGATIVE
    INTO TABLE table6
    COLUMNS TERMINATED BY "\t"
)
WITH BROKER
(
    "hadoop.security.authentication" = "kerberos",
    "kerberos_principal" = "selena@YOUR.COM",
    "kerberos_keytab" = "/home/selena/selena.keytab"
);
```

Указание целевых партиций

Ваша база данных Selena `test_db` содержит таблицу с именем `table7`. Таблица состоит из трех столбцов, которые представляют собой `col1`, `col2` и `col3` последовательно.

Ваш файл данных `example7.csv` также состоит из трех столбцов, которые сопоставляются последовательно с `col1`, `col2` и `col3` таблицы `table7`.

Если вы хотите загрузить все данные из `example7.csv` в две партиции, `p1` и `p2`, таблицы `table7`, выполните следующую команду:

```
LOAD LABEL test_db.label7
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/selena/data/input/example7.csv")
    INTO TABLE table7
    PARTITION (p1, p2)
    COLUMNS TERMINATED BY ","
)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
);
```

Настройка сопоставления столбцов

Ваша база данных Selena `test_db` содержит таблицу с именем `table8`. Таблица состоит из трех столбцов, которые представляют собой `col1`, `col2` и `col3` последовательно.

Ваш файл данных `example8.csv` также состоит из трех столбцов, которые сопоставляются последовательно с `col2`, `col1` и `col3` таблицы `table8`.

Если вы хотите загрузить все данные из `example8.csv` в `table8`, выполните следующую команду:

```
LOAD LABEL test_db.label8
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/selena/data/input/example8.csv")
    INTO TABLE table8
    COLUMNS TERMINATED BY ","
    (col2, col1, col3)
)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
);
```

> **NOTE**
>
> В приведенном выше примере столбцы `example8.csv` не могут быть сопоставлены со столбцами `table8` в той же последовательности, в которой эти столбцы расположены в `table8`. Поэтому вам нужно использовать `column_list` для настройки сопоставления столбцов между `example8.csv` и `table8`.

Установка условий фильтрации

Ваша база данных Selena `test_db` содержит таблицу с именем `table9`. Таблица состоит из трех столбцов, которые представляют собой `col1`, `col2` и `col3` последовательно.

Ваш файл данных `example9.csv` также состоит из трех столбцов, которые сопоставляются последовательно с `col1`, `col2` и `col3` таблицы `table9`.

Если вы хотите загрузить только записи данных, значения которых в первом столбце больше `20180601`, из `example9.csv` в `table9`, выполните следующую команду:

```
LOAD LABEL test_db.label9
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/selena/data/input/example9.csv")
    INTO TABLE table9
    (col1, col2, col3)
    where col1 > 20180601
)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
);
```

> **NOTE**
>
> В приведенном выше примере столбцы `example9.csv` могут быть сопоставлены со столбцами `table9` в той же последовательности, в которой эти столбцы расположены в `table9`, но вам нужно использовать предложение WHERE для указания условий фильтрации на основе столбцов. Поэтому вам нужно использовать `column_list` для настройки сопоставления столбцов между `example9.csv` и `table9`.

Загрузка данных в таблицы, содержащие столбцы типа HLL

Ваша база данных Selena `test_db` содержит таблицу с именем `table10`. Таблица состоит из четырех столбцов, которые представляют собой `id`, `col1`, `col2` и `col3` последовательно. `col1` и `col2` определены как столбцы типа HLL.

Ваш файл данных `example10.csv` состоит из трех столбцов, среди которых первый столбец сопоставляется с `id` таблицы `table10`, а второй и третий столбцы сопоставляются последовательно с `col1` и `col2` таблицы `table10`. Значения во втором и третьем столбцах `example10.csv` могут быть преобразованы в данные типа HLL с помощью функций перед загрузкой в `col1` и `col2` таблицы `table10`.

Если вы хотите загрузить все данные из `example10.csv` в `table10`, выполните следующую команду:

```
LOAD LABEL test_db.label10
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/selena/data/input/example10.csv")
    INTO TABLE table10
    COLUMNS TERMINATED BY ","
    (id, temp1, temp2)
    SET
    (
        col1 = hll_hash(temp1),
        col2 = hll_hash(temp2),
        col3 = empty_hll()
     )
 )
 WITH BROKER
 (
     "username" = "<hdfs_username>",
     "password" = "<hdfs_password>"
 );
```

> **NOTE**
>
> В приведенном выше примере три столбца `example10.csv` названы `id`, `temp1` и `temp2` последовательно с помощью `column_list`. Затем функции используются для преобразования данных следующим образом:
>
> - Функция `hll_hash` используется для преобразования значений в `temp1` и `temp2` `example10.csv` в данные типа HLL и сопоставления `temp1` и `temp2` `example10.csv` с `col1` и `col2` таблицы `table10`.
>
> - Функция `hll_empty` используется для заполнения указанного значения по умолчанию в `col3` таблицы `table10`.

Для использования функций `hll_hash` и `hll_empty` см. hll_hash и hll_empty.

Извлечение значений полей партиции из пути к файлу

Broker Load поддерживает анализ значений определенных полей партиции, содержащихся в пути к файлу, на основе определений столбцов целевой таблицы Selena. Эта функция Selena аналогична функции Partition Discovery Apache Spark™.

Ваша база данных Selena `test_db` содержит таблицу с именем `table11`. Таблица состоит из пяти столбцов, которые представляют собой `col1`, `col2`, `col3`, `city` и `utc_date` последовательно.

Путь к файлу `/user/selena/data/input/dir/city=beijing` вашего HDFS-кластера содержит следующие файлы данных:

- `/user/selena/data/input/dir/city=beijing/utc_date=2019-06-26/0000.csv`
- `/user/selena/data/input/dir/city=beijing/utc_date=2019-06-26/0001.csv`

Эти файлы данных каждый состоят из трех столбцов, которые сопоставляются последовательно с `col1`, `col2` и `col3` таблицы `table11`.

Если вы хотите загрузить данные из всех файлов данных из пути к файлу `/user/selena/data/input/dir/city=beijing/utc_date=*/*` в `table11` и, в то же время, вы хотите извлечь значения полей партиции `city` и `utc_date`, содержащихся в пути к файлу, и загрузить извлеченные значения в `city` и `utc_date` таблицы `table11`, выполните следующую команду:

```
LOAD LABEL test_db.label11
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/selena/data/input/dir/city=beijing/*/*")
    INTO TABLE table11
    FORMAT AS "csv"
    (col1, col2, col3)
    COLUMNS FROM PATH AS (city, utc_date)
    SET (uniq_id = md5sum(k1, city))
)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
);
```

Извлечение значений полей партиции из пути к файлу, содержащего `%3A`

В HDFS пути к файлам не могут содержать двоеточия (:). Все двоеточия (:) будут преобразованы в `%3A`.

Ваша база данных Selena `test_db` содержит таблицу с именем `table12`. Таблица состоит из трех столбцов, которые представляют собой `data_time`, `col1` и `col2` последовательно. Схема таблицы следующая:

```
data_time DATETIME,
col1        INT,
col2        INT
```

Путь к файлу `/user/selena/data` вашего HDFS-кластера содержит следующие файлы данных:

- `/user/selena/data/data_time=2020-02-17 00%3A00%3A00/example12.csv`
- `/user/selena/data/data_time=2020-02-18 00%3A00%3A00/example12.csv`

Если вы хотите загрузить все данные из `example12.csv` в `table12` и, в то же время, вы хотите извлечь значения поля партиции `data_time` из пути к файлу и загрузить извлеченные значения в `data_time` таблицы `table12`, выполните следующую команду:

```
LOAD LABEL test_db.label12
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/selena/data/*/example12.csv")
    INTO TABLE table12
    COLUMNS TERMINATED BY ","
    FORMAT AS "csv"
    (col1,col2)
    COLUMNS FROM PATH AS (data_time)
    SET (data_time = str_to_date(data_time, '%Y-%m-%d %H%%3A%i%%3A%s'))
)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
);
```

> **NOTE**
>
> В приведенном выше примере значения, извлеченные из поля партиции `data_time`, - это строки, содержащие `%3A`, такие как `2020-02-17 00%3A00%3A00`. Поэтому вам

> нужно использовать функцию `str_to_date` для преобразования строк в данные типа DATETIME перед их загрузкой в `data_time` таблицы `table8`.

Установка `format_type_options`

Ваша база данных Selena `test_db` содержит таблицу с именем `table13`. Таблица состоит из трех столбцов, которые представляют собой `col1`, `col2` и `col3` последовательно.

Ваш файл данных `example13.csv` также состоит из трех столбцов, которые сопоставляются последовательно с `col2`, `col1` и `col3` таблицы `table13`.

Если вы хотите загрузить все данные из `example13.csv` в `table13` с намерением пропустить первые две строки `example13.csv`, удалить пробелы перед и после разделителей столбцов и установить `enclose` в `\` и `escape` в `\`, выполните следующую команду:

```
LOAD LABEL test_db.label13
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/selena/data/*/example13.csv")
    INTO TABLE table13
    COLUMNS TERMINATED BY ","
    FORMAT AS "CSV"
    (
        skip_header = 2
        trim_space = TRUE
        enclose = "\""
        escape = "\\"
    )
    (col2, col1, col3)
)
WITH BROKER
(
    "username" = "hdfs_username",
    "password" = "hdfs_password"
)
PROPERTIES
(
    "timeout" = "3600"
);
```

Загрузка Parquet-данных

Этот раздел описывает некоторые настройки параметров, на которые вам нужно обратить внимание при загрузке Parquet-данных.

Ваша база данных Selena `test_db` содержит таблицу с именем `table13`. Таблица состоит из трех столбцов, которые представляют собой `col1`, `col2` и `col3` последовательно.

Ваш файл данных `example13.parquet` также состоит из трех столбцов, которые сопоставляются последовательно с `col1`, `col2` и `col3` таблицы `table13`.

Если вы хотите загрузить все данные из `example13.parquet` в `table13`, выполните следующую команду:

```
LOAD LABEL test_db.label13
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/selena/data/input/example13.parquet")
    INTO TABLE table13
    FORMAT AS "parquet"
    (col1, col2, col3)
)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
);
```

> **NOTE**
>
> По умолчанию, при загрузке Parquet-данных Selena определяет формат файла данных на основе того, содержит ли имя файла расширение **.parquet**. Если имя файла не содержит расширение **.parquet**, вы должны использовать `FORMAT AS` для указания формата файла данных как `Parquet`.

Загрузка ORC-данных

Этот раздел описывает некоторые настройки параметров, на которые вам нужно обратить внимание при загрузке ORC-данных.

Ваша база данных Selena `test_db` содержит таблицу с именем `table14`. Таблица состоит из трех столбцов, которые представляют собой `col1`, `col2` и `col3` последовательно.

Ваш файл данных `example14.orc` также содержит три столбца, которые сопоставляются последовательно с `col1`, `col2` и `col3` таблицы `table14`.

Если вы хотите загрузить все данные из `example14.orc` в `table14`, выполните следующую команду:

```
LOAD LABEL test_db.label14
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/selena/data/input/example14.orc")
    INTO TABLE table14
    FORMAT AS "orc"
    (col1, col2, col3)
)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
);
```

> **NOTE**
>
> - По умолчанию, при загрузке ORC-данных Selena определяет формат файла данных на основе того, содержит ли имя файла расширение **.orc**. Если имя файла не содержит расширение **.orc**, вы должны использовать `FORMAT AS` для указания формата файла данных как `ORC`.
> - В Selena v1.5.2 и более ранних, если файл данных содержит столбцы типа ARRAY, вы должны убедиться, что столбцы ORC-файла данных имеют те же имена, что и их сопоставляемые столбцы в таблице Selena, и столбцы не могут быть указаны в предложении SET.

Загрузка JSON-данных

Этот раздел описывает настройки параметров, на которые вам нужно обратить внимание при загрузке JSON-данных.

Ваша база данных Selena `test_db` содержит таблицу с именем `tbl1`, схема которой следующая:

```
`category` varchar(512) NULL COMMENT "",
`author` varchar(512) NULL COMMENT "",
`title` varchar(512) NULL COMMENT "",
`price` double NULL COMMENT ""
```

Загрузка JSON-данных с использованием простого режима

Предположим, что ваш файл данных `example1.json` состоит из следующих данных:

```
{"category":"C++","author":"avc","title":"C++ primer","price":895}
```

Чтобы загрузить все данные из `example1.json` в `tbl1`, выполните следующую команду:

```
LOAD LABEL test_db.label15
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/selena/data/input/example1.csv")
    INTO TABLE tbl1
    FORMAT AS "json"
)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
);
```

> **NOTE**
>
> В приведенном выше примере параметры `columns` и `jsonpaths` не указаны. Поэтому ключи в `example1.json` сопоставляются по имени со столбцами `tbl1`.

Загрузка JSON-данных с использованием режима соответствия

Selena выполняет следующие шаги для сопоставления и обработки JSON-данных:

1. (Опционально) Удаляет самую внешнюю структуру массива согласно настройке параметра `strip_outer_array`.

> **NOTE**
>
> Этот шаг выполняется только когда самый внешний слой JSON-данных - структура массива, обозначенная парой квадратных скобок `[]`. Вам нужно установить `strip_outer_array` в `true`.

2. (Опционально) Сопоставляет корневой элемент JSON-данных согласно настройке параметра `json_root`.

> **NOTE**
>
> Этот шаг выполняется только когда JSON-данные имеют корневой элемент. Вам нужно указать корневой элемент с помощью параметра `json_root`.

3. Извлекает указанные JSON-данные согласно настройке параметра `jsonpaths`.

Загрузка JSON-данных с использованием режима соответствия без указания корневого элемента

Предположим, что ваш файл данных `example2.json` состоит из следующих данных:

```
[
{"category":"xuxb111","author":"1avc","title":"SayingsoftheCentury","price":8
{"category":"xuxb222","author":"2avc","title":"SayingsoftheCentury","price":8
{"category":"xuxb333","author":"3avc","title":"SayingsoftheCentury","price":8
]
```

Чтобы загрузить только `category`, `author` и `price` из `example2.json`, выполните следующую команду:

```
LOAD LABEL test_db.label16
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/selena/data/input/example2.csv")
    INTO TABLE tbl1
    FORMAT AS "json"
    (category, price, author)
)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
)
PROPERTIES
(
    "strip_outer_array" = "true",
    "jsonpaths" = "[\"$.category\",\"$.price\",\"$.author\"]"
);
```

> **NOTE**
>
> В приведенном выше примере самый внешний слой JSON-данных - структура массива, обозначенная парой квадратных скобок `[]`. Структура массива состоит из нескольких JSON-объектов, каждый из которых представляет запись данных. Поэтому вам нужно установить `strip_outer_array` в `true`, чтобы удалить самую внешнюю структуру массива. Ключ **title**, который вы не хотите загружать, игнорируется во время загрузки.

Загрузка JSON-данных с использованием режима соответствия с указанием корневого элемента

Предположим, ваш файл данных `example3.json` состоит из следующих данных:

```
{
    "id": 10001,
    "RECORDS":[

{"category":"11","title":"SayingsoftheCentury","price":895,"timestamp":158919
        {"category":"22","author":"2avc","price":895,"timestamp":1589191487},

{"category":"33","author":"3avc","title":"SayingsoftheCentury","timestamp":15
    ],
    "comments": ["3 records", "there will be 3 rows"]
}
```

Чтобы загрузить только `category`, `author` и `price` из `example3.json`, выполните следующую команду:

```
LOAD LABEL test_db.label17
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/selena/data/input/example3.csv")
    INTO TABLE tbl1
    FORMAT AS "json"
    (category, price, author)
)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
)
PROPERTIES
(
    "json_root"="$.RECORDS",
    "strip_outer_array" = "true",
    "jsonpaths" = "[\"$.category\",\"$.price\",\"$.author\"]"
);
```

> **NOTE**
>
> В приведенном выше примере самый внешний слой JSON-данных - структура массива, обозначенная парой квадратных скобок `[]`. Структура массива состоит из нескольких JSON-объектов, каждый из которых представляет запись данных. Поэтому вам нужно установить `strip_outer_array` в `true`, чтобы удалить самую

> внешнюю структуру массива. Ключи `title` и `timestamp`, которые вы не хотите загружать, игнорируются во время загрузки. Кроме того, параметр `json_root` используется для указания корневого элемента, который является массивом, JSON-данных.

CANCEL LOAD

CANCEL LOAD отменяет заданное задание загрузки: Broker Load, Spark Load или INSERT. Задание загрузки в состоянии `PREPARED`, `CANCELLED` или `FINISHED` не может быть отменено.

Отмена задания загрузки является асинхронным процессом. Вы можете использовать оператор SHOW LOAD, чтобы проверить, было ли задание загрузки успешно отменено. Задание загрузки успешно отменено, если значение `State` равно `CANCELLED`, а значение `type` (отображается в `ErrorMsg`) равно `USER_CANCEL`.

Синтаксис

```
CANCEL LOAD
[FROM db_name]
WHERE LABEL = "label_name"
```

Параметры

Параметр	Обязательный	Описание
db_name	Нет	Имя базы данных, к которой принадлежит задание загрузки. Если этот параметр не указан, по умолчанию отменяется задание загрузки в текущей базе данных.
label_name	Да	Метка задания загрузки.

Примеры

Пример 1: Отмена задания загрузки с меткой `example_label` в текущей базе данных.

```
CANCEL LOAD
WHERE LABEL = "example_label";
```

Пример 2: Отмена задания загрузки с меткой `example_label` в базе данных `example_db`.

```
CANCEL LOAD
FROM example_db
WHERE LABEL = "example_label";
```

INSERT

Вставляет данные в определенную таблицу или перезаписывает определенную таблицу данными. Начиная с версии v1.5.2, INSERT поддерживает запись данных в файлы в удаленном хранилище. Вы можете использовать INSERT INTO FILES() для выгрузки данных из Selena в удаленное хранилище.

Вы можете отправить асинхронную задачу INSERT, используя SUBMIT TASK.

Синтаксис

- **Загрузка данных**:

```
INSERT { INTO | OVERWRITE } [db_name.]<table_name>
[ PARTITION (<partition_name> [, ...] ) ]
[ TEMPORARY PARTITION (<temporary_partition_name> [, ...] ) ]
[ WITH LABEL <label>]
[ (<column_name>[, ...]) | BY NAME ]
[ PROPERTIES ("key"="value", ...) ]
{ VALUES ( { <expression> | DEFAULT } [, ...] ) | <query> }
```

- **Выгрузка данных**:

```
INSERT INTO FILES()
[ WITH LABEL <label> ]
{ VALUES ( { <expression> | DEFAULT } [, ...] ) | <query> }
```

Параметры

Параметр	Описание
INTO	Для добавления данных в таблицу.
OVERWRITE	Для перезаписи таблицы данными.

Параметр	Описание
table_name	Имя таблицы, в которую вы хотите загрузить данные. Может быть указано с базой данных, в которой находится таблица, как `db_name.table_name`.
PARTITION	Партиции, в которые вы хотите загрузить данные. Вы можете указать несколько партиций, которые должны быть разделены запятыми (,). Должно быть установлено в партиции, которые существуют в целевой таблице. Если вы указываете этот параметр, данные будут вставлены только в указанные партиции. Если вы не указываете этот параметр, данные будут вставлены во все партиции.
TEMPORARY PARTITION	Имя временной партиции, в которую вы хотите загрузить данные. Вы можете указать несколько временных партиций, которые должны быть разделены запятыми (,).
label	Уникальная идентификационная метка для каждой транзакции загрузки данных в базе данных. Если она не указана, система автоматически генерирует её для транзакции. Мы рекомендуем указывать метку для транзакции. В противном случае вы не сможете проверить статус транзакции, если произойдет ошибка соединения и результат не будет возвращен. Вы можете проверить статус транзакции с помощью выражения `SHOW LOAD WHERE label="label"`. Для соглашений об именовании меток см. System Limits.
column_name	Имя целевого столбца (столбцов) для загрузки данных. Должно быть установлено как столбцы, которые существуют в целевой таблице. **Вы не можете указать одновременно `column_name` и `BY NAME`.** • Если `BY NAME` не указано, указанные вами целевые столбцы сопоставляются один к одному последовательно с исходными столбцами, независимо от того, какие имена у целевых столбцов. • Если `BY NAME` указано, целевые столбцы сопоставляются с исходными столбцами с теми же именами, независимо от порядка столбцов в целевой и исходной таблицах. • Если целевой столбец не указан, значение по умолчанию - все столбцы в целевой таблице. • Если указанный столбец в исходной таблице не существует в целевом столбце, в этот столбец будет записано значение по умолчанию. • Если указанный столбец не имеет значения по умолчанию, транзакция завершится неудачей. • Если тип столбца исходной таблицы несовместим с типом целевой таблицы, система выполнит неявное преобразование несоответствующего столбца. • Если преобразование завершится неудачей, будет возвращена ошибка синтаксического разбора. **NOTE** Начиная с версии v1.5.2, указание списка столбцов в выражении INSERT INTO для Primary Key table будет выполнять Partial Updates (вместо Full Upsert в более ранних версиях). Если список столбцов не указан, система выполнит Full Upsert.
BY NAME	Для сопоставления исходных и целевых столбцов по их именам. **Вы не можете указать одновременно `column_name` и `BY NAME`.** Если не указано, целевые столбцы сопоставляются один к одному последовательно с исходными столбцами, независимо от того, какие имена у целевых столбцов.

Параметр	Описание
PROPERTIES	Свойства задачи INSERT. Каждое свойство должно быть парой ключ-значение. Для поддерживаемых свойств см. PROPERTIES.
expression	Выражение, которое присваивает значения столбцу.
DEFAULT	Присваивает значение по умолчанию столбцу.
query	Запрос, результат которого будет загружен в целевую таблицу. Может быть любым SQL-выражением, поддерживаемым Selena.
FILES()	Табличная функция FILES(). Вы можете использовать эту функцию для выгрузки данных в удаленное хранилище.

PROPERTIES

Выражения INSERT поддерживают настройку PROPERTIES начиная с версии v1.5.2.

Свойство	Описание
timeout	Продолжительность тайм-аута задачи INSERT. Единица измерения: секунды. Вы также можете установить продолжительность тайм-аута для INSERT в рамках сессии или глобально, используя переменную `insert_timeout`.
strict_mode	Включать ли строгий режим при загрузке данных с помощью INSERT из FILES(). Допустимые значения: `true` (по умолчанию) и `false`. Когда строгий режим включен, система загружает только квалифицированные строки. Она фильтрует неквалифицированные строки и возвращает сведения о неквалифицированных строках. Для получения дополнительной информации см. Strict mode. Вы также можете включить строгий режим для INSERT из FILES() в рамках сессии или глобально, используя переменную `enable_insert_strict`.
max_filter_ratio	Максимальная допустимая ошибка INSERT из FILES(). Это максимальное соотношение записей данных, которые могут быть отфильтрованы из-за недостаточного качества данных. Когда соотношение неквалифицированных записей данных достигает этого порога, задача завершается неудачей. Значение по умолчанию: `0`. Диапазон: [0, 1]. Вы также можете установить максимальную допустимую ошибку для INSERT из FILES() в рамках сессии или глобально, используя переменную `insert_max_filter_ratio`.

> **ПРИМЕЧАНИЕ**
>
> - `strict_mode` и `max_filter_ratio` поддерживаются только для INSERT из FILES(). INSERT из таблиц не поддерживает эти свойства.
> - Начиная с версии v1.5.2, когда `enable_insert_strict` установлен в `true`, система загружает только квалифицированные строки. Она фильтрует неквалифицированные строки и возвращает сведения о неквалифицированных строках. Вместо этого, в версиях до v1.5.2, когда `enable_insert_strict` установлен в `true`, задачи INSERT завершаются неудачей, когда есть неквалифицированная строка.

Возвращаемое значение

```
Query OK, 5 rows affected, 2 warnings (0.05 sec)
{'label':'insert_load_test', 'status':'VISIBLE', 'txnId':'1008'}
```

Возвращаемое значение	Описание
rows affected	Указывает, сколько строк загружено. `warnings` указывает строки, которые были отфильтрованы.
label	Уникальная идентификационная метка для каждой транзакции загрузки данных в базе данных. Может быть назначена пользователем или автоматически системой.
status	Указывает, видны ли загруженные данные. `VISIBLE`: данные успешно загружены и видны. `COMMITTED`: данные успешно загружены, но пока не видны.
txnId	Идентификационный номер, соответствующий каждой транзакции INSERT.

Dynamic Overwrite

Начиная с версии v1.5.2, Selena поддерживает новую семантику - Dynamic Overwrite для INSERT OVERWRITE с партиционированными таблицами.

В настоящее время поведение INSERT OVERWRITE по умолчанию следующее:

- При перезаписи партиционированной таблицы целиком (то есть без указания предложения PARTITION) новые записи данных заменят данные в соответствующих партициях. Если есть партиции, которые не задействованы, они будут усечены, в то время как другие будут перезаписаны.

- При перезаписи пустой партиционированной таблицы (то есть без партиций в ней) и указании предложения PARTITION система возвращает ошибку `ERROR 1064 (HY000): Getting analyzing error. Detail message: Unknown partition 'xxx' in table 'yyy'`.
- При перезаписи партиционированной таблицы и указании несуществующей партиции в предложении PARTITION система возвращает ошибку `ERROR 1064 (HY000): Getting analyzing error. Detail message: Unknown partition 'xxx' in table 'yyy'`.
- При перезаписи партиционированной таблицы с записями данных, которые не соответствуют ни одной из указанных партиций в предложении PARTITION, система либо возвращает ошибку `ERROR 1064 (HY000): Insert has filtered data in strict mode` (если строгий режим включен), либо фильтрует неквалифицированные записи данных (если строгий режим отключен).

Поведение новой семантики Dynamic Overwrite сильно отличается:

При перезаписи партиционированной таблицы целиком новые записи данных заменят данные в соответствующих партициях. Если есть партиции, которые не задействованы, они останутся нетронутыми, вместо того чтобы быть усеченными или удаленными. И если есть новые записи данных, соответствующие несуществующей партиции, система создаст партицию.

Семантика Dynamic Overwrite отключена по умолчанию. Чтобы включить её, вам нужно установить системную переменную `dynamic_overwrite` в `true`.

Включить Dynamic Overwrite в текущей сессии:

```
SET dynamic_overwrite = true;
```

Вы также можете установить её в подсказке выражения INSERT OVERWRITE, чтобы она вступила в силу только для этого выражения.

Пример:

```
INSERT /*+set_var(dynamic_overwrite = true)*/ OVERWRITE insert_wiki_edit
SELECT * FROM source_wiki_edit;
```

Замечания по использованию

- Что касается текущей версии, когда Selena выполняет выражение INSERT INTO, если какая-либо строка данных не соответствует формату целевой таблицы (например, строка слишком длинная), транзакция INSERT завершается неудачей по умолчанию. Вы можете установить переменную сессии `enable_insert_strict` в `false`, чтобы система отфильтровала данные, которые не соответствуют формату целевой таблицы, и продолжила выполнение транзакции.

- После выполнения выражения INSERT OVERWRITE Selena создает временные партиции для партиций, которые хранят исходные данные, вставляет данные во временные партиции и меняет местами исходные партиции с временными партициями. Все эти операции выполняются в узле Leader FE. Поэтому, если узел Leader FE выходит из строя во время выполнения выражения INSERT OVERWRITE, вся транзакция загрузки завершается неудачей, и временные партиции удаляются.

- Selena поддерживает запись в не-выраженные партиции через выражение INSERT OVERWRITE. Однако при записи новых данных в партицию, которая ранее не существовала, система не создаст несуществующую партицию.

О Dynamic Overwrite

- Dynamic Overwrite поддерживает только выраженные партиции. При записи новых данных в партицию, которая ранее не существовала, система автоматически создаст несуществующую партицию.
- Dynamic Overwrite поддерживает партиции на основе смешанных выражений.

Примеры

Пример 1: Общее использование

Следующие примеры основаны на таблице `test`, которая содержит два столбца `c1` и `c2`. Столбец `c2` имеет значение по умолчанию DEFAULT.

- Импортировать одну строку данных в таблицу `test`.

```
INSERT INTO test VALUES (1, 2);
INSERT INTO test (c1, c2) VALUES (1, 2);
INSERT INTO test (c1, c2) VALUES (1, DEFAULT);
INSERT INTO test (c1) VALUES (1);
```

Когда целевой столбец не указан, столбцы загружаются последовательно в целевую таблицу по умолчанию. Поэтому в приведенном выше примере результаты первого и второго SQL-выражений одинаковы.

Если целевой столбец (с или без вставленных данных) использует DEFAULT в качестве значения, столбец будет использовать значение по умолчанию в качестве загруженных данных. Поэтому в приведенном выше примере результаты третьего и четвертого выражений одинаковы.

- Загрузить несколько строк данных в таблицу `test` за один раз.

```
INSERT INTO test VALUES (1, 2), (3, 2 + 2);
INSERT INTO test (c1, c2) VALUES (1, 2), (3, 2 * 2);
INSERT INTO test (c1, c2) VALUES (1, DEFAULT), (3, DEFAULT);
INSERT INTO test (c1) VALUES (1), (3);
```

Поскольку результаты выражений эквивалентны, результаты первого и второго выражений одинаковы. Результаты третьего и четвертого выражений одинаковы, потому что они оба используют значение по умолчанию.

- Импортировать результат запроса в таблицу `test`.

```
INSERT INTO test SELECT * FROM test2;
INSERT INTO test (c1, c2) SELECT * from test2;
```

- Импортировать результат запроса в таблицу `test` и указать партицию и метку.

```
INSERT INTO test PARTITION(p1, p2) WITH LABEL `label1` SELECT * FROM test2;
INSERT INTO test WITH LABEL `label1` (c1, c2) SELECT * from test2;
```

- Перезаписать таблицу `test` результатом запроса и указать партицию и метку.

```
INSERT OVERWRITE test PARTITION(p1, p2) WITH LABEL `label1` SELECT * FROM
test3;
INSERT OVERWRITE test WITH LABEL `label1` (c1, c2) SELECT * from test3;
```

Пример 2: Загрузка Parquet-файлов из AWS S3 с использованием INSERT из FILES()

Следующий пример вставляет строки данных из Parquet-файла **parquet/insert_wiki_edit_append.parquet** в AWS S3 bucket `inserttest` в таблицу `insert_wiki_edit`:

```
INSERT INTO insert_wiki_edit
    SELECT * FROM FILES(
        "path" = "s3://inserttest/parquet/insert_wiki_edit_append.parquet",
        "format" = "parquet",
        "aws.s3.access_key" = "XXXXXXXXXX",
        "aws.s3.secret_key" = "YYYYYYYYYY",
        "aws.s3.region" = "ap-southeast-1"
);
```

Пример 3: Тайм-аут INSERT

Следующий пример вставляет данные из исходной таблицы `source_wiki_edit` в целевую таблицу `insert_wiki_edit` с установленной продолжительностью тайм-аута в `2` секунды.

```
INSERT INTO insert_wiki_edit
PROPERTIES(
    "timeout" = "2"
)
SELECT * FROM source_wiki_edit;
```

Если вы хотите вставить большой набор данных, вы можете установить большее значение для `timeout` или для переменной сессии `insert_timeout`.

Пример 4: Строгий режим INSERT и максимальное соотношение фильтрации

Следующий пример вставляет строки данных из Parquet-файла **parquet/insert_wiki_edit_append.parquet** в AWS S3 bucket `inserttest` в таблицу `insert_wiki_edit`, включает строгий режим для фильтрации неквалифицированных записей данных и допускает максимум 10% ошибочных данных:

```
INSERT INTO insert_wiki_edit
PROPERTIES(
    "strict_mode" = "true",
    "max_filter_ratio" = "0.1"
)
SELECT * FROM FILES(
    "path" = "s3://inserttest/parquet/insert_wiki_edit_append.parquet",
    "format" = "parquet",
    "aws.s3.access_key" = "XXXXXXXXXX",
    "aws.s3.secret_key" = "YYYYYYYYYY",
    "aws.s3.region" = "us-west-2"
);
```

Пример 5: Сопоставление столбцов INSERT по имени

Следующий пример сопоставляет каждый столбец в исходной и целевой таблицах по их именам:

```
INSERT INTO insert_wiki_edit BY NAME
SELECT event_time, user, channel FROM source_wiki_edit;
```

В этом случае изменение порядка `channel` и `user` не изменит сопоставление столбцов.

SHOW LOAD

SHOW LOAD отображает информацию обо всех загрузочных заданиях или заданных загрузочных заданиях в базе данных. Эта команда может отображать только загрузочные задания, созданные с помощью Broker Load, INSERT и SPARK_LOAD. Вы также можете просматривать информацию о загрузочных заданиях с помощью команды `curl`. Начиная с версии 3.1, мы рекомендуем использовать команду SELECT для запроса результатов заданий Broker Load или Insert из таблицы `loads` в базе данных `information_schema`. Для получения дополнительной информации см. Loading.

Помимо вышеупомянутых методов загрузки, Selena поддерживает использование Stream Load и Routine Load для загрузки данных. Stream Load является синхронной операцией и будет напрямую возвращать информацию о заданиях Stream Load. Routine Load является асинхронной операцией, где вы можете использовать команду SHOW ROUTINE LOAD для отображения информации о заданиях Routine Load.

Синтаксис

```
SHOW LOAD [ FROM db_name ]
[
   WHERE [ LABEL { = "label_name" | LIKE "label_matcher" } ]
         [ [AND] STATE = { "PENDING" | "ETL" | "LOADING" | "FINISHED" |
"CANCELLED" } ]
]
[ ORDER BY field_name [ ASC | DESC ] ]
[ LIMIT { [offset, ] limit | limit OFFSET offset } ]
```

> **Примечание**
>
> Вы можете добавить опцию `\G` к команде (например, `SHOW LOAD WHERE LABEL = "label1"\G;`), чтобы отображать вывод вертикально, а не в обычном горизонтальном табличном формате. Для получения дополнительной информации см. Пример 1.

Параметры

Параметр	Обязательный	Описание
db_name	Нет	Имя базы данных. Если этот параметр не указан, по умолчанию используется текущая база данных.
LABEL = "label_name"	Нет	Метки загрузочных заданий.
LABEL LIKE "label_matcher"	Нет	Если этот параметр указан, возвращается информация о загрузочных заданиях, метки которых содержат `label_matcher`.
AND	Нет	• Если вы указываете только одно условие фильтра в предложении WHERE, не указывайте это ключевое слово. Пример: `WHERE STATE = "PENDING"`. • Если вы указываете два или три условия фильтра в предложении WHERE, вы должны указать это ключевое слово. Пример: `WHERE LABEL = "label_name" AND STATE = "PENDING"`.

STATE	Нет	Состояния загрузочных заданий. Состояния различаются в зависимости от методов загрузки.
		• Broker Load
		  ◦ `PENDING`: Загрузочное задание создано.
		  ◦ `QUEUEING`: Загрузочное задание находится в очереди в ожидании планирования.

Параметр	Обязательный	Описание
		◦ `LOADING`: Загрузочное задание выполняется. ◦ `PREPARED`: Транзакция зафиксирована. ◦ `FINISHED`: Загрузочное задание выполнено успешно. ◦ `CANCELLED`: Загрузочное задание завершилось с ошибкой. • Spark Load ◦ `PENDING`: Ваш кластер Selena готовит конфигурации, связанные с ETL, а затем отправляет задание ETL в ваш кластер Apache Spark™. ◦ `ETL`: Ваш кластер Spark выполняет задание ETL, а затем записывает данные в соответствующий кластер HDFS. ◦ `LOADING`: Данные из кластера HDFS загружаются в ваш кластер Selena, что означает, что загрузочное задание выполняется. ◦ `PREPARED`: Транзакция зафиксирована. ◦ `FINISHED`: Загрузочное задание выполнено успешно. ◦ `CANCELLED`: Загрузочное задание завершилось с ошибкой. • INSERT ◦ `FINISHED`: Загрузочное задание выполнено успешно. ◦ `CANCELLED`: Загрузочное задание завершилось с ошибкой. Если параметр `STATE` не указан, по умолчанию возвращается информация о загрузочных заданиях во всех состояниях. Если параметр `STATE` указан, возвращается только информация о загрузочных заданиях в заданном состоянии. Например, `STATE = "PENDING"` возвращает информацию о загрузочных заданиях в состоянии `PENDING`.
ORDER BY field_name [ASC | DESC]	Нет	Если этот параметр указан, вывод сортируется по возрастанию или убыванию на основе поля. Поддерживаются следующие поля: `JobId`, `Label`, `State`, `Progress`, `Type`, `EtlInfo`, `TaskInfo`, `ErrorMsg`, `CreateTime`, `EtlStartTime`, `EtlFinishTime`, `LoadStartTime`, `LoadFinishTime`, `URL` и `JobDetails`. • Для сортировки вывода по возрастанию укажите `ORDER BY field_name ASC`. • Для сортировки вывода по убыванию укажите `ORDER BY field_name DESC`. Если вы не указываете поле и порядок сортировки, вывод по умолчанию сортируется по возрастанию `JobId`.
LIMIT limit	Нет	Количество загрузочных заданий, которые разрешено отображать. Если этот параметр не указан, отображается информация обо всех загрузочных заданиях, соответствующих условиям фильтра. Если этот параметр указан, например, `LIMIT 10`, возвращается только информация о 10 загрузочных заданиях, соответствующих условиям фильтра.
OFFSET offset	Нет	Параметр `offset` определяет количество загрузочных заданий, которые нужно пропустить. Например, `OFFSET 5` пропускает первые пять загрузочных заданий и возвращает остальные. Значение параметра `offset` по умолчанию равно `0`.

Вывод

```
+-------+-------+-------+----------+------+---------+----------+----------
+------------+--------------+--------------+--------------+--------------
--+-----+------------+
| JobId | Label | State | Progress | Type | Priority | EtlInfo | TaskInfo |
ErrorMsg | CreateTime | EtlStartTime | EtlFinishTime | LoadStartTime |
LoadFinishTime | URL | JobDetails |
+-------+-------+-------+----------+------+---------+----------+----------
+------------+--------------+--------------+--------------+--------------
--+-----+------------+
```

Вывод этой команды различается в зависимости от методов загрузки.

Поле	Broker Load	Spark Load	INSERT
JobId	Уникальный ID, присвоенный Selena для идентификации загрузочного задания в вашем кластере Selena.	Поле имеет то же значение в задании Spark Load, что и в задании Broker Load.	Поле имеет то же значение в задании INSERT, что и в задании Broker Load.
Label	Метка загрузочного задания. Метка загрузочного задания уникальна в пределах базы данных, но может дублироваться в разных базах данных.	Поле имеет то же значение в задании Spark Load, что и в задании Broker Load.	Поле имеет то же значение в задании INSERT, что и в задании Broker Load.
State	Состояние загрузочного задания. • `PENDING`: Загрузочное задание создано. • `QUEUEING`: Загрузочное задание находится в очереди в ожидании планирования. • `LOADING`: Загрузочное задание выполняется. • `PREPARED`: Транзакция зафиксирована. • `FINISHED`: Загрузочное задание выполнено успешно. • `CANCELLED`: Загрузочное задание завершилось с ошибкой.	Состояние загрузочного задания. • `PENDING`: Ваш кластер Selena готовит конфигурации, связанные с ETL, а затем отправляет задание ETL в ваш кластер Spark. • `ETL`: Ваш кластер Spark выполняет задание ETL, а затем записывает данные в соответствующий кластер HDFS. • `LOADING`: Данные из кластера HDFS загружаются в ваш кластер Selena, что означает, что	Состояние загрузочного задания. • `FINISHED`: Загрузочное задание выполнено успешно. • `CANCELLED`: Загрузочное задание завершилось с ошибкой.

Поле	Broker Load	Spark Load	INSERT
		загрузочное задание выполняется. • `PREPARED`: Транзакция зафиксирована. • `FINISHED`: Загрузочное задание выполнено успешно. • `CANCELLED`: Загрузочное задание завершилось с ошибкой.	
Progress	Стадия загрузочного задания. Задание Broker Load имеет только стадию `LOAD`, которая варьируется от 0% до 100% для описания прогресса стадии. Когда загрузочное задание входит в стадию `LOAD`, для параметра `State` возвращается `LOADING`. Задание Broker Load не имеет стадии `ETL`. Параметр `ETL` действителен только для задания Spark Load. **Примечание** • Формула для расчета прогресса стадии `LOAD`: Количество таблиц Selena, которые завершили загрузку данных/Количество таблиц Selena, в которые вы планируете загрузить данные * 100%. • Когда все данные загружены в Selena, для параметра `LOAD` возвращается `99%`. Затем загруженные данные начинают вступать в силу в Selena. После того как данные вступят в силу, для параметра `LOAD` возвращается `100%`. • Прогресс стадии `LOAD` не является линейным. Поэтому	Стадия загрузочного задания. Задание Spark Load имеет две стадии: • `ETL`: варьируется от 0% до 100% для описания прогресса стадии `ETL`. • `LOAD`: варьируется от 0% до 100% для описания прогресса стадии `Load`. Когда загрузочное задание входит в стадию `ETL`, для параметра `State` возвращается `ETL`. Когда загрузочное задание переходит в стадию `LOAD`, для параметра `State` возвращается `LOADING`. **Примечание** такое же, как для Broker Load.	Стадия загрузочного задания. Задание INSERT имеет только стадию `LOAD`, которая варьируется от 0% до 100% для описания прогресса стадии. Когда загрузочное задание входит в стадию `LOAD`, для параметра `State` возвращается `LOADING`. Задание INSERT не имеет стадии `ETL`. Параметр `ETL` действителен только для задания Spark Load. **Примечание** такое же, как для Broker Load.

Поле	Broker Load	Spark Load	INSERT
	значение параметра `LOAD` может не изменяться в течение определенного периода времени, даже если загрузка данных все еще продолжается.		
Type	Метод загрузочного задания. Значение этого параметра по умолчанию равно `BROKER`.	Метод загрузочного задания. Значение этого параметра по умолчанию равно `SPARK`.	Метод загрузочного задания. Значение этого параметра по умолчанию равно `INSERT`.
Priority	Приоритет загрузочного задания. Допустимые значения: LOWEST, LOW, NORMAL, HIGH и HIGHEST.	-	-
EtlInfo	Метрики, связанные с ETL. • `unselected.rows`: Количество строк, отфильтрованных предложением WHERE. • `dpp.abnorm.ALL`: Количество строк, отфильтрованных из-за проблем с качеством данных, которое относится к несоответствиям между исходными таблицами и таблицами Selena в, например, типе данных и количестве столбцов. • `dpp.norm.ALL`: Количество строк, загруженных в ваш кластер Selena. Сумма вышеуказанных метрик представляет собой общее количество строк необработанных данных. Вы можете использовать следующую формулу для расчета того, превышает ли процент некачественных данных значение параметра `max-filter-ratio`: `dpp.abnorm.ALL`/(`unselected.rows` + `dpp.abnorm.ALL` + `dpp.norm.ALL`).	Поле имеет то же значение в задании Spark Load, что и в задании Broker Load.	Метрики, связанные с ETL. Задание INSERT не имеет стадии `ETL`. Поэтому возвращается `NULL`.
Task	Параметры, которые указываются при создании загрузочного	Параметры, которые указываются при создании	Параметры, которые указываются при создании загрузочного

Поле	Broker Load	Spark Load	INSERT
Info	задания. • `resource`: Этот параметр действителен только в задании Spark Load. • `timeout`: Период времени, в течение которого загрузочному заданию разрешено выполняться. Единица измерения: секунды. • `max-filter-ratio`: Наибольший процент строк, отфильтрованных из-за проблем с качеством данных. Для получения дополнительной информации см. BROKER LOAD.	загрузочного задания. • `resource`: Имя ресурса. • `timeout`: Период времени, в течение которого загрузочному заданию разрешено выполняться. Единица измерения: секунды. • `max-filter-ratio`: Наибольший процент строк, отфильтрованных из-за проблем с качеством данных. Для получения дополнительной информации см. SPARK LOAD.	задания. • `resource`: Этот параметр действителен только в задании Spark Load. • `timeout`: Период времени, в течение которого загрузочному заданию разрешено выполняться. Единица измерения: секунды. • `max-filter-ratio`: Наибольший процент строк, отфильтрованных из-за проблем с качеством данных. Для получения дополнительной информации см. INSERT.

ErrorMsg	Сообщение об ошибке, возвращаемое при сбое загрузочного задания. Когда состояние загрузочного задания `PENDING`, `LOADING` или `FINISHED`, для поля `ErrorMsg` возвращается `NULL`. Когда состояние загрузочного задания `CANCELLED`, значение, возвращаемое для поля `ErrorMsg`, состоит из двух частей: `type` и `msg`.
	• Часть `type` может быть любым из следующих значений:
	  ◦ `USER_CANCEL`: Загрузочное задание было отменено вручную.
	  ◦ `ETL_SUBMIT_FAIL`: Не удалось отправить загрузочное задание.
	  ◦ `ETL-QUALITY-UNSATISFIED`: Загрузочное задание завершилось с ошибкой, поскольку процент
		Сообщение об ошибке, возвращаемое при сбое загрузочного задания. Когда состояние загрузочного задания `PENDING`, `LOADING` или `FINISHED`, для поля `ErrorMsg` возвращается `NULL`. Когда состояние загрузочного задания `CANCELLED`, значение, возвращаемое для поля `ErrorMsg`, состоит из двух частей: `type` и `msg`.
		• Часть `type` может быть любым из следующих значений:
		  ◦ `USER_CANCEL`: Загрузочное задание было отменено вручную.
		  ◦ `ETL_SUBMIT_FAIL`: Selena не удалось отправить задание ETL в Spark.
		  ◦ `ETL-RUN-FAIL`: Spark не удалось выполнить
			Сообщение об ошибке, возвращаемое при сбое загрузочного задания. Когда состояние загрузочного задания `FINISHED`, для поля `ErrorMsg` возвращается `NULL`. Когда состояние загрузочного задания `CANCELLED`, значение, возвращаемое для поля `ErrorMsg`, состоит из двух частей: `type` и `msg`.
			• Часть `type` может быть любым из следующих значений:
			  ◦ `USER_CANCEL`: Загрузочное задание было отменено вручную.
			  ◦ `ETL_SUBMIT_FAIL`: Не удалось отправить загрузочное задание.
			  ◦ `ETL_RUN_FAIL`: Не удалось запустить загрузочное задание.

Поле	Broker Load	Spark Load	INSERT
	некачественных данных превышает значение параметра `max-filter-ratio`. ○ `LOAD-RUN-FAIL`: Загрузочное задание завершилось с ошибкой на стадии `LOAD`. ○ `TIMEOUT`: Загрузочное задание не удалось завершить в течение указанного периода времени ожидания. ○ `UNKNOWN`: Загрузочное задание завершилось с ошибкой из-за неизвестной ошибки. • Часть `msg` содержит подробную причину сбоя загрузки.	задание ETL. ○ `ETL-QUALITY-UNSATISFIED`: Загрузочное задание завершилось с ошибкой, поскольку процент некачественных данных превышает значение параметра `max-filter-ratio`. ○ `LOAD-RUN-FAIL`: Загрузочное задание завершилось с ошибкой на стадии `LOAD`. ○ `TIMEOUT`: Загрузочное задание не удалось завершить в течение указанного периода времени ожидания. ○ `UNKNOWN`: Загрузочное задание завершилось с ошибкой из-за неизвестной ошибки. • Часть `msg` содержит подробную причину сбоя загрузки.	○ `ETL_QUALITY_UNSATISFIED`: Загрузочное задание завершилось с ошибкой из-за проблем с качеством необработанных данных. ○ `LOAD-RUN-FAIL`: Загрузочное задание завершилось с ошибкой на стадии `LOAD`. ○ `TIMEOUT`: Загрузочное задание не удалось завершить в течение указанного периода времени ожидания. ○ `UNKNOWN`: Загрузочное задание завершилось с ошибкой из-за неизвестной ошибки. ○ `TXN_UNKNOWN`: Загрузочное задание завершилось с ошибкой, поскольку состояние транзакции загрузочного задания неизвестно. • Часть `msg` содержит подробную причину сбоя загрузки.
CreateTime	Время, когда было создано загрузочное задание.	Поле имеет то же значение в задании Spark Load, что и в задании Broker Load.	Поле имеет то же значение в задании INSERT, что и в задании Broker Load.
EtlStartTi	Задание Broker Load не имеет стадии `ETL`. Поэтому значение этого поля совпадает со значением поля `LoadStartTime`.	Время начала стадии `ETL`.	Задание INSERT не имеет стадии `ETL`. Поэтому значение этого поля совпадает со значением поля `LoadStartTime`.

Поле	Broker Load	Spark Load	INSERT
me			
EtlFinishTime	Задание Broker Load не имеет стадии `ETL`. Поэтому значение этого поля совпадает со значением поля `LoadStartTime`.	Время завершения стадии `ETL`.	Задание INSERT не имеет стадии `ETL`. Поэтому значение этого поля совпадает со значением поля `LoadStartTime`.
LoadStartTime	Время начала стадии `LOAD`.	Поле имеет то же значение в задании Spark Load, что и в задании Broker Load.	Поле имеет то же значение в задании INSERT, что и в задании Broker Load.
LoadFinishTime	Время завершения загрузочного задания.	Поле имеет то же значение в задании Spark Load, что и в задании Broker Load.	Поле имеет то же значение в задании INSERT, что и в задании Broker Load.
URL	URL-адрес, который используется для доступа к некачественным данным, обнаруженным в загрузочном задании. Вы можете использовать команду `curl` или `wget` для доступа к URL-адресу и получения некачественных данных. Если некачественные данные не обнаружены, возвращается `NULL`.	Поле имеет то же значение в задании Spark Load, что и в задании Broker Load.	Поле имеет то же значение в задании INSERT, что и в задании Broker Load.
JobDetails	Другая информация, связанная с загрузочным заданием. • `Unfinished backends`: ID BE, который не завершил загрузку данных. • `ScannedRows`: Общее количество строк,	Поле имеет то же значение в задании Spark Load, что и в задании Broker Load.	Поле имеет то же значение в задании INSERT, что и в задании Broker Load.

Поле	Broker Load	Spark Load	INSERT
	загруженных в Selena, и количество строк, которые были отфильтрованы. • `TaskNumber`: Загрузочное задание может быть разделено на одну или несколько задач, которые выполняются параллельно. Это поле указывает количество загрузочных задач. • `All backends`: ID BE, который выполняет загрузку данных. • `FileNumber`: Количество файлов исходных данных. • `FileSize`: Объем данных файлов исходных данных. Единица измерения: байты.		

Примечания по использованию

- Информация, возвращаемая командой SHOW LOAD, действительна в течение 3 дней с момента `LoadFinishTime` загрузочного задания. По истечении 3 дней информация не может быть отображена. Вы можете использовать параметр `label_keep_max_second` для изменения периода действия по умолчанию.

```
ADMIN SET FRONTEND CONFIG ("label_keep_max_second" = "value");
```

- Если значение поля `LoadStartTime` долгое время остается `N/A`, это означает, что загрузочные задания сильно накапливаются. Мы рекомендуем сократить частоту создания загрузочных заданий.

- Общий период времени, затраченный на загрузочное задание = `LoadFinishTime` - `CreateTime`.

- Общий период времени, затраченный загрузочным заданием на стадии `LOAD` = `LoadFinishTime` - `LoadStartTime`.

Примеры

Пример 1: Вертикально отобразить все загрузочные задания в текущей базе данных.

```
SHOW LOAD\G
*************************** 1. row ***************************
         JobId: 976331
         Label: duplicate_table_with_null
         State: FINISHED
      Progress: ETL:100%; LOAD:100%
          Type: BROKER
      Priority: NORMAL
       EtlInfo: unselected.rows=0; dpp.abnorm.ALL=0; dpp.norm.ALL=65546
      TaskInfo: resource:N/A; timeout(s):300; max_filter_ratio:0.0
      ErrorMsg: NULL
    CreateTime: 2022-10-17 19:35:00
  EtlStartTime: 2022-10-17 19:35:04
 EtlFinishTime: 2022-10-17 19:35:04
 LoadStartTime: 2022-10-17 19:35:04
LoadFinishTime: 2022-10-17 19:35:06
           URL: NULL
    JobDetails: {"Unfinished backends":{"b90a703c-6e5a-4fcb-a8e1-
94eca5be0b8f":[]},"ScannedRows":65546,"TaskNumber":1,"All backends":
{"b90a703c-6e5a-4fcb-a8e1-94eca5be0b8f":
[10004]},"FileNumber":1,"FileSize":548622}
```

Пример 2: Отобразить два загрузочных задания, метки которых содержат строку `null` в текущей базе данных.

```
SHOW LOAD
WHERE LABEL LIKE "null"
LIMIT 2;

+-------+---------------------------+----------+---------------------+-----
---+-------------------------------------------------------------+-------------
-----------------------------------------------------------------------------
-----------------+----------+---------------------+---------------------+--
-------------------+---------------------+---------------------+-----------
-------------------------------------------------------------------------+-----
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
----------------------------------------------+
| JobId | Label                     | State    | Progress            | Type
| EtlInfo                                                     | TaskInfo
| ErrorMsg | CreateTime          | EtlStartTime        | EtlFinishTime
| LoadStartTime       | LoadFinishTime      | URL
| JobDetails
|
+-------+---------------------------+----------+---------------------+-----
---+-------------------------------------------------------------+-------------
-----------------------------------------------------------------------------
-----------------+----------+---------------------+---------------------+--
-------------------+---------------------+---------------------+-----------
-------------------------------------------------------------------------+-----
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
----------------------------------------------+
| 10082 | duplicate_table_with_null | FINISHED | ETL:100%; LOAD:100% |
BROKER | unselected.rows=0; dpp.abnorm.ALL=0; dpp.norm.ALL=65546 |
resource:N/A; timeout(s):300; max_filter_ratio:0.0
| NULL     | 2022-08-02 14:53:27 | 2022-08-02 14:53:30 | 2022-08-02
14:53:30 | 2022-08-02 14:53:30 | 2022-08-02 14:53:31 | NULL
| {"Unfinished backends":{"4393c992-5da1-4e9f-8b03-895dc0c96dbc":
[]},"ScannedRows":65546,"TaskNumber":1,"All backends":{"4393c992-5da1-4e9f-
8b03-895dc0c96dbc":[10002]},"FileNumber":1,"FileSize":548622} |
| 10103 | unique_table_with_null    | FINISHED | ETL:100%; LOAD:100% |
SPARK  | unselected.rows=0; dpp.abnorm.ALL=0; dpp.norm.ALL=65546 |
resource:test_spark_resource_07af473a_1230_11ed_b483_00163e0e550b;
timeout(s):300; max_filter_ratio:0.0 | NULL     | 2022-08-02 14:56:06 |
2022-08-02 14:56:19 | 2022-08-02 14:56:41 | 2022-08-02 14:56:41 | 2022-08-
02 14:56:44 | http://emr-header-1.cluster-
49091:20888/proxy/application_1655710334658_26391/ | {"Unfinished
backends":{"00000000-0000-0000-0000-000000000000":
[]},"ScannedRows":65546,"TaskNumber":1,"All backends":{"00000000-0000-0000-
```

```
0000-000000000000":[-1]},"FileNumber":1,"FileSize":8790855}   |
+-------+---------------------------+----------+---------------------+-----
---+-------------------------------------------------------------+-------------
-----------------------------------------------------------------------------
-----------------+----------+---------------------+---------------------+--
-------------------+---------------------+---------------------+-----------
-----------------------------------------------------------------------+-----
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
----------------------------------------------+
```

Пример 3: Отобразить загрузочные задания, метки которых содержат строку `table` в `example_db`. Кроме того, возвращаемые загрузочные задания отображаются в порядке убывания поля `LoadStartTime`.

```
SHOW LOAD FROM example_db
WHERE LABEL Like "table"
ORDER BY LoadStartTime DESC;

+-------+---------------------------+----------+---------------------+-----
---+-------------------------------------------------------------+-------------
-----------------------------------------------------------------------------
-----------------+----------+---------------------+---------------------+--
-------------------+---------------------+---------------------+-----------
-----------------------------------------------------------------------+-----
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
----------------------------------------------+
| JobId | Label                     | State    | Progress            | Type
| EtlInfo                                                     | TaskInfo
| ErrorMsg | CreateTime          | EtlStartTime        | EtlFinishTime
| LoadStartTime       | LoadFinishTime      | URL
| JobDetails
|
+-------+---------------------------+----------+---------------------+-----
---+-------------------------------------------------------------+-------------
-----------------------------------------------------------------------------
-----------------+----------+---------------------+---------------------+--
-------------------+---------------------+---------------------+-----------
-----------------------------------------------------------------------+-----
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
----------------------------------------------+
| 10103 | unique_table_with_null    | FINISHED | ETL:100%; LOAD:100% |
SPARK  | unselected.rows=0; dpp.abnorm.ALL=0; dpp.norm.ALL=65546 |
resource:test_spark_resource_07af473a_1230_11ed_b483_00163e0e550b;
timeout(s):300; max_filter_ratio:0.0 | NULL     | 2022-08-02 14:56:06 |
2022-08-02 14:56:19 | 2022-08-02 14:56:41 | 2022-08-02 14:56:41 | 2022-08-
02 14:56:44 | http://emr-header-1.cluster-
49091:20888/proxy/application_1655710334658_26391/ | {"Unfinished
backends":{"00000000-0000-0000-0000-000000000000":
[]},"ScannedRows":65546,"TaskNumber":1,"All backends":{"00000000-0000-0000-
0000-000000000000":[-1]},"FileNumber":1,"FileSize":8790855}   |
| 10082 | duplicate_table_with_null | FINISHED | ETL:100%; LOAD:100% |
BROKER | unselected.rows=0; dpp.abnorm.ALL=0; dpp.norm.ALL=65546 |
resource:N/A; timeout(s):300; max_filter_ratio:0.0
| NULL     | 2022-08-02 14:53:27 | 2022-08-02 14:53:30 | 2022-08-02
14:53:30 | 2022-08-02 14:53:30 | 2022-08-02 14:53:31 | NULL
| {"Unfinished backends":{"4393c992-5da1-4e9f-8b03-895dc0c96dbc":
[]},"ScannedRows":65546,"TaskNumber":1,"All backends":{"4393c992-5da1-4e9f-
```

```
8b03-895dc0c96dbc":[10002]},"FileNumber":1,"FileSize":548622} |
+-------+---------------------------+----------+---------------------+-----
---+-------------------------------------------------------------+-------------
-----------------------------------------------------------------------------
-----------------+----------+---------------------+---------------------+--
-------------------+---------------------+---------------------+-----------
-----------------------------------------------------------------------+-----
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
----------------------------------------------+
```

Пример 4: Отобразить загрузочное задание с меткой `duplicate_table_with_null` и состоянием `FINISHED` в `example_db`.

```
SHOW LOAD FROM example_db
WHERE LABEL = "duplicate_table_with_null" AND STATE = "FINISHED";

+-------+---------------------------+----------+---------------------+-----
---+-------------------------------------------------------------+-------------
---------------------------------------+----------+---------------------+--
-------------------+---------------------+---------------------+-----------
----------+------+-------------------------------------------------------------
-----------------------------------------------------------------------------
-------------------------------------------------------------------------+
| JobId | Label                     | State    | Progress            | Type
| EtlInfo                                                     | TaskInfo
| ErrorMsg | CreateTime          | EtlStartTime        | EtlFinishTime
| LoadStartTime       | LoadFinishTime      | URL  | JobDetails
|
+-------+---------------------------+----------+---------------------+-----
---+-------------------------------------------------------------+-------------
---------------------------------------+----------+---------------------+--
-------------------+---------------------+---------------------+-----------
----------+------+-------------------------------------------------------------
-----------------------------------------------------------------------------
-------------------------------------------------------------------------+
| 10082 | duplicate_table_with_null | FINISHED | ETL:100%; LOAD:100% |
BROKER | unselected.rows=0; dpp.abnorm.ALL=0; dpp.norm.ALL=65546 |
resource:N/A; timeout(s):300; max_filter_ratio:0.0 | NULL     | 2022-08-02
14:53:27 | 2022-08-02 14:53:30 | 2022-08-02 14:53:30 | 2022-08-02 14:53:30
| 2022-08-02 14:53:31 | NULL | {"Unfinished backends":{"4393c992-5da1-4e9f-
8b03-895dc0c96dbc":[]},"ScannedRows":65546,"TaskNumber":1,"All backends":
{"4393c992-5da1-4e9f-8b03-895dc0c96dbc":
[10002]},"FileNumber":1,"FileSize":548622} |
+-------+---------------------------+----------+---------------------+-----
---+-------------------------------------------------------------+-------------
---------------------------------------+----------+---------------------+--
-------------------+---------------------+---------------------+-----------
----------+------+-------------------------------------------------------------
-----------------------------------------------------------------------------
-------------------------------------------------------------------------+
```

Пример 5: Пропустить первое загрузочное задание и отобразить следующие два загрузочных задания. Кроме того, эти два загрузочных задания сортируются по возрастанию.

```
SHOW LOAD FROM example_db
ORDER BY CreateTime ASC
LIMIT 2 OFFSET 1;
```

Или

```
SHOW LOAD FROM example_db
ORDER BY CreateTime ASC
LIMIT 1,2;
```

Вывод вышеуказанных команд выглядит следующим образом.

```
+-------+----------------------------------------------+----------+---------
------------+--------+-------------------------------------------------------
----+-----------------------------------------------------------------------
-----------------------------------+----------+---------------------+------
---------------+---------------------+---------------------+---------------
------+-----------------------------------------------------------------------
------------+----------------------------------------------------------------
-----------------------------------------------------------------------------
-------------------------------------------------------------+
| JobId | Label                                          | State    | Progress
| Type   | EtlInfo                                                  |
TaskInfo
| ErrorMsg | CreateTime          | EtlStartTime        | EtlFinishTime
| LoadStartTime       | LoadFinishTime      | URL
| JobDetails
|
+-------+----------------------------------------------+----------+---------
------------+--------+-------------------------------------------------------
----+-----------------------------------------------------------------------
-----------------------------------+----------+---------------------+------
---------------+---------------------+---------------------+---------------
------+-----------------------------------------------------------------------
------------+----------------------------------------------------------------
-----------------------------------------------------------------------------
-------------------------------------------------------------+
| 10103 | unique_table_with_null                         | FINISHED |
ETL:100%; LOAD:100% | SPARK  | unselected.rows=0; dpp.abnorm.ALL=0;
dpp.norm.ALL=65546 |
resource:test_spark_resource_07af473a_1230_11ed_b483_00163e0e550b;
timeout(s):300; max_filter_ratio:0.0 | NULL     | 2022-08-02 14:56:06 |
2022-08-02 14:56:19 | 2022-08-02 14:56:41 | 2022-08-02 14:56:41 | 2022-08-
02 14:56:44 | http://emr-header-1.cluster-
49091:20888/proxy/application_1655710334658_26391/ | {"Unfinished
backends":{"00000000-0000-0000-0000-000000000000":
[]},"ScannedRows":65546,"TaskNumber":1,"All backends":{"00000000-0000-0000-
0000-000000000000":[-1]},"FileNumber":1,"FileSize":8790855} |
| 10120 | insert_3a57b595-1230-11ed-b075-00163e14c85e | FINISHED |
ETL:100%; LOAD:100% | INSERT | NULL
| resource:N/A; timeout(s):3600; max_filter_ratio:0.0
| NULL     | 2022-08-02 14:56:26 | 2022-08-02 14:56:26 | 2022-08-02
14:56:26 | 2022-08-02 14:56:26 | 2022-08-02 14:56:26 |
| {"Unfinished backends":{},"ScannedRows":0,"TaskNumber":0,"All backends":
{},"FileNumber":0,"FileSize":0}
|
+-------+----------------------------------------------+----------+---------
```

```
------------+--------+-------------------------------------------------------
----+-----------------------------------------------------------------------
-----------------------------------+----------+---------------------+------
---------------+---------------------+---------------------+---------------
------+-----------------------------------------------------------------------
------------+----------------------------------------------------------------
-----------------------------------------------------------------------------
-------------------------------------------------------------+
```

SHOW TRANSACTION

SHOW TRANSACTION используется для просмотра деталей транзакции по указанному идентификатору транзакции.

Синтаксис:

```
SHOW TRANSACTION
[FROM <db_name>]
WHERE id = transaction_id
```

Пример возвращаемых результатов:

```
TransactionId: 4005
Label: insert_8d807d5d-bcdd-46eb-be6d-3fa87aa4952d
Coordinator: FE: 10.74.167.16
TransactionStatus: VISIBLE
LoadJobSourceType: INSERT_STREAMING
PrepareTime: 2020-01-09 14:59:07
PreparedTime: 2020-01-09 14:59:08
CommitTime: 2020-01-09 14:59:09
FinishTime: 2020-01-09 14:59:09
Reason:
ErrorReplicasCount: 0
ListenerId: -1
TimeoutMs: 300000
PreparedTimeoutMs: 86400000
```

- TransactionId: идентификатор транзакции
- Label: метка, соответствующая задаче импорта
- Coordinator: узел, ответственный за координацию транзакции

- TransactionStatus: статус транзакции
- PREPARE: стадия подготовки
- COMMITTED: транзакция успешно завершена, но данные ещё не видны
- VISIBLE: транзакция успешна, данные видны
- ABORTED: транзакция не удалась
- LoadJobSourceType: тип задачи импорта
- PrepareTime: время начала транзакции
- PreparedTime: время успешной подготовки транзакции (поддерживается начиная с версии 3.5.4)
- CommitTime: время успешной фиксации транзакции
- FinishTime: время, когда данные стали видны
- Reason: сообщение об ошибке
- ErrorReplicasCount: количество replica с ошибками
- ListenerId: идентификатор связанного задания импорта
- TimeoutMs: тайм-аут для транзакции от состояния `PREPARE` до состояния `PREPARED`, в миллисекундах
- PreparedTimeoutMs: тайм-аут для транзакции от состояния `PREPARED` до состояния `COMMITTED`, в миллисекундах (поддерживается начиная с версии 3.5.4)

Примеры

1. Просмотр транзакции с идентификатором 4005:

```
SHOW TRANSACTION WHERE ID=4005;
```

2. В указанной базе данных просмотр транзакции с идентификатором 4005:

```
SHOW TRANSACTION FROM db WHERE ID=4005;
```

SPARK LOAD

SPARK LOAD предварительно обрабатывает импортируемые данные с помощью внешних ресурсов spark, повышает производительность импорта большого объема данных Selena и экономит вычислительные ресурсы кластера Selena. Он в основном используется в сценарии начальной миграции и импорта больших объемов данных в Selena.

Spark load является асинхронным методом импорта. Пользователям необходимо создавать задачи импорта типа Spark через протокол MySQL и просматривать результаты импорта с помощью `SHOW LOAD`.

> **ПРИМЕЧАНИЕ**
>
> - Вы можете загружать данные в таблицы Selena только как пользователь, имеющий привилегию INSERT для этих таблиц Selena. Если у вас нет привилегии INSERT, следуйте инструкциям, приведенным в GRANT, чтобы предоставить привилегию INSERT пользователю, который используется для подключения к вашему кластеру Selena.
> - При использовании Spark Load для загрузки данных в таблицу Selena, столбец bucketing таблицы Selena не может быть типа DATE, DATETIME или DECIMAL.

Синтаксис

```
LOAD LABEL load_label
(
data_desc1[, data_desc2, ...]
)
WITH RESOURCE resource_name
[resource_properties]
[opt_properties]
```

1.load_label

Метка текущей импортируемой партии. Уникальна в пределах базы данных.

Синтаксис:

```
[database_name.]your_label
```

2.data_desc

Используется для описания партии импортируемых данных.

Синтаксис:

```
DATA INFILE
(
"file_path1"[, file_path2, ...]
)
[NEGATIVE]
INTO TABLE `table_name`
[PARTITION (p1, p2)]
[COLUMNS TERMINATED BY "column_separator"]
[FORMAT AS "file_type"]
[(column_list)]
[COLUMNS FROM PATH AS (col2, ...)]
[SET (k1 = func(k2))]
[WHERE predicate]

DATA FROM TABLE hive_external_tbl
[NEGATIVE]
INTO TABLE tbl_name
[PARTITION (p1, p2)]
[SET (k1=f1(xx), k2=f2(xx))]
[WHERE predicate]
```

Примечание

```
file_path:

Путь к файлу может указывать на один файл, или можно использовать
подстановочный знак * для указания всех файлов в каталоге. Подстановочные
знаки должны соответствовать файлам, а не каталогам.

hive_external_tbl:

Имя внешней таблицы hive.
Требуется, чтобы столбцы в импортируемой таблице selena существовали во
внешней таблице hive.
Каждая задача загрузки поддерживает загрузку только из одной внешней
таблицы Hive.
Не может использоваться одновременно с режимом file_path.

PARTITION:

Если этот параметр указан, будет импортирован только указанный partition, и
данные вне импортируемого partition будут отфильтрованы.
Если не указан, по умолчанию будут импортированы все partition таблицы.

NEGATIVE:

Если этот параметр указан, это эквивалентно загрузке партии "отрицательных"
данных. Используется для компенсации той же партии ранее импортированных
данных.
Этот параметр применим только тогда, когда существует столбец значений и
тип агрегации столбца значений равен только SUM.

column_separator:

Указывает разделитель столбцов в импортируемом файле. По умолчанию \ t
Если это невидимый символ, вам нужно добавить к нему префикс \ \ x и
использовать шестнадцатеричное представление для разделителя.
Например, разделитель файла hive \ x01 указывается как "\ \ x01"

file_type:

Используется для указания типа импортируемого файла. В настоящее время
поддерживаемые типы файлов: csv, orc и parquet.

column_list:

Используется для указания соответствия между столбцами в импортируемом
файле и столбцами в таблице.
```

```
Когда вам нужно пропустить столбец в импортируемом файле, укажите столбец
как имя столбца, которое не существует в таблице.

Синтаксис:
(col_name1, col_name2, ...)

SET:

Если указать этот параметр, вы можете преобразовать столбец исходного файла
в соответствии с функцией, а затем импортировать преобразованные результаты
в таблицу. Синтаксис: column_name = expression.
Поддерживаются только встроенные функции Spark SQL. Пожалуйста, обратитесь
к https://spark.apache.org/docs/2.4.6/api/sql/index.html.
Приведем несколько примеров для понимания.
Пример 1: в таблице есть три столбца "c1, c2, c3", и первые два столбца в
исходном файле соответствуют (c1, c2), а сумма последних двух столбцов
соответствует C3; тогда необходимо указать columns (c1, c2, tmp_c3, tmp_c4)
set (c3 = tmp_c3 + tmp_c4);
Пример 2: в таблице есть три столбца "year, month и day", и в исходном
файле есть только один столбец времени в формате "2018-06-01 01:02:03".
Тогда вы можете указать columns (tmp_time) set (year = year (tmp_time),
month = month (tmp_time), day = day (tmp_time)) для завершения импорта.

WHERE:

Фильтрация преобразованных данных, и только данные, удовлетворяющие условию
where, могут быть импортированы. В операторе WHERE могут ссылаться только
имена столбцов в таблице
```

3.resource_name

Имя используемого ресурса spark можно просмотреть с помощью команды `SHOW RESOURCES`.

4.resource_properties

Когда у вас есть временная потребность, например, изменение конфигураций Spark и HDFS, вы можете установить параметры здесь, которые вступят в силу только в этом конкретном задании загрузки spark и не повлияют на существующие конфигурации в кластере Selena.

5.opt_properties

Используется для указания некоторых специальных параметров.

Синтаксис:

```
[PROPERTIES ("key"="value", ...)]
```

Вы можете указать следующие параметры: timeout: указывает время ожидания операции импорта. Время ожидания по умолчанию составляет 4 часа. В секундах. max_filter_ratio:максимально допустимая доля данных, которая может быть отфильтрована (по причинам, таким как нестандартные данные). По умолчанию нулевая терпимость. strict mode: строго ли ограничивать данные. По умолчанию false. timezone: указывает часовой пояс некоторых функций, зависящих от часового пояса, таких как strftime / alignment_timestamp/from_unixtime и т.д. Пожалуйста, обратитесь к документу [time zone] для получения подробной информации. Если не указано, используется часовой пояс "Asia / Shanghai".

6.Пример формата импортируемых данных

int (TINYINT/SMALLINT/INT/BIGINT/LARGEINT): 1, 1000, 1234 float (FLOAT/DOUBLE/DECIMAL): 1.1, 0.23, .356 date (DATE/DATETIME) :2017-10-03, 2017-06-13 12:34:03. (Примечание: для других форматов дат вы можете использовать функцию strftime или time_format для преобразования в команде Import) string class (CHAR/VARCHAR): "I am a student", "a"

NULL значение: \ N

Примеры

1. Импортировать партию данных из HDFS, указав время ожидания и коэффициент фильтрации. Использовать имя my_spark resources для spark.

```
LOAD LABEL example_db.label1
(
DATA INFILE("hdfs://hdfs_host:hdfs_port/user/selena/data/input/file")
INTO TABLE `my_table`
)
WITH RESOURCE 'my_spark'
PROPERTIES
(
    "timeout" = "3600",
    "max_filter_ratio" = "0.1"
);
```

Где hdfs_host — это хост namenode, hdfs_port — это порт fs.defaultfs (по умолчанию 9000)

2. Импортировать партию "отрицательных" данных из HDFS, указать разделитель как запятую, использовать подстановочный знак * для указания всех файлов в каталоге и указать временные параметры ресурсов spark.

```
LOAD LABEL example_db.label3
(
DATA INFILE("hdfs://hdfs_host:hdfs_port/user/selena/data/input/*")
NEGATIVE
INTO TABLE `my_table`
COLUMNS TERMINATED BY ","
)
WITH RESOURCE 'my_spark'
(
    "spark.executor.memory" = "3g",
    "broker.username" = "hdfs_user",
    "broker.password" = "hdfs_passwd"
);
```

3. Импортировать партию данных из HDFS, указать partition и выполнить некоторое преобразование столбцов импортируемого файла, как показано ниже:

```
Структура таблицы:
k1 varchar(20)
k2 int

Предположим, что файл данных имеет только одну строку данных:

Adele,1,1

Каждый столбец в файле данных соответствует каждому столбцу, указанному
в операторе импорта:
k1,tmp_k2,tmp_k3

Преобразование выглядит следующим образом:

1. k1: без преобразования
2. k2: сумма tmp_k2 и tmp_k3

LOAD LABEL example_db.label6
(
DATA INFILE("hdfs://hdfs_host:hdfs_port/user/selena/data/input/file")
INTO TABLE `my_table`
PARTITION (p1, p2)
COLUMNS TERMINATED BY ","
(k1, tmp_k2, tmp_k3)
SET (
k2 = tmp_k2 + tmp_k3
)
)
WITH RESOURCE 'my_spark';
```

4. Извлечь поле partition из пути к файлу

   При необходимости разделенные поля в пути к файлу будут разрешены в соответствии с типами полей, определенными в таблице, аналогично функции Partition Discovery в Spark

```
LOAD LABEL example_db.label10
(
DATA
INFILE("hdfs://hdfs_host:hdfs_port/user/selena/data/input/dir/city=beijing
INTO TABLE `my_table`
(k1, k2, k3)
COLUMNS FROM PATH AS (city, utc_date)
SET (uniq_id = md5sum(k1, city))
)
WITH RESOURCE 'my_spark';
```

Каталог `hdfs://hdfs_host:hdfs_port/user/selena/data/input/dir/city=beijing` включает следующие файлы:

`[hdfs://hdfs_host:hdfs_port/user/selena/data/input/dir/city=beijing/utc_date=2019-06-26/0000.csv, hdfs://hdfs_host:hdfs_port/user/selena/data/input/dir/city=beijing/utc_date=2019-06-26/0001.csv, ...]`

Поля city и utc_date извлекаются из пути к файлу

5. Отфильтровать импортируемые данные. Могут быть импортированы только столбцы со значением k1 больше 10.

```
LOAD LABEL example_db.label10
(
DATA INFILE("hdfs://hdfs_host:hdfs_port/user/selena/data/input/file")
INTO TABLE `my_table`
WHERE k1 > 10
)
WITH RESOURCE 'my_spark';
```

6. Импортировать из внешней таблицы hive и преобразовать столбец uuid в исходной таблице в тип bitmap с помощью глобального словаря.

```
LOAD LABEL db1.label1
(
DATA FROM TABLE hive_t1
INTO TABLE tbl1
SET
(
uuid=bitmap_dict(uuid)
)
)
WITH RESOURCE 'my_spark';
```

STREAM LOAD

Selena предоставляет метод загрузки на основе HTTP — STREAM LOAD, который помогает загружать данные из локальной файловой системы или из источника потоковых данных. После отправки загрузочного задания Selena синхронно выполняет задание и возвращает результат задания после его завершения. Вы можете определить, было ли задание успешным, на основе результата задания. Для получения информации о сценариях применения, ограничениях, принципах и поддерживаемых форматах файлов данных Stream Load см. Loading from a local file system via Stream Load.

> **ПОДСКАЗКА**
>
> Для быстрого старта с Selena см. раздел Быстрый старт.

Начиная с версии 3.2.7, Stream Load поддерживает сжатие данных JSON во время передачи, что снижает нагрузку на сетевую пропускную способность. Пользователи могут указать различные алгоритмы сжатия, используя параметры `compression` и `Content-Encoding`. Поддерживаемые алгоритмы сжатия включают GZIP, BZIP2, LZ4_FRAME и ZSTD. Для получения дополнительной информации см. data_desc.

Начиная с версии 3.4.0, система поддерживает объединение нескольких запросов Stream Load. Для получения дополнительной информации см. Параметры Merge Commit.

> **ПРИМЕЧАНИЕ**
>
> - После загрузки данных в таблицу Selena с помощью Stream Load данные материализованных представлений, созданных для этой таблицы, также обновляются.

> - Вы можете загружать данные в таблицы Selena только как пользователь, имеющий привилегию INSERT для этих таблиц Selena. Если у вас нет привилегии INSERT, следуйте инструкциям, приведенным в GRANT, чтобы предоставить привилегию INSERT пользователю, который используется для подключения к вашему кластеру Selena.

Синтаксис

```
curl --location-trusted -u <username>:<password> -XPUT <url>
(
    data_desc
)
[opt_properties]
```

В этом разделе используется curl в качестве примера для описания того, как загружать данные с помощью Stream Load. В дополнение к curl вы также можете использовать другие инструменты или языки, совместимые с HTTP, для выполнения Stream Load. Параметры, связанные с загрузкой, включены в поля заголовка HTTP-запроса. При вводе этих параметров обратите внимание на следующие моменты:

- Вы можете использовать chunked transfer encoding, как показано в этом разделе. Если вы не выбираете chunked transfer encoding, вы должны ввести поле заголовка `Content-Length`, чтобы указать длину содержимого, которое необходимо передать, тем самым обеспечивая целостность данных.

  > **ПРИМЕЧАНИЕ**
  >
  > Если вы используете curl для выполнения Stream Load, Selena автоматически добавляет поле заголовка `Content-Length`, и вам не нужно вводить его вручную.

- Вы должны добавить поле заголовка `Expect` и указать его значение как `100-continue`, как в `"Expect:100-continue"`. Это помогает предотвратить ненужные передачи данных и снизить потребление ресурсов в случае отклонения вашего запроса на задание.

Обратите внимание, что в Selena некоторые литералы используются как зарезервированные ключевые слова языком SQL. Не используйте эти ключевые слова напрямую в операторах SQL. Если вы хотите использовать такое ключевое слово в операторе SQL, заключите его в пару обратных кавычек (`). См. Keywords.

Параметры

username и password

Укажите имя пользователя и пароль учетной записи, которую вы используете для подключения к вашему кластеру Selena. Это обязательный параметр. Если вы используете учетную запись, для которой не установлен пароль, вам нужно ввести только `<username>:`.

XPUT

Указывает метод HTTP-запроса. Это обязательный параметр. Stream Load поддерживает только метод PUT.

url

Указывает URL-адрес таблицы Selena. Синтаксис:

```
http://<fe_host>:
<fe_http_port>/api/<database_name>/<table_name>/_stream_load
```

В следующей таблице описываются параметры в URL.

Параметр	Обязательный	Описание
fe_host	Да	IP-адрес узла FE в вашем кластере Selena. **ПРИМЕЧАНИЕ** Если вы отправляете загрузочное задание на конкретный узел BE или CN, вы должны ввести IP-адрес узла BE или CN.
fe_http_port	Да	Номер порта HTTP узла FE в вашем кластере Selena. Номер порта по умолчанию — `8030`. **ПРИМЕЧАНИЕ** Если вы отправляете загрузочное задание на конкретный узел BE или CN, вы должны ввести номер порта HTTP узла BE или CN. Номер порта по умолчанию — `8030`.
database_name	Да	Имя базы данных, к которой принадлежит таблица Selena.
table_name	Да	Имя таблицы Selena.

> **ПРИМЕЧАНИЕ**

> Вы можете использовать SHOW FRONTENDS для просмотра IP-адреса и порта HTTP узла FE.

data_desc

Описывает файл данных, который вы хотите загрузить. Дескриптор `data_desc` может включать имя файла данных, формат, разделитель столбцов, разделитель строк, целевые partition и сопоставление столбцов с таблицей Selena. Синтаксис:

```
-T <file_path>
-H "format: CSV | JSON"
-H "column_separator: <column_separator>"
-H "row_delimiter: <row_delimiter>"
-H "columns: <column1_name>[, <column2_name>, ... ]"
-H "partitions: <partition1_name>[, <partition2_name>, ...]"
-H "temporary_partitions: <temporary_partition1_name>[,
<temporary_partition2_name>, ...]"
-H "jsonpaths: [ \"<json_path1>\"[, \"<json_path2>\", ...] ]"
-H "strip_outer_array: true | false"
-H "json_root: <json_path>"
-H "ignore_json_size: true | false"
-H "compression: <compression_algorithm> | Content-Encoding:
<compression_algorithm>"
```

Параметры в дескрипторе `data_desc` можно разделить на три типа: общие параметры, параметры CSV и параметры JSON.

Общие параметры

Параметр	Обязательный	Описание
file_path	Да	Путь сохранения файла данных. При желании вы можете включить расширение имени файла.
format	Нет	Формат файла данных. Допустимые значения: `CSV` и `JSON`. Значение по умолчанию: `CSV`.
partitions	Нет	Partition, в которые вы хотите загрузить файл данных. По умолчанию, если вы не укажете этот параметр, Selena загружает файл данных во все partition таблицы Selena.
tempora	Нет	Имя временного partition, в который вы хотите загрузить файл данных. Вы можете указать несколько временных partition, которые должны быть разделены запятыми (,).

Параметр	Обязательный	Описание
ry_partitions		
columns	Нет	Сопоставление столбцов между файлом данных и таблицей Selena. Если поля в файле данных могут быть последовательно сопоставлены со столбцами в таблице Selena, вам не нужно указывать этот параметр. Вместо этого вы можете использовать этот параметр для выполнения преобразований данных. Например, если вы загружаете файл данных CSV, и файл состоит из двух столбцов, которые могут быть последовательно сопоставлены с двумя столбцами `id` и `city` таблицы Selena, вы можете указать `"columns: city,tmp_id, id = tmp_id * 100"`. Для получения дополнительной информации см. раздел "Сопоставление столбцов" в этом разделе.

Параметры CSV

Параметр	Обязательный	Описание
column_separator	Нет	Символы, которые используются в файле данных для разделения полей. Если вы не укажете этот параметр, этот параметр по умолчанию равен `\t`, что указывает на табуляцию. Убедитесь, что разделитель столбцов, который вы указываете с помощью этого параметра, совпадает с разделителем столбцов, используемым в файле данных. **ПРИМЕЧАНИЕ** - Для данных CSV вы можете использовать строку UTF-8, такую как запятая (,), табуляция или вертикальная черта (|), длина которой не превышает 50 байт, в качестве текстового разделителя. - Если файл данных использует последовательные непечатаемые символы (например, `\r\n`) в качестве разделителя столбцов, вы должны установить этот параметр как `\\x0D0A`.
row_delimiter	Нет	Символы, которые используются в файле данных для разделения строк. Если вы не укажете этот параметр, этот параметр по умолчанию равен `\n`. **ПРИМЕЧАНИЕ** Если файл данных использует последовательные непечатаемые символы (например, `\r\n`) в качестве разделителя строк, вы должны установить этот параметр как `\\x0D0A`.
skip_header	Нет	Указывает, следует ли пропускать первые несколько строк файла данных, когда файл данных в формате CSV. Тип: INTEGER. Значение по умолчанию: `0`. В некоторых файлах данных в формате CSV первые несколько строк в начале используются для определения метаданных, таких как имена столбцов и типы данных столбцов. Установив параметр `skip_header`, вы можете включить Selena для пропуска первых нескольких строк файла данных во время загрузки данных. Например, если вы установите этот параметр в `1`, Selena пропустит

Параметр	Обязательный	Описание
		первую строку файла данных во время загрузки данных. Первые несколько строк в начале в файле данных должны быть разделены с помощью разделителя строк, который вы указываете в команде загрузки.

trim_space	Нет	Указывает, следует ли удалять пробелы перед и после разделителей столбцов из файла данных, когда файл данных в формате CSV. Тип: BOOLEAN. Значение по умолчанию: `false`.
		Для некоторых баз данных пробелы добавляются к разделителям столбцов при экспорте данных в виде файла данных в формате CSV. Такие пробелы называются начальными или конечными пробелами в зависимости от их расположения. Установив параметр `trim_space`, вы можете включить Selena для удаления таких ненужных пробелов во время загрузки данных.
		Обратите внимание, что Selena не удаляет пробелы (включая начальные и конечные пробелы) внутри поля, заключенного в пару символов, указанных в `enclose`. Например, следующие значения полей используют вертикальную черту (`|`) в качестве разделителя столбцов и двойные кавычки (`"`) в качестве символа, указанного в `enclose`:
		`|"Love Selena"|`
		`|" Love Selena "|`
		`| "Love Selena" |`
		Если вы установите `trim_space` в `true`, Selena обрабатывает предыдущие значения полей следующим образом:
		`|"Love Selena"|`
		`|" Love Selena "|`
		`|"Love Selena"|`

enclose	Нет	Указывает символ, который используется для обертывания значений полей в файле данных в соответствии с RFC4180, когда файл данных в формате CSV. Тип: однобайтовый символ. Значение по умолчанию: `NONE`. Наиболее распространенными символами являются одинарная кавычка (`'`) и двойная кавычка (`"`). Все специальные символы (включая разделители строк и разделители столбцов), обернутые с помощью символа, указанного в `enclose`, считаются обычными символами. Selena может сделать больше, чем RFC4180, поскольку позволяет вам указать любой однобайтовый символ в качестве символа, указанного в `enclose`. Если значение поля содержит символ, указанный в `enclose`, вы можете использовать тот же символ для экранирования этого символа, указанного в `enclose`. Например, вы устанавливаете `enclose` в `"`, и значение поля — `a "quoted" c`. В этом случае вы можете ввести значение поля как `"a ""quoted"" c"` в файл данных.
escape	Нет	Указывает символ, который используется для экранирования различных специальных символов, таких как разделители строк, разделители столбцов, символы экранирования и символы, указанные в `enclose`, которые затем считаются Selena обычными символами и анализируются как часть значений полей, в которых они находятся. Тип: однобайтовый символ. Значение по умолчанию: `NONE`. Наиболее распространенным символом является косая черта (`\`), которая должна быть записана как двойные косые черты (`\\`) в операторах SQL.

Параметр	Обязательный	Описание
		ПРИМЕЧАНИЕ Символ, указанный в `escape`, применяется как внутри, так и вне каждой пары символов, указанных в `enclose`. Два примера следующие: • Когда вы устанавливаете `enclose` в `"` и `escape` в `\`, Selena анализирует `"say \"Hello world\""` в `say "Hello world"`. • Предположим, что разделитель столбцов — запятая (`,`). Когда вы устанавливаете `escape` в `\`, Selena анализирует `a, b\, c` в два отдельных значения полей: `a` и `b, c`.

ПРИМЕЧАНИЕ

- Для данных CSV вы можете использовать строку UTF-8, такую как запятая (,), табуляция или вертикальная черта (|), длина которой не превышает 50 байт, в качестве текстового разделителя.
- Значения NULL обозначаются с помощью `\N`. Например, файл данных состоит из трех столбцов, и запись из этого файла данных содержит данные в первом и третьем столбцах, но не содержит данных во втором столбце. В этой ситуации вам нужно использовать `\N` во втором столбце для обозначения значения null. Это означает, что запись должна быть скомпилирована как `a,\N,b` вместо `a,,b`. `a,,b` обозначает, что второй столбец записи содержит пустую строку.
- Параметры формата, включая `skip_header`, `trim_space`, `enclose` и `escape`, поддерживаются в версии 3.0 и более поздних версиях.

Параметры JSON

Параметр	Обязательный	Описание
jsonpaths	Нет	Имена ключей, которые вы хотите загрузить из файла данных JSON. Вам нужно указать этот параметр только при загрузке данных JSON с использованием режима сопоставления. Значение этого параметра в формате JSON. См. Настройка сопоставления столбцов для загрузки данных JSON.
strip_outer_a	Нет	Указывает, следует ли удалять самую внешнюю структуру массива. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`.

Параметр	Обязательный	Описание
rray		В реальных бизнес-сценариях данные JSON могут иметь самую внешнюю структуру массива, обозначенную парой квадратных скобок `[]`. В этой ситуации мы рекомендуем установить этот параметр в `true`, чтобы Selena удаляла самые внешние квадратные скобки `[]` и загружала каждый внутренний массив как отдельную запись данных. Если вы установите этот параметр в `false`, Selena анализирует весь файл данных JSON в один массив и загружает массив как одну запись данных. Например, данные JSON — `[ {"category" : 1, "author" : 2}, {"category" : 3, "author" : 4} ]`. Если вы установите этот параметр в `true`, `{"category" : 1, "author" : 2}` и `{"category" : 3, "author" : 4}` анализируются в отдельные записи данных, которые загружаются в отдельные строки таблицы Selena.
json_root	Нет	Корневой элемент данных JSON, которые вы хотите загрузить из файла данных JSON. Вам нужно указать этот параметр только при загрузке данных JSON с использованием режима сопоставления. Значение этого параметра — действительная строка JsonPath. По умолчанию значение этого параметра пусто, что указывает на то, что все данные из файла данных JSON будут загружены. Для получения дополнительной информации см. раздел "Загрузка данных JSON с использованием режима сопоставления с указанным корневым элементом" этого раздела.
ignore_json_size	Нет	Указывает, следует ли проверять размер тела JSON в HTTP-запросе. **ПРИМЕЧАНИЕ** По умолчанию размер тела JSON в HTTP-запросе не может превышать 100 МБ. Если тело JSON превышает 100 МБ в размере, возникает ошибка "The size of this batch exceed the max size [104857600] of json type data data [8617627793]. Set ignore_json_size to skip check, although it may lead huge memory consuming.". Чтобы предотвратить эту ошибку, вы можете добавить `"ignore_json_size:true"` в заголовок HTTP-запроса, чтобы указать Selena не проверять размер тела JSON.
compression, Content-Encoding	НЕТ	Алгоритм кодирования, который применяется к данным во время передачи. Поддерживаемые алгоритмы включают GZIP, BZIP2, LZ4_FRAME и ZSTD. Пример: `curl --location-trusted -u root: -v 'http://127.0.0.1:18030/api/db0/tbl_simple/_stream_load' \-X PUT  -H "expect:100-continue" \-H 'format: json' -H 'compression: lz4_frame'   -T ./b.json.lz4`.

При загрузке данных JSON также обратите внимание, что размер каждого объекта JSON не может превышать 4 ГБ. Если отдельный объект JSON в файле данных JSON превышает 4 ГБ в размере, возникает ошибка "This parser can't support a document that big.".

Параметры Merge Commit

Включает Merge Commit для нескольких параллельных запросов Stream Load в пределах указанного временного окна и объединяет их в одну транзакцию.

> **WARNING**
>
> Обратите внимание, что оптимизация Merge Commit подходит для сценария с **параллельными** заданиями Stream Load на одной таблице. Она не рекомендуется, если параллелизм равен единице. В то же время дважды подумайте, прежде чем устанавливать `merge_commit_async` в `false` и `merge_commit_interval_ms` в большое значение, поскольку они могут привести к снижению производительности загрузки.

Параметр	Обязательный	Описание
enable_merge_commit	Нет	Включить ли Merge Commit для запроса на загрузку. Допустимые значения: `true` и `false` (по умолчанию).
merge_commit_async	Нет	Режим возврата сервера. Допустимые значения: • `true`: Включает асинхронный режим, где сервер возвращается немедленно после получения данных. Этот режим не гарантирует успешную загрузку. • `false`(по умолчанию): Включает синхронный режим, где сервер возвращается только после фиксации объединенной транзакции, гарантируя успешную загрузку и видимость.
merge_commit_interval_ms	Да	Размер окна слияния времени. Единица измерения: миллисекунды. Merge Commit пытается объединить запросы на загрузку, полученные в этом окне, в одну транзакцию. Большее окно повышает эффективность слияния, но увеличивает задержку.
merge_commit_parallel	Да	Степень параллелизма для плана загрузки, созданного для каждого окна слияния. Параллелизм можно настроить на основе нагрузки на прием. Увеличьте это значение, если много запросов и/или большое количество данных для загрузки. Параллелизм ограничен количеством узлов BE, рассчитывается как `min(merge_commit_parallel, количество узлов BE)`.

ПРИМЕЧАНИЕ

- Merge Commit поддерживает объединение только **однородных** запросов на загрузку в одну базу данных и таблицу. "Однородные" означает, что параметры Stream Load идентичны, включая: общие параметры, параметры формата JSON, параметры формата CSV, `opt_properties` и параметры Merge Commit.
- Для загрузки данных в формате CSV вы должны убедиться, что каждая строка заканчивается разделителем строк. `skip_header` не поддерживается.
- Сервер автоматически генерирует метки для транзакций. Они будут проигнорированы, если указаны.
- Merge Commit объединяет несколько запросов на загрузку в одну транзакцию. Если один запрос содержит проблемы с качеством данных, все запросы в транзакции завершатся неудачей.

opt_properties

Указывает некоторые необязательные параметры, которые применяются ко всему загрузочному заданию. Синтаксис:

```
-H "label: <label_name>"
-H "where: <condition1>[, <condition2>, ...]"
-H "max_filter_ratio: <num>"
-H "timeout: <num>"
-H "strict_mode: true | false"
-H "timezone: <string>"
-H "load_mem_limit: <num>"
-H "partial_update: true | false"
-H "partial_update_mode: row | column"
-H "merge_condition: <column_name>"
```

В следующей таблице описываются необязательные параметры.

Параметр	Обязательный	Описание
label	Нет	Метка загрузочного задания. Если вы не укажете этот параметр, Selena автоматически сгенерирует метку для загрузочного задания. Selena не позволяет использовать одну метку для многократной загрузки партии данных. Таким

Параметр	Обязательный	Описание
		образом, Selena предотвращает повторную загрузку одних и тех же данных. Для правил именования меток см. System limits. По умолчанию Selena сохраняет метки загрузочных заданий, которые были успешно завершены за последние три дня. Вы можете использовать параметр FE `label_keep_max_second` для изменения периода хранения меток.
where	Нет	Условия, на основе которых Selena фильтрует предварительно обработанные данные. Selena загружает только предварительно обработанные данные, которые соответствуют условиям фильтра, указанным в предложении WHERE.
max_filter_ratio	Нет	Максимальная ошибочная терпимость загрузочного задания. Ошибочная терпимость — это максимальный процент записей данных, которые могут быть отфильтрованы из-за неадекватного качества данных во всех записях данных, запрошенных загрузочным заданием. Допустимые значения: от `0` до `1`. Значение по умолчанию: `0`. Мы рекомендуем сохранить значение по умолчанию `0`. Таким образом, если обнаружены неквалифицированные записи данных, загрузочное задание завершается неудачей, тем самым обеспечивая правильность данных. Если вы хотите игнорировать неквалифицированные записи данных, вы можете установить этот параметр в значение больше `0`. Таким образом, загрузочное задание может быть успешным, даже если файл данных содержит неквалифицированные записи данных. **ПРИМЕЧАНИЕ** Неквалифицированные записи данных не включают записи данных, которые фильтруются предложением WHERE.
log_rejected_record_num	Нет	Указывает максимальное количество неквалифицированных строк данных, которые могут быть зарегистрированы. Этот параметр поддерживается начиная с версии 3.1. Допустимые значения: `0`, `-1` и любое ненулевое положительное целое число. Значение по умолчанию: `0`. • Значение `0` указывает, что строки данных, которые фильтруются, не будут зарегистрированы. • Значение `-1` указывает, что все строки данных, которые фильтруются, будут зарегистрированы. • Ненулевое положительное целое число, такое как `n`, указывает, что до `n` строк данных, которые фильтруются, могут быть зарегистрированы на каждом BE или CN.
timeout	Нет	Период времени ожидания загрузочного задания. Допустимые значения: от `1` до `259200`. Единица измерения: секунда. Значение по умолчанию: `600`. **ПРИМЕЧАНИЕ** В дополнение к параметру `timeout` вы также можете использовать параметр FE `stream_load_default_timeout_second` для централизованного управления периодом времени ожидания для всех заданий Stream Load в вашем кластере Selena. Если вы укажете параметр `timeout`, период времени ожидания, указанный параметром `timeout`, будет иметь приоритет. Если вы не укажете параметр `timeout`, период времени ожидания, указанный параметром `stream_load_default_timeout_second`, будет иметь приоритет.

Параметр	Обязательный	Описание
strict_mode	Нет	Указывает, следует ли включить строгий режим. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`. Значение `true` указывает на включение строгого режима, а значение `false` указывает на отключение строгого режима.
timezone	Нет	Часовой пояс, используемый загрузочным заданием. Значение по умолчанию: `Asia/Shanghai`. Значение этого параметра влияет на результаты, возвращаемые функциями, такими как strftime, alignment_timestamp и from_unixtime. Часовой пояс, указанный этим параметром, является часовым поясом уровня сеанса. Для получения дополнительной информации см. Configure a time zone.
load_mem_limit	Нет	Максимальный объем памяти, который может быть выделен для загрузочного задания. Единица измерения: байты. По умолчанию максимальный размер памяти для загрузочного задания составляет 2 ГБ. Значение этого параметра не может превышать максимальный объем памяти, который может быть выделен для каждого BE или CN.
partial_update	Нет	Использовать ли частичные обновления. Допустимые значения: `TRUE` и `FALSE`. Значение по умолчанию: `FALSE`, что указывает на отключение этой функции.
partial_update_mode	Нет	Указывает режим частичных обновлений. Допустимые значения: `row` и `column`. • Значение `row` (по умолчанию) означает частичные обновления в режиме строк, что более подходит для обновлений в реальном времени с множеством столбцов и небольшими партиями. • Значение `column` означает частичные обновления в режиме столбцов, что более подходит для пакетных обновлений с небольшим количеством столбцов и множеством строк. В таких сценариях включение режима столбцов обеспечивает более быструю скорость обновления. Например, в таблице со 100 столбцами, если обновляется только 10 столбцов (10% от общего числа) для всех строк, скорость обновления в режиме столбцов в 10 раз быстрее.
merge_condition	Нет	Указывает имя столбца, который вы хотите использовать в качестве условия для определения того, могут ли обновления вступить в силу. Обновление из исходной записи в целевую запись вступает в силу только тогда, когда исходная запись данных имеет большее или равное значение, чем целевая запись данных в указанном столбце. Selena поддерживает условные обновления начиная с версии 2.5. **ПРИМЕЧАНИЕ** Указанный вами столбец не может быть столбцом первичного ключа. Кроме того, только таблицы, использующие таблицу первичного ключа, поддерживают условные обновления.

Сопоставление столбцов

Настройка сопоставления столбцов для загрузки данных CSV

Если столбцы файла данных могут быть последовательно сопоставлены один к одному со столбцами таблицы Selena, вам не нужно настраивать сопоставление столбцов между файлом данных и таблицей Selena.

Если столбцы файла данных не могут быть последовательно сопоставлены один к одному со столбцами таблицы Selena, вам нужно использовать параметр `columns` для настройки сопоставления столбцов между файлом данных и таблицей Selena. Это включает следующие два случая использования:

- **Одинаковое количество столбцов, но разная последовательность столбцов. Кроме того, данные из файла данных не нужно вычислять с помощью функций перед загрузкой в соответствующие столбцы таблицы Selena.**

  В параметре `columns` вам нужно указать имена столбцов таблицы Selena в той же последовательности, в которой расположены столбцы файла данных.

  Например, таблица Selena состоит из трех столбцов, которые по порядку `col1`, `col2` и `col3`, и файл данных также состоит из трех столбцов, которые могут быть сопоставлены со столбцами таблицы Selena `col3`, `col2` и `col1` по порядку. В этом случае вам нужно указать `"columns: col3, col2, col1"`.

- **Разное количество столбцов и разная последовательность столбцов. Кроме того, данные из файла данных необходимо вычислить с помощью функций перед загрузкой в соответствующие столбцы таблицы Selena.**

  В параметре `columns` вам нужно указать имена столбцов таблицы Selena в той же последовательности, в которой расположены столбцы файла данных, и указать функции, которые вы хотите использовать для вычисления данных. Два примера следующие:

  - Таблица Selena состоит из трех столбцов, которые по порядку `col1`, `col2` и `col3`. Файл данных состоит из четырех столбцов, среди которых первые три столбца могут быть последовательно сопоставлены со столбцами таблицы Selena `col1`, `col2` и `col3`, а четвертый столбец не может быть сопоставлен ни с одним из столбцов таблицы Selena. В этом случае вам нужно временно указать

имя для четвертого столбца файла данных, и временное имя должно отличаться от любого из имен столбцов таблицы Selena. Например, вы можете указать `"columns: col1, col2, col3, temp"`, в котором четвертый столбец файла данных временно назван `temp`.

- Таблица Selena состоит из трех столбцов, которые по порядку `year`, `month` и `day`. Файл данных состоит только из одного столбца, который содержит значения даты и времени в формате `yyyy-mm-dd hh:mm:ss`. В этом случае вы можете указать `"columns: col, year = year(col), month=month(col), day=day(col)"`, в котором `col` — временное имя столбца файла данных, а функции `year = year(col)`, `month=month(col)` и `day=day(col)` используются для извлечения данных из столбца файла данных `col` и загрузки данных в сопоставляемые столбцы таблицы Selena. Например, `year = year(col)` используется для извлечения данных `yyyy` из столбца файла данных `col` и загрузки данных в столбец таблицы Selena `year`.

Для получения подробных примеров см. Настройка сопоставления столбцов.

Настройка сопоставления столбцов для загрузки данных JSON

Если ключи документа JSON имеют те же имена, что и столбцы таблицы Selena, вы можете загрузить данные в формате JSON, используя простой режим. В простом режиме вам не нужно указывать параметр `jsonpaths`. Этот режим требует, чтобы данные в формате JSON были объектом, обозначенным фигурными скобками `{}`, таким как `{"category": 1, "author": 2, "price": "3"}`. В этом примере `category`, `author` и `price` — это имена ключей, и эти ключи могут быть сопоставлены один к одному по имени со столбцами `category`, `author` и `price` таблицы Selena.

Если ключи документа JSON имеют разные имена, чем столбцы таблицы Selena, вы можете загрузить данные в формате JSON, используя режим сопоставления. В режиме сопоставления вам нужно использовать параметры `jsonpaths` и `COLUMNS` для указания сопоставления столбцов между документом JSON и таблицей Selena:

- В параметре `jsonpaths` укажите ключи JSON в последовательности, в которой они расположены в документе JSON.
- В параметре `COLUMNS` укажите сопоставление между ключами JSON и столбцами таблицы Selena:
  - Имена столбцов, указанные в параметре `COLUMNS`, сопоставляются один к одному в последовательности с ключами JSON.

- Имена столбцов, указанные в параметре `COLUMNS`, сопоставляются один к одному по имени со столбцами таблицы Selena.

Для примеров загрузки данных в формате JSON с использованием режима сопоставления см. Загрузка данных JSON с использованием режима сопоставления.

Возвращаемое значение

После завершения загрузочного задания Selena возвращает результат задания в формате JSON. Пример:

```
{
    "TxnId": 1003,
    "Label": "label123",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 1000000,
    "NumberLoadedRows": 999999,
    "NumberFilteredRows": 1,
    "NumberUnselectedRows": 0,
    "LoadBytes": 40888898,
    "LoadTimeMs": 2144,
    "BeginTxnTimeMs": 0,
    "StreamLoadPlanTimeMs": 1,
    "ReadDataTimeMs": 0,
    "WriteDataTimeMs": 11,
    "CommitAndPublishTimeMs": 16,
}
```

В следующей таблице описываются параметры в возвращенном результате задания.

Параметр	Описание
TxnId	ID транзакции загрузочного задания.
Label	Метка загрузочного задания.
Status	Конечный статус загруженных данных. • `Success`: Данные успешно загружены и могут быть запрошены. • `Publish Timeout`: Загрузочное задание успешно отправлено, но данные все еще не могут быть запрошены. Вам не нужно повторять попытку загрузки данных. • `Label Already Exists`: Метка загрузочного задания уже использовалась для другого загрузочного задания. Данные могли быть успешно загружены или загружаются. • `Fail`: Не удалось загрузить данные. Вы можете повторить попытку загрузочного задания.

Параметр	Описание
Message	Статус загрузочного задания. Если загрузочное задание завершается с ошибкой, возвращается подробная причина сбоя.
NumberTotalRows	Общее количество записей данных, которые читаются.
NumberLoadedRows	Общее количество записей данных, которые успешно загружены. Этот параметр действителен только тогда, когда значение, возвращенное для `Status`, равно `Success`.
NumberFilteredRows	Количество записей данных, которые фильтруются из-за неадекватного качества данных.
NumberUnselectedRows	Количество записей данных, которые фильтруются предложением WHERE.
LoadBytes	Объем загруженных данных. Единица измерения: байты.
LoadTimeMs	Время, затраченное на загрузочное задание. Единица измерения: мс.
BeginTxnTimeMs	Время, затраченное на выполнение транзакции для загрузочного задания.
StreamLoadPlanTimeMs	Время, затраченное на создание плана выполнения для загрузочного задания.
ReadDataTimeMs	Время, затраченное на чтение данных для загрузочного задания.
WriteDataTimeMs	Время, затраченное на запись данных для загрузочного задания.
CommitAndPublishTimeMs	Время, затраченное на фиксацию и публикацию данных для загрузочного задания.

Если загрузочное задание завершается с ошибкой, Selena также возвращает `ErrorURL`. Пример:

```
{"ErrorURL": "http://172.26.195.68:8045/api/_load_error_log?
file=error_log_3a4eb8421f0878a6_9a54df29fd9206be"}
```

`ErrorURL` предоставляет URL-адрес, из которого вы можете получить подробную информацию о неквалифицированных записях данных, которые были отфильтрованы. Вы можете указать максимальное количество неквалифицированных строк данных, которые могут быть зарегистрированы, используя необязательный параметр `log_rejected_record_num`, который устанавливается при отправке загрузочного задания.

Вы можете выполнить `curl "url"`, чтобы напрямую просмотреть подробную информацию об отфильтрованных неквалифицированных записях данных. Вы также

можете выполнить `wget "url"`, чтобы экспортировать подробную информацию об этих записях данных:

```
wget http://172.26.195.68:8045/api/_load_error_log?
file=error_log_3a4eb8421f0878a6_9a54df29fd9206be
```

Экспортированные подробные сведения о записях данных сохраняются в локальный файл с именем, похожим на `_load_error_log?file=error_log_3a4eb8421f0878a6_9a54df29fd9206be`. Вы можете использовать команду `cat` для просмотра файла.

Затем вы можете настроить конфигурацию загрузочного задания и снова отправить загрузочное задание.

Примеры

Загрузка данных CSV

В этом разделе данные CSV используются в качестве примера для описания того, как вы можете использовать различные настройки параметров и комбинации для удовлетворения ваших разнообразных требований к загрузке.

Установка периода времени ожидания

Ваша база данных Selena `test_db` содержит таблицу с именем `table1`. Таблица состоит из трех столбцов, которые по порядку `col1`, `col2` и `col3`.

Ваш файл данных `example1.csv` также состоит из трех столбцов, которые могут быть последовательно сопоставлены с `col1`, `col2` и `col3` таблицы `table1`.

Если вы хотите загрузить все данные из `example1.csv` в `table1` в течение до 100 секунд, выполните следующую команду:

```
curl --location-trusted -u <username>:<password> -H "label:label1" \
    -H "Expect:100-continue" \
    -H "timeout:100" \
    -H "max_filter_ratio:0.2" \
    -T example1.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table1/_stream_load
```

Установка ошибочной терпимости

Ваша база данных Selena `test_db` содержит таблицу с именем `table2`. Таблица состоит из трех столбцов, которые по порядку `col1`, `col2` и `col3`.

Ваш файл данных `example2.csv` также состоит из трех столбцов, которые могут быть последовательно сопоставлены с `col1`, `col2` и `col3` таблицы `table2`.

Если вы хотите загрузить все данные из `example2.csv` в `table2` с максимальной ошибочной терпимостью `0.2`, выполните следующую команду:

```
curl --location-trusted -u <username>:<password> -H "label:label2" \
    -H "Expect:100-continue" \
    -H "max_filter_ratio:0.2" \
    -T example2.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table2/_stream_load
```

Настройка сопоставления столбцов

Ваша база данных Selena `test_db` содержит таблицу с именем `table3`. Таблица состоит из трех столбцов, которые по порядку `col1`, `col2` и `col3`.

Ваш файл данных `example3.csv` также состоит из трех столбцов, которые могут быть последовательно сопоставлены с `col2`, `col1` и `col3` таблицы `table3`.

Если вы хотите загрузить все данные из `example3.csv` в `table3`, выполните следующую команду:

```
curl --location-trusted -u <username>:<password>  -H "label:label3" \
    -H "Expect:100-continue" \
    -H "columns: col2, col1, col3" \
    -T example3.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table3/_stream_load
```

> **ПРИМЕЧАНИЕ**
>
> В предыдущем примере столбцы `example3.csv` не могут быть сопоставлены со столбцами `table3` в той же последовательности, в которой эти столбцы

> расположены в `table3`. Поэтому вам нужно использовать параметр `columns` для настройки сопоставления столбцов между `example3.csv` и `table3`.

Установка условий фильтра

Ваша база данных Selena `test_db` содержит таблицу с именем `table4`. Таблица состоит из трех столбцов, которые по порядку `col1`, `col2` и `col3`.

Ваш файл данных `example4.csv` также состоит из трех столбцов, которые могут быть последовательно сопоставлены с `col1`, `col2` и `col3` таблицы `table4`.

Если вы хотите загрузить только записи данных, значения которых в первом столбце `example4.csv` равны `20180601`, в `table4`, выполните следующую команду:

```
curl --location-trusted -u <username>:<password> -H "label:label4" \
    -H "Expect:100-continue" \
    -H "columns: col1, col2, col3"\
    -H "where: col1 = 20180601" \
    -T example4.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table4/_stream_load
```

> **ПРИМЕЧАНИЕ**
>
> В предыдущем примере `example4.csv` и `table4` имеют одинаковое количество столбцов, которые могут быть последовательно сопоставлены, но вам нужно использовать предложение WHERE для указания условий фильтра на основе столбцов. Поэтому вам нужно использовать параметр `columns` для определения временных имен для столбцов `example4.csv`.

Установка целевых partition

Ваша база данных Selena `test_db` содержит таблицу с именем `table5`. Таблица состоит из трех столбцов, которые по порядку `col1`, `col2` и `col3`.

Ваш файл данных `example5.csv` также состоит из трех столбцов, которые могут быть последовательно сопоставлены с `col1`, `col2` и `col3` таблицы `table5`.

Если вы хотите загрузить все данные из `example5.csv` в partition `p1` и `p2` таблицы `table5`, выполните следующую команду:

```
curl --location-trusted -u <username>:<password>  -H "label:label5" \
    -H "Expect:100-continue" \
    -H "partitions: p1, p2" \
    -T example5.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table5/_stream_load
```

Установка строгого режима и часового пояса

Ваша база данных Selena `test_db` содержит таблицу с именем `table6`. Таблица состоит из трех столбцов, которые по порядку `col1`, `col2` и `col3`.

Ваш файл данных `example6.csv` также состоит из трех столбцов, которые могут быть последовательно сопоставлены с `col1`, `col2` и `col3` таблицы `table6`.

Если вы хотите загрузить все данные из `example6.csv` в `table6`, используя строгий режим и часовой пояс `Africa/Abidjan`, выполните следующую команду:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "strict_mode: true" \
    -H "timezone: Africa/Abidjan" \
    -T example6.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table6/_stream_load
```

Загрузка данных в таблицы, содержащие столбцы типа HLL

Ваша база данных Selena `test_db` содержит таблицу с именем `table7`. Таблица состоит из двух столбцов типа HLL, которые по порядку `col1` и `col2`.

Ваш файл данных `example7.csv` также состоит из двух столбцов, среди которых первый столбец может быть сопоставлен с `col1` таблицы `table7`, а второй столбец не может быть сопоставлен ни с одним столбцом таблицы `table7`. Значения в первом столбце `example7.csv` могут быть преобразованы в данные типа HLL с помощью функций перед загрузкой в `col1` таблицы `table7`.

Если вы хотите загрузить данные из `example7.csv` в `table7`, выполните следующую команду:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "columns: temp1, temp2, col1=hll_hash(temp1), col2=hll_empty()" \
    -T example7.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table7/_stream_load
```

> **ПРИМЕЧАНИЕ**
>
> В предыдущем примере два столбца `example7.csv` названы `temp1` и `temp2` по порядку с использованием параметра `columns`. Затем функции используются для преобразования данных следующим образом:
>
> - Функция `hll_hash` используется для преобразования значений в `temp1` `example7.csv` в данные типа HLL и сопоставления `temp1` `example7.csv` с `col1` `table7`.
> - Функция `hll_empty` используется для заполнения указанного значения по умолчанию в `col2` `table7`.

Для использования функций `hll_hash` и `hll_empty` см. hll_hash и hll_empty.

Загрузка данных в таблицы, содержащие столбцы типа BITMAP

Ваша база данных Selena `test_db` содержит таблицу с именем `table8`. Таблица состоит из двух столбцов типа BITMAP, которые по порядку `col1` и `col2`.

Ваш файл данных `example8.csv` также состоит из двух столбцов, среди которых первый столбец может быть сопоставлен с `col1` таблицы `table8`, а второй столбец не может быть сопоставлен ни с одним столбцом таблицы `table8`. Значения в первом столбце `example8.csv` могут быть преобразованы с помощью функций перед загрузкой в `col1` таблицы `table8`.

Если вы хотите загрузить данные из `example8.csv` в `table8`, выполните следующую команду:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "columns: temp1, temp2, col1=to_bitmap(temp1), col2=bitmap_empty()"
\
    -T example8.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table8/_stream_load
```

> **ПРИМЕЧАНИЕ**
>
> В предыдущем примере два столбца `example8.csv` названы `temp1` и `temp2` по порядку с использованием параметра `columns`. Затем функции используются для преобразования данных следующим образом:
>
> - Функция `to_bitmap` используется для преобразования значений в `temp1` `example8.csv` в данные типа BITMAP и сопоставления `temp1` `example8.csv` с `col1` `table8`.
> - Функция `bitmap_empty` используется для заполнения указанного значения по умолчанию в `col2` `table8`.

Для использования функций `to_bitmap` и `bitmap_empty` см. to_bitmap и bitmap_empty.

Установка `skip_header`, `trim_space`, `enclose` и `escape`

Ваша база данных Selena `test_db` содержит таблицу с именем `table9`. Таблица состоит из трех столбцов, которые по порядку `col1`, `col2` и `col3`.

Ваш файл данных `example9.csv` также состоит из трех столбцов, которые последовательно сопоставлены с `col2`, `col1` и `col3` таблицы `table13`.

Если вы хотите загрузить все данные из `example9.csv` в `table9` с намерением пропустить первые пять строк `example9.csv`, удалить пробелы перед и после разделителей столбцов и установить `enclose` в `\` и `escape` в `\`, выполните следующую команду:

```
curl --location-trusted -u <username>:<password> -H "label:3875" \
    -H "Expect:100-continue" \
    -H "trim_space: true" -H "skip_header: 5" \
    -H "column_separator:," -H "enclose:\"" -H "escape:\\" \
    -H "columns: col2, col1, col3" \
    -T example9.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/tbl9/_stream_load
```

Установка `column_separator` и `row_delimiter`

Ваша база данных Selena `test_db` содержит таблицу с именем `table10`. Таблица состоит из трех столбцов, которые по порядку `col1`, `col2` и `col3`.

Ваш файл данных `example10.csv` также состоит из трех столбцов, которые могут быть последовательно сопоставлены с `col1`, `col2` и `col3` таблицы `table10`. Столбцы в строке данных разделены запятыми (`,`), а строки данных разделены двумя последовательными непечатаемыми символами `\r\n`.

Если вы хотите загрузить все данные из `example10.csv` в `table10`, выполните следующую команду:

```
curl --location-trusted -u <username>:<password> -H "label:label10" \
    -H "Expect:100-continue" \
    -H "column_separator:," \
    -H "row_delimiter:\\x0D0A" \
    -T example10.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table10/_stream_load
```

Загрузка данных JSON

В этом разделе описываются настройки параметров, на которые вам нужно обратить внимание при загрузке данных JSON.

Ваша база данных Selena `test_db` содержит таблицу с именем `tbl1`, схема которой следующая:

```
`category` varchar(512) NULL COMMENT "",`author` varchar(512) NULL COMMENT
"",`title` varchar(512) NULL COMMENT "",`price` double NULL COMMENT ""
```

Загрузка данных JSON с использованием простого режима

Предположим, что ваш файл данных `example1.json` состоит из следующих данных:

```
{"category":"C++","author":"avc","title":"C++ primer","price":895}
```

Чтобы загрузить все данные из `example1.json` в `tbl1`, выполните следующую команду:

```
curl --location-trusted -u <username>:<password> -H "label:label6" \
    -H "Expect:100-continue" \
    -H "format: json" \
    -T example1.json -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/tbl1/_stream_load
```

> **ПРИМЕЧАНИЕ**
>
> В предыдущем примере параметры `columns` и `jsonpaths` не указаны. Поэтому ключи в `example1.json` сопоставляются по имени со столбцами `tbl1`.

Для увеличения пропускной способности Stream Load поддерживает загрузку нескольких записей данных одновременно. Пример:

```
[{"category":"C++","author":"avc","title":"C++ primer","price":89.5},
{"category":"Java","author":"avc","title":"Effective Java","price":95},
{"category":"Linux","author":"avc","title":"Linux kernel","price":195}]
```

Загрузка данных JSON с использованием режима сопоставления

Selena выполняет следующие шаги для сопоставления и обработки данных JSON:

1. (Необязательно) Удаляет самую внешнюю структуру массива в соответствии с настройкой параметра `strip_outer_array`.

   > **ПРИМЕЧАНИЕ**
   >
   > Этот шаг выполняется только тогда, когда самый внешний слой данных JSON — это структура массива, обозначенная парой квадратных скобок `[]`. Вам нужно установить `strip_outer_array` в `true`.

2. (Необязательно) Сопоставляет корневой элемент данных JSON в соответствии с настройкой параметра `json_root`.

> **ПРИМЕЧАНИЕ**
>
> Этот шаг выполняется только тогда, когда данные JSON имеют корневой элемент. Вам нужно указать корневой элемент, используя параметр `json_root`.

3. Извлекает указанные данные JSON в соответствии с настройкой параметра `jsonpaths`.

Загрузка данных JSON с использованием режима сопоставления без указания корневого элемента

Предположим, что ваш файл данных `example2.json` состоит из следующих данных:

```
[{"category":"xuxb111","author":"1avc","title":"SayingsoftheCentury","price":
{"category":"xuxb222","author":"2avc","title":"SayingsoftheCentury","price":8
{"category":"xuxb333","author":"3avc","title":"SayingsoftheCentury","price":8
```

Чтобы загрузить только `category`, `author` и `price` из `example2.json`, выполните следующую команду:

```
curl --location-trusted -u <username>:<password> -H "label:label7" \
    -H "Expect:100-continue" \
    -H "format: json" \
    -H "strip_outer_array: true" \
    -H "jsonpaths: [\"$.category\",\"$.price\",\"$.author\"]" \
    -H "columns: category, price, author" \
    -T example2.json -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/tbl1/_stream_load
```

> **ПРИМЕЧАНИЕ**
>
> В предыдущем примере самый внешний слой данных JSON — это структура массива, обозначенная парой квадратных скобок `[]`. Структура массива состоит из нескольких объектов JSON, каждый из которых представляет собой запись данных. Поэтому вам нужно установить `strip_outer_array` в `true`, чтобы удалить самую

> внешнюю структуру массива. Ключ **title**, который вы не хотите загружать, игнорируется во время загрузки.

Загрузка данных JSON с использованием режима сопоставления с указанным корневым элементом

Предположим, что ваш файл данных `example3.json` состоит из следующих данных:

```
{"id": 10001,"RECORDS":
[{"category":"11","title":"SayingsoftheCentury","price":895,"timestamp":15891
{"category":"22","author":"2avc","price":895,"timestamp":1589191487},
{"category":"33","author":"3avc","title":"SayingsoftheCentury","timestamp":15
["3 records", "there will be 3 rows"]}
```

Чтобы загрузить только `category`, `author` и `price` из `example3.json`, выполните следующую команду:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "format: json" \
    -H "json_root: $.RECORDS" \
    -H "strip_outer_array: true" \
    -H "jsonpaths: [\"$.category\",\"$.price\",\"$.author\"]" \
    -H "columns: category, price, author" -H "label:label8" \
    -T example3.json -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/tbl1/_stream_load
```

> **ПРИМЕЧАНИЕ**
>
> В предыдущем примере самый внешний слой данных JSON — это структура массива, обозначенная парой квадратных скобок `[]`. Структура массива состоит из нескольких объектов JSON, каждый из которых представляет собой запись данных. Поэтому вам нужно установить `strip_outer_array` в `true`, чтобы удалить самую внешнюю структуру массива. Ключи `title` и `timestamp`, которые вы не хотите загружать, игнорируются во время загрузки. Кроме того, параметр `json_root` используется для указания корневого элемента, который является массивом, данных JSON.

Объединение запросов Stream Load

- Выполните следующую команду, чтобы запустить задание Stream Load с включенным Merge Commit в синхронном режиме и установить окно слияния на `5000` миллисекунд и степень параллелизма на `2`:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "column_separator:," \
    -H "columns: id, name, score" \
    -H "enable_merge_commit:true" \
    -H "merge_commit_interval_ms:5000" \
    -H "merge_commit_parallel:2" \
    -T example1.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/mydatabase/table1/_stream_load
```

- Выполните следующую команду, чтобы запустить задание Stream Load с включенным Merge Commit в асинхронном режиме и установить окно слияния на `60000` миллисекунд и степень параллелизма на `2`:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "column_separator:," \
    -H "columns: id, name, score" \
    -H "enable_merge_commit:true" \
    -H "merge_commit_async:true" \
    -H "merge_commit_interval_ms:60000" \
    -H "merge_commit_parallel:2" \
    -T example1.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/mydatabase/table1/_stream_load
```

3.4 Catalog

CREATE EXTERNAL CATALOG

CREATE EXTERNAL CATALOG создаёт внешний каталог. Вы можете использовать внешние каталоги для запросов к данным во внешних источниках данных без загрузки данных в Selena или создания внешних таблиц. В настоящее время вы можете создать следующие типы внешних каталогов:

- Hive catalog: используется для запросов к данным из Apache Hive™.
- Iceberg catalog: используется для запросов к данным из Apache Iceberg.
- Hudi catalog: используется для запросов к данным из Apache Hudi.
- Delta Lake catalog: используется для запросов к данным из Delta Lake.
- JDBC catalog: используется для запросов к данным из JDBC-совместимых источников данных.
- Elasticsearch catalog: используется для запросов к данным из Elasticsearch. Каталоги Elasticsearch поддерживаются начиная с версии 3.1.
- Paimon catalog: используется для запросов к данным из Apache Paimon. Каталоги Paimon поддерживаются начиная с версии 3.1.
- Unified catalog: используется для запросов к данным из Hive, Iceberg, Hudi и Delta Lake как к единому источнику данных. Унифицированные каталоги поддерживаются начиная с версии 3.2.

> **ПРИМЕЧАНИЕ**
>
> - В версии 3.0 и более поздних для выполнения этого оператора требуется привилегия CREATE EXTERNAL CATALOG на уровне SYSTEM.
> - Перед созданием внешних каталогов настройте ваш cluster Selena в соответствии с требованиями системы хранения данных (например, Amazon S3), сервиса метаданных (например, Hive metastore) и сервиса аутентификации (например, Kerberos) внешних источников данных. Для получения дополнительной информации см. раздел "Перед началом работы" в каждой теме о внешних каталогах.

Синтаксис

```
CREATE EXTERNAL CATALOG [IF NOT EXISTS] <catalog_name>
[COMMENT <comment>]
PROPERTIES ("key"="value", ...)
```

Параметры

Параметр	Обязательный	Описание
catalog_name	Да	Имя внешнего каталога. Правила именования см. в разделе Системные ограничения.

Параметр	Обязательный	Описание
comment	Нет	Описание внешнего каталога.
PROPERTIES	Да	Свойства внешнего каталога. Настройте свойства в зависимости от типа внешнего каталога. Для получения дополнительной информации см. Hive catalog, Iceberg catalog, Hudi catalog, Delta Lake catalog и JDBC Catalog.

Примеры

Пример 1: Создание каталога Hive с именем `hive_metastore_catalog`. Соответствующий cluster Hive использует Hive metastore в качестве сервиса метаданных.

```
CREATE EXTERNAL CATALOG hive_metastore_catalog
COMMENT "External catalog to Hive"
PROPERTIES(
   "type"="hive",
   "hive.metastore.uris"="thrift://xx.xx.xx.xx:9083"
);
```

Пример 2: Создание каталога Hive с именем `hive_glue_catalog`. Соответствующий cluster Hive использует AWS Glue в качестве сервиса метаданных.

```
CREATE EXTERNAL CATALOG hive_glue_catalog
COMMENT "External catalog to Hive"
PROPERTIES(
    "type"="hive",
    "hive.metastore.type"="glue",
    "aws.hive.metastore.glue.aws-access-key"="xxxxxx",
    "aws.hive.metastore.glue.aws-secret-key"="xxxxxxxxxxxx",
    "aws.hive.metastore.glue.endpoint"="https://glue.x-x-x.amazonaws.com"
);
```

Пример 3: Создание каталога Iceberg с именем `iceberg_metastore_catalog`. Соответствующий cluster Iceberg использует Hive metastore в качестве сервиса метаданных.

```
CREATE EXTERNAL CATALOG iceberg_metastore_catalog
COMMENT "External catalog to Iceberg"
PROPERTIES(
    "type"="iceberg",
    "iceberg.catalog.type"="hive",
    "iceberg.catalog.hive.metastore.uris"="thrift://xx.xx.xx.xx:9083"
);
```

Пример 4: Создание каталога Iceberg с именем `iceberg_glue_catalog`. Соответствующий cluster Iceberg использует AWS Glue в качестве сервиса метаданных.

```
CREATE EXTERNAL CATALOG iceberg_glue_catalog
COMMENT "External catalog to Iceberg"
PROPERTIES(
    "type"="iceberg",
    "iceberg.catalog.type"="glue",
    "aws.hive.metastore.glue.aws-access-key"="xxxxx",
    "aws.hive.metastore.glue.aws-secret-key"="xxxxxxxxxxxx",
    "aws.hive.metastore.glue.endpoint"="https://glue.x-x-x.amazonaws.com"
);
```

Пример 5: Создание каталога Hudi с именем `hudi_metastore_catalog`. Соответствующий cluster Hudi использует Hive metastore в качестве сервиса метаданных.

```
CREATE EXTERNAL CATALOG hudi_metastore_catalog
COMMENT "External catalog to Hudi"
PROPERTIES(
    "type"="hudi",
    "hive.metastore.uris"="thrift://xx.xx.xx.xx:9083"
);
```

Пример 6: Создание каталога Hudi с именем `hudi_glue_catalog`. Соответствующий cluster Hudi использует AWS Glue в качестве сервиса метаданных.

```
CREATE EXTERNAL CATALOG hudi_glue_catalog
COMMENT "External catalog to Hudi"
PROPERTIES(
    "type"="hudi",
    "hive.metastore.type"="glue",
    "aws.hive.metastore.glue.aws-access-key"="xxxxxx",
    "aws.hive.metastore.glue.aws-secret-key"="xxxxxxxxxxxx",
    "aws.hive.metastore.glue.endpoint"="https://glue.x-x-x.amazonaws.com"
);
```

Пример 7: Создание каталога Delta Lake с именем `delta_metastore_catalog`. Соответствующий сервис Delta Lake использует Hive metastore в качестве сервиса метаданных.

```
CREATE EXTERNAL CATALOG delta_metastore_catalog
COMMENT "External catalog to Delta"
PROPERTIES(
    "type"="deltalake",
    "hive.metastore.uris"="thrift://xx.xx.xx.xx:9083"
);
```

Пример 8: Создание каталога Delta Lake с именем `delta_glue_catalog`. Соответствующий сервис Delta Lake использует AWS Glue в качестве сервиса метаданных.

```
CREATE EXTERNAL CATALOG delta_glue_catalog
COMMENT "External catalog to Delta"
PROPERTIES(
    "type"="deltalake",
    "hive.metastore.type"="glue",
    "aws.hive.metastore.glue.aws-access-key"="xxxxxx",
    "aws.hive.metastore.glue.aws-secret-key"="xxxxxxxxxxxx",
    "aws.hive.metastore.glue.endpoint"="https://glue.x-x-x.amazonaws.com"
);
```

См. также

- Для просмотра всех каталогов в вашем cluster Selena см. SHOW CATALOGS.
- Для просмотра оператора создания внешнего каталога см. SHOW CREATE CATALOG.

- Для удаления внешнего каталога из вашего cluster Selena см. DROP CATALOG.

DROP CATALOG

DROP CATALOG удаляет внешний каталог. Внутренний каталог не может быть удалён. Cluster Selena имеет только один внутренний каталог с именем `default_catalog`.

Синтаксис

```
DROP CATALOG [IF EXISTS] <catalog_name>
```

Параметры

`catalog_name`: Имя внешнего каталога.

Примеры

Создание каталога Hive с именем `hive1`.

```
CREATE EXTERNAL CATALOG hive1
PROPERTIES(
  "type"="hive",
  "hive.metastore.uris"="thrift://xx.xx.xx.xx:9083"
);
```

Удаление каталога Hive.

```
DROP CATALOG hive1;
```

SET CATALOG

Переключает на указанный каталог в текущей сессии.

Эта команда поддерживается начиная с версии 3.0.

> **ПРИМЕЧАНИЕ**
>
> Для вновь развёрнутого cluster Selena версии 3.1 вы должны иметь привилегию USAGE на целевой внешний каталог, если хотите выполнить SET CATALOG для

> переключения на этот каталог. Вы можете использовать GRANT для предоставления необходимых привилегий. Для cluster версии 3.1, обновлённого с более ранней версии, вы можете выполнить SET CATALOG с унаследованными привилегиями.

Синтаксис

```
SET CATALOG <catalog_name>
```

Параметр

`catalog_name`: имя каталога для использования в текущей сессии. Вы можете переключиться на внутренний или внешний каталог. Если указанный каталог не существует, будет вызвано исключение.

Примеры

Выполните следующую команду для переключения на каталог Hive с именем `hive_metastore` в текущей сессии:

```
SET CATALOG hive_metastore;
```

Выполните следующую команду для переключения на внутренний каталог `default_catalog` в текущей сессии:

```
SET CATALOG default_catalog;
```

SHOW CATALOGS

SHOW CATALOGS запрашивает все каталоги в текущем cluster Selena, включая внутренний каталог и внешние каталоги.

> **ПРИМЕЧАНИЕ**
>
> SHOW CATALOGS возвращает внешние каталоги пользователям, имеющим привилегию USAGE на этот внешний каталог. Если пользователи или роли не имеют этой привилегии ни на один внешний каталог, эта команда возвращает только default_catalog.

Синтаксис

```
SHOW CATALOGS
```

Вывод

```
+----------+--------+----------+
| Catalog  | Type   | Comment  |
+----------+--------+----------+
```

Следующая таблица описывает поля, возвращаемые этим оператором.

Поле	Описание
Catalog	Имя каталога.
Type	Тип каталога. `Internal` возвращается, если каталог — `default_catalog`. Соответствующий тип каталога возвращается, если каталог является внешним, например `Hive`, `Hudi` или `Iceberg`.
Comment	Комментарии к каталогу. Selena не поддерживает добавление комментариев к внешнему каталогу. Поэтому значение `NULL` для внешнего каталога. Если каталог — `default_catalog`, комментарий по умолчанию — `An internal catalog contains this cluster's self-managed tables.` `default_catalog` — единственный внутренний каталог в cluster Selena.

Примеры

Запрос всех каталогов в текущем cluster.

```
SHOW CATALOGS\G
*************************** 1. row ***************************
Catalog: default_catalog
   Type: Internal
Comment: An internal catalog contains this cluster's self-managed tables.
*************************** 2. row ***************************
Catalog: hudi_catalog
   Type: Hudi
Comment: NULL
*************************** 3. row ***************************
Catalog: iceberg_catalog
   Type: Iceberg
Comment: NULL
```

SHOW CREATE CATALOG

SHOW CREATE CATALOG запрашивает оператор создания внешнего каталога.

Обратите внимание, что информация, связанная с аутентификацией, в возвращаемом результате будет анонимизирована.

Эта команда поддерживается начиная с версии 3.0.

Синтаксис

```
SHOW CREATE CATALOG <catalog_name>;
```

Параметры

Параметр	Обязательный	Описание
catalog_name	Да	Имя каталога, оператор создания которого вы хотите просмотреть.

Возвращаемый результат

```
+------------+-----------------+
| Catalog    | Create Catalog  |
+------------+-----------------+
```

Поле	Описание
Catalog	Имя каталога.
Create Catalog	Оператор, который был выполнен для создания каталога.

Примеры

Следующий пример запрашивает оператор создания каталога Hive с именем `hive_catalog_hms`:

```
SHOW CREATE CATALOG hive_catalog_hms;
```

Возвращаемый результат выглядит следующим образом:

```
CREATE EXTERNAL CATALOG `hive_catalog_hms`
PROPERTIES ("aws.s3.access_key"  =  "AK******M4",
"hive.metastore.type"  =  "glue",
"aws.s3.secret_key"  =  "iV******iD",
"aws.glue.secret_key"  =  "iV******iD",
"aws.s3.use_instance_profile"  =  "false",
"aws.s3.region"  =  "us-west-1",
"aws.glue.region"  =  "us-west-1",
"type"  =  "hive",
"aws.glue.access_key"  =  "AK******M4"
)
```

3.5 Database

ALTER DATABASE

ALTER DATABASE настраивает свойства указанной базы данных.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия ALTER на целевую базу данных. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис

1. Установить квоту данных базы данных в B/K/KB/M/MB/G/GB/T/TB/P/PB.

```
ALTER DATABASE <db_name> SET DATA QUOTA <quota>;
```

2. Переименовать базу данных.

```
ALTER DATABASE <db_name> RENAME <new_db_name>;
```

3. Установить квоту replica базы данных.

```
ALTER DATABASE <db_name> SET REPLICA QUOTA <quota>;
```

4. Установить том хранения базы данных

```
ALTER DATABASE <db_name> SET ("storage_volume" = "
<new_storage_volume_name>");
```

Примечание:

```
- После переименования базы данных используйте команды REVOKE и GRANT для
изменения соответствующих разрешений пользователя при необходимости.
- Квота данных по умолчанию и квота replica по умолчанию для базы данных
составляют 2^63-1.
- Том хранения базы данных по умолчанию — это том хранения по умолчанию на
момент создания базы данных, если свойство "storage_volume" не указано
явно.
```

Примеры

1. Установить квоту данных для базы данных.

```
ALTER DATABASE example_db SET DATA QUOTA 109951162777760B;
-- Указанная выше единица измерения — байты, эквивалентно следующим
операторам.
ALTER DATABASE example_db SET DATA QUOTA 10T;
ALTER DATABASE example_db SET DATA QUOTA 100G;
ALTER DATABASE example_db SET DATA QUOTA 200M;
```

2. Переименовать базу данных `example_db` в `example_db2`.

```
ALTER DATABASE example_db RENAME example_db2;
```

3. Установить квоту replica базы данных.

```
ALTER DATABASE example_db SET REPLICA QUOTA 102400;
```

4. Установить том хранения базы данных.

```
ALTER DATABASE example_db SET ("storage_volume" =
"aws_s3_storage_volume");
```

См. также

- CREATE DATABASE
- USE
- SHOW DATABASES
- DESC
- DROP DATABASE

CREATE DATABASE

CREATE DATABASE используется для создания баз данных.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия CREATE DATABASE на целевой каталог. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.

Синтаксис

```
CREATE DATABASE [IF NOT EXISTS] <db_name>
[PROPERTIES ("key"="value", ...)]
```

Параметры

`db_name`: имя создаваемой базы данных. Правила именования см. в разделе Системные ограничения.

PROPERTIES (Необязательно)

`storage_volume`: указывает имя тома хранения, используемого для хранения базы данных в cluster с разделяемыми данными. Если не указано, к базе данных будет прикреплён том хранения по умолчанию. Свойство может быть изменено через оператор `ALTER DATABASE`.

Примеры

1. Создать базу данных `db_test`.

```
CREATE DATABASE db_test;
```

2. Создать облачную базу данных `cloud_db` с томом хранения `s3_storage_volume`.

```
CREATE DATABASE cloud_db
PROPERTIES ("storage_volume"="s3_storage_volume");
```

См. также

- SHOW CREATE DATABASE
- USE
- SHOW DATABASES
- DESC
- DROP DATABASE
- ALTER DATABASE

DROP DATABASE

DROP DATABASE используется для удаления базы данных в Selena.

> **ПРИМЕЧАНИЕ**
>
> Для выполнения этой операции требуется привилегия DROP на целевую базу данных.

Синтаксис

```
DROP DATABASE [IF EXISTS] <db_name> [FORCE]
```

Обратите внимание на следующие моменты:

- После выполнения DROP DATABASE для удаления базы данных вы можете восстановить удалённую базу данных с помощью оператора RECOVER в течение указанного периода хранения (период хранения по умолчанию составляет один день),

но конвейеры (поддерживаются начиная с версии 3.2), удалённые вместе с базой данных, восстановить невозможно.

- Если вы выполните `DROP DATABASE FORCE` для удаления базы данных, база данных удаляется напрямую без каких-либо проверок наличия незавершённых операций и не может быть восстановлена. Как правило, эта операция не рекомендуется.
- При удалении базы данных все конвейеры (поддерживаются начиная с версии 3.2), принадлежащие базе данных, удаляются вместе с ней.

Примеры

1. Удалить базу данных db_text.

```
DROP DATABASE db_test;
```

См. также

- CREATE DATABASE
- SHOW CREATE DATABASE
- USE
- DESC
- SHOW DATABASES

SHOW CREATE DATABASE

Показывает SQL-команду, использованную для создания базы данных.

Синтаксис

```
SHOW CREATE DATABASE <db_name>
```

Параметры

`db_name`: имя базы данных, обязательный параметр.

Возвращаемые значения

- `Database`: имя базы данных
- `Create Database`: SQL-команда, использованная для создания базы данных

Примеры

```
mysql > show create database zj_test;
+----------+---------------------------+
| Database | Create Database           |
+----------+---------------------------+
| zj_test  | CREATE DATABASE `zj_test` |
+----------+---------------------------+
```

См. также

- CREATE DATABASE
- SHOW DATABASES
- USE
- DESC
- DROP DATABASE

SHOW DATA

SHOW DATA используется для отображения объёма данных, количества копий и количества статистических строк в базе данных или таблице базы данных.

Синтаксис:

```
SHOW DATA [FROM [<db_name>.]<table_name>]
```

Примечание:

1. Если предложение FROM не указано, отображается объём данных и копий с разбивкой по каждой таблице в текущей базе данных. При этом объём данных — это общий объём данных всех replica. Количество копий — это количество копий всех партиций таблицы и всех материализованных представлений.
2. Если предложение FROM указано, отображается объём данных, количество копий и количество статистических строк с разбивкой по каждому материализованному представлению таблицы. При этом объём данных — это общий объём данных всех replica. Количество копий — это количество копий всех партиций, соответствующих материализованному представлению. Количество статистических строк — это

количество статистических строк всех партиций, соответствующих материализованному представлению.

3. При подсчёте количества строк берётся копия с наибольшим количеством строк среди нескольких копий.
4. Строка Total в результирующем наборе представляет итоговую строку. Строка Quota представляет квоту, установленную для текущей базы данных. Строка Left представляет оставшуюся квоту.

Примеры

Пример 1: Отображение объёма данных, количества копий, суммарного объёма данных и суммарного количества копий каждой таблицы в текущей базе данных.

```
SHOW DATA;
+-----------+-------------+--------------+
| TableName | Size        | ReplicaCount |
+-----------+-------------+--------------+
| tbl1      | 900.000 B   | 6            |
| tbl2      | 500.000 B   | 3            |
| Total     | 1.400 KB    | 9            |
| Quota     | 1024.000 GB | 1073741824   |
| Left      | 1021.921 GB | 1073741815   |
+-----------+-------------+--------------+
```

Пример 2: Отображение объёма данных с разбивкой, количества копий и количества статистических строк в указанной таблице.

```
SHOW DATA FROM example_db.test;
+-----------+-----------+-----------+--------------+----------+
| TableName | IndexName | Size      | ReplicaCount | RowCount |
+-----------+-----------+-----------+--------------+----------+
| test      | r1        | 10.000MB  | 30           | 10000    |
|           | r2        | 20.000MB  | 30           | 20000    |
|           | test2     | 50.000MB  | 30           | 50000    |
|           | Total     | 80.000    | 90           |          |
+-----------+-----------+-----------+--------------+----------+
```

SHOW DATABASES

SHOW DATABASES выводит список баз данных в вашем текущем cluster Selena или во внешнем источнике данных. Selena поддерживает просмотр баз данных внешнего источника данных начиная с версии 2.3.

Синтаксис

```
SHOW DATABASES [FROM <catalog_name>]
```

Параметры

Параметр	Обязательный	Описание
catalog_name	Нет	Имя внутреннего каталога или внешнего каталога. • Если вы не укажете параметр или укажете имя внутреннего каталога, которым является `default_catalog`, вы можете просмотреть базы данных в вашем текущем cluster Selena. • Если вы установите значение параметра равным имени внешнего каталога, вы можете просмотреть базы данных в соответствующем внешнем источнике данных. Вы можете выполнить SHOW CATALOGS для просмотра внутренних и внешних каталогов.

Примеры

Пример 1: Просмотр баз данных в вашем текущем cluster Selena.

```
SHOW DATABASES;
```

Или

```
SHOW DATABASES FROM default_catalog;
```

Вывод предыдущих операторов выглядит следующим образом.

```
+----------+
| Database |
+----------+
| db1      |
| db2      |
| db3      |
+----------+
```

Пример 2: Просмотр баз данных в cluster Hive с помощью внешнего каталога `Hive1`.

```
SHOW DATABASES FROM hive1;

+-----------+
| Database  |
+-----------+
| hive_db1  |
| hive_db2  |
| hive_db3  |
+-----------+
```

См. также

- CREATE DATABASE
- SHOW CREATE DATABASE
- USE
- DESC
- DROP DATABASE

USE

USE указывает активную базу данных для вашей сессии. После этого вы можете выполнять операции, такие как создание таблиц и выполнение запросов.

Синтаксис

```
USE [<catalog_name>.]<db_name>
```

Параметры

Параметр	Обязательный	Описание
catalog_name	Нет	Имя каталога. • Если этот параметр не указан, по умолчанию используется база данных в `default_catalog`. • Вы должны указать этот параметр при использовании базы данных из внешнего каталога. Дополнительную информацию см. в Примере 2. • Вы должны указать этот параметр при переключении баз данных между разными каталогами. Дополнительную информацию см. в Примере 3. Дополнительную информацию о каталогах см. в разделе Обзор.
db_name	Да	Имя базы данных. База данных должна существовать.

Примеры

Пример 1: Использование `example_db` из `default_catalog` в качестве активной базы данных вашей сессии.

```
USE default_catalog.example_db;
```

Или

```
USE example_db;
```

Пример 2: Использование `example_db` из `hive_catalog` в качестве активной базы данных вашей сессии.

```
USE hive_catalog.example_db;
```

Пример 3: Переключение активной базы данных вашей сессии с `hive_catalog.example_table1` на `iceberg_catalog.example_table2`.

```
USE iceberg_catalog.example_table2;
```

3.6 Resource

ALTER RESOURCE

Используйте команду ALTER RESOURCE для изменения свойств ресурса.

Синтаксис

```
ALTER RESOURCE 'resource_name' SET PROPERTIES ("key"="value", ...)
```

Параметры

- `resource_name`: имя ресурса, который нужно изменить.
- `PROPERTIES ("key"="value", ...)`: свойства ресурса. Вы можете изменять различные свойства в зависимости от типа ресурса. В настоящее время Selena поддерживает изменение URI Hive metastore для следующих ресурсов.
  - Ресурсы Apache Iceberg поддерживают изменение следующих свойств:
    - `iceberg.catalog-impl`: полное имя класса custom catalog.
    - `iceberg.catalog.hive.metastore.uris`: URI Hive metastore.
  - Ресурсы Apache Hive™ и Apache Hudi поддерживают изменение `hive.metastore.uris`, который указывает URI Hive metastore.

Примечания к использованию

После того как вы используете ресурс для создания внешней таблицы, если вы измените URI Hive metastore этого ресурса, внешняя таблица станет недоступной. Если вы всё ещё хотите использовать внешнюю таблицу для запроса данных, убедитесь, что новый metastore содержит таблицу с таким же именем и схемой, как в исходном metastore.

Примеры

Изменить URI Hive metastore для Hive ресурса `hive0`.

```
ALTER RESOURCE 'hive0' SET PROPERTIES ("hive.metastore.uris" =
"thrift://xx.xx.xx.xx:9083")
```

CREATE RESOURCE

Создаёт ресурсы. Могут быть созданы следующие типы ресурсов: Apache Spark™, Apache Hive™, Apache Iceberg, Apache Hudi и JDBC. Spark ресурсы используются в Spark Load для управления информацией о загрузке, такой как конфигурации YARN, путь хранения промежуточных данных и конфигурации Broker. Ресурсы Hive, Iceberg, Hudi и JDBC используются для управления информацией о доступе к источникам данных при запросах к внешним таблицам.

ПОДСКАЗКА

- Только пользователи с привилегией CREATE RESOURCE на уровне SYSTEM могут выполнять эту операцию.
- Вы можете создавать JDBC ресурсы только в Selena v1.5.2 и выше.

Синтаксис

```
CREATE [EXTERNAL] RESOURCE "resource_name"
PROPERTIES ("key"="value", ...)
```

Параметры

- `resource_name`: имя создаваемого ресурса. Для соглашений об именовании см. Системные ограничения.

- `PROPERTIES`: определяет свойства типа ресурса. PROPERTIES различаются в зависимости от типа ресурса. Подробности см. в разделе Примеры.

Примеры

1. Создать Spark ресурс с именем spark0 в режиме yarn Cluster.

```
CREATE EXTERNAL RESOURCE "spark0"
PROPERTIES
(
    "type" = "spark",
    "spark.master" = "yarn",
    "spark.submit.deployMode" = "cluster",
    "spark.jars" = "xxx.jar,yyy.jar",
    "spark.files" = "/tmp/aaa,/tmp/bbb",
    "spark.executor.memory" = "1g",
    "spark.yarn.queue" = "queue0",
    "spark.hadoop.yarn.resourcemanager.address" = "127.0.0.1:9999",
    "spark.hadoop.fs.defaultFS" = "hdfs://127.0.0.1:10000",
    "working_dir" = "hdfs://127.0.0.1:10000/tmp/selena",
    "broker" = "broker0",
    "broker.username" = "user0",
    "broker.password" = "password0"
);
```

Параметры, связанные со Spark, следующие:

```
1. spark.master: обязательный. В настоящее время поддерживаются yarn и
spark://host:port.
2. spark.submit.deployMode: режим развёртывания программы Spark,
обязательный. Поддерживаются cluster и client.
3. spark.hadoop.yarn.resourcemanager.address: обязательный, когда
master это yarn.
4. spark.hadoop.fs.defaultFS: обязательный, когда master это yarn.
5. Другие параметры необязательны. Пожалуйста, обратитесь к
http://spark.apache.org/docs/latest/configuration.html
```

Если Spark используется для ETL, необходимо указать working_DIR и broker.

Инструкции следующие:

```
working_dir: Директория, используемая ETL. Обязательна, когда spark
используется как ETL ресурс. Например: hdfs://host:port/tmp/selena.
broker: Имя broker. Обязательно, когда spark используется как ETL
ресурс, и должен быть предварительно настроен с помощью команды `ALTER
SYSTEM ADD BROKER`.
broker.property_key: Это информация о свойствах, которую необходимо
указать, когда broker читает промежуточные файлы, созданные ETL.
```

2. Создать Hive ресурс с именем hive0.

```
CREATE EXTERNAL RESOURCE "hive0"
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.uris" = "thrift://10.10.44.98:9083"
);
```

DROP RESOURCE

Используйте DROP RESOURCE для удаления существующего ресурса. Только root или superuser могут удалять ресурсы.

Синтаксис:

```
DROP RESOURCE 'resource_name'
```

Примеры

1. Удалить Spark ресурс с именем spark0.

```
DROP RESOURCE 'spark0';
```

2. Удалить Hive ресурс с именем hive0.

```
DROP RESOURCE 'hive0';
```

SHOW RESOURCES

SHOW RESOURCES показывает ресурсы, на использование которых у пользователей есть разрешения. Обычные пользователи могут использовать только те ресурсы, на которые у них есть разрешение, в то время как пользователи root или admin могут просматривать все ресурсы.

Синтаксис:

```
SHOW RESOURCES
```

3.7 Table, bucket, partition, and index

ALTER TABLE

ALTER TABLE изменяет существующую таблицу, включая:

- Переименование таблицы, раздела, rollup или столбца
- Изменение комментария к таблице
- Изменение разделов (добавление/удаление разделов и изменение атрибутов разделов)
- Изменение метода bucketing и количества buckets
- Изменение столбцов (добавление/удаление столбцов, изменение порядка столбцов и изменение комментария к столбцу)
- Создание/удаление rollup
- Создание/удаление индекса
- Изменение свойств таблицы
- Атомарный обмен
- Ручное сжатие версий данных
- Удаление Primary Key Persistent Index

> **ПОДСКАЗКА**
>
> Эта операция требует привилегию ALTER на целевой таблице.

Синтаксис

```
ALTER TABLE [<db_name>.]<tbl_name>
alter_clause1[, alter_clause2, ...]
```

`alter_clause` может выполнять следующие операции: rename, comment, partition, bucket, column, rollup, index, table property, swap и compaction.

- rename: переименование таблицы, rollup, раздела или столбца (поддерживается **начиная с v1.5.2**).
- comment: изменение комментария к таблице (поддерживается **начиная с v1.5.2**).

- partition: изменение свойств раздела, удаление раздела или добавление раздела.
- bucket: изменение метода bucketing и количества buckets.
- column: добавление, удаление или переупорядочивание столбцов, изменение типа столбца или комментария
- rollup: создание или удаление rollup.
- index: изменение индексов.
- swap: атомарный обмен двух таблиц.
- compaction: выполнение ручного сжатия для объединения версий загруженных данных (поддерживается **начиная с v1.5.2**).
- drop persistent index: Удаление persistent index для таблицы с Primary Key в shared-data кластере (поддерживается **начиная с v1.5.2**).

Ограничения и примечания по использованию

- Операции с разделами, столбцами и rollup индексами не могут быть выполнены в одном операторе ALTER TABLE.
- Одна таблица может иметь только одну текущую операцию изменения схемы. Вы не можете запускать две команды изменения схемы для одной таблицы одновременно.
- Операции с bucket, column и rollup являются асинхронными операциями. Сообщение об успехе возвращается сразу после отправки задачи. Вы можете выполнить команду SHOW ALTER TABLE для проверки прогресса и команду CANCEL ALTER TABLE для отмены операции.
- Операции с rename, comment, partition, index и swap являются синхронными операциями, и возврат команды означает завершение выполнения.

Rename

Rename поддерживает изменение имени таблицы, rollup и имени раздела.

Переименование таблицы

```
ALTER TABLE <tbl_name> RENAME <new_tbl_name>
```

Переименование rollup

```
ALTER TABLE [<db_name>.]<tbl_name>
RENAME ROLLUP <old_rollup_name> <new_rollup_name>
```

Переименование раздела

```
ALTER TABLE [<db_name>.]<tbl_name>
RENAME PARTITION <old_partition_name> <new_partition_name>
```

Переименование столбца

Начиная с версии v1.5.2, Selena поддерживает переименование столбцов.

```
ALTER TABLE [<db_name>.]<tbl_name>
RENAME COLUMN <old_col_name> [ TO ] <new_col_name>
```

> **ПРИМЕЧАНИЕ**
>
> - После переименования столбца с A на B добавление нового столбца с именем A не поддерживается.
> - Материализованные представления, построенные на переименованном столбце, не будут работать. Вы должны пересоздать их для столбца с новым именем.

Alter table comment (from v1.5.2)

Синтаксис:

```
ALTER TABLE [<db_name>.]<tbl_name> COMMENT = "<new table comment>";
```

Modify partition

ADD PARTITION(S)

Вы должны строго следовать соответствующему синтаксису для добавления range разделов или list разделов.

ПРИМЕЧАНИЕ

- Добавление expression разделов не поддерживается.
- Обратите внимание, что `PARTITION BY date_trunc(column)` и `PARTITION BY time_slice(column)` считаются range разделением, несмотря на их формат expression разделения. Поэтому вы можете использовать следующий синтаксис для range разделов для добавления новых разделов в таблицы, использующие такие стратегии разделения.

Синтаксис:

- Range разделы

```
ALTER TABLE
    ADD { single_range_partition | multi_range_partitions }
[distribution_desc] ["key"="value"];

single_range_partition ::=
    PARTITION [IF NOT EXISTS] <partition_name> VALUES
partition_key_desc

partition_key_desc ::=
    { LESS THAN { MAXVALUE | value_list }
    | [ value_list , value_list ) } -- Обратите внимание, что [
представляет левый закрытый интервал.

value_list ::=
    ( <value> [, ...] )

multi_range_partitions ::=
    { PARTITIONS START ("<start_date_value>") END ("<end_date_value>")
EVERY ( INTERVAL <N> <time_unit> )
    | PARTITIONS START ("<start_integer_value>") END ("
<end_integer_value>") EVERY ( <granularity> ) } -- Значения столбца
раздела все равно должны быть заключены в двойные кавычки, даже если
значения столбца раздела, указанные START и END, являются целыми
числами. Однако значения интервала в предложении EVERY не нужно
заключать в двойные кавычки.
```

- List разделы

```
ALTER TABLE
    ADD PARTITION <partition_name> VALUES IN (value_list)
[distribution_desc] ["key"="value"];

value_list ::=
    value_item [, ...]

value_item ::=
    { <value> | ( <value> [, ...] ) }
```

Параметры:

- Параметры, связанные с разделами:
  - Для range разделов вы можете добавить один range раздел (`single_range_partition`) или несколько range разделов пакетно (`multi_range_partitions`).
  - Для list разделов вы можете добавить только один list раздел.
- `distribution_desc`:

  Вы можете установить количество buckets для нового раздела отдельно, но не можете установить метод bucketing отдельно.

- `"key"="value"`:

  Вы можете установить свойства для нового раздела. Для получения подробной информации см. CREATE TABLE.

Примеры:

- Range разделы
  - Если столбец раздела указан как `event_day` при создании таблицы, например `PARTITION BY RANGE(event_day)`, и после создания таблицы нужно добавить новый раздел, вы можете выполнить:

    ```
    ALTER TABLE site_access ADD PARTITION p4 VALUES LESS THAN ("2020-
    04-30");
    ```

- Если столбец раздела указан как `datekey` при создании таблицы, например `PARTITION BY RANGE (datekey)`, и после создания таблицы нужно добавить несколько разделов пакетно, вы можете выполнить:

```
ALTER TABLE site_access
    ADD PARTITIONS START ("2021-01-05") END ("2021-01-10") EVERY
(INTERVAL 1 DAY);
```

- List разделы

  - Если при создании таблицы указан один столбец раздела, например `PARTITION BY LIST (city)`, и после создания таблицы нужно добавить новый раздел, вы можете выполнить:

```
ALTER TABLE t_recharge_detail2
ADD PARTITION pCalifornia VALUES IN ("Los Angeles","San
Francisco","San Diego");
```

  - Если при создании таблицы указано несколько столбцов раздела, например `PARTITION BY LIST (dt,city)`, и после создания таблицы нужно добавить новый раздел, вы можете выполнить:

```
ALTER TABLE t_recharge_detail4
ADD PARTITION p202204_California VALUES IN
(
    ("2022-04-01", "Los Angeles"),
    ("2022-04-01", "San Francisco"),
    ("2022-04-02", "Los Angeles"),
    ("2022-04-02", "San Francisco")
);
```

DROP PARTITION(S)

- Удалить один раздел:

```
ALTER TABLE [<db_name>.]<tbl_name>
DROP PARTITION [ IF EXISTS ] <partition_name> [ FORCE ]
```

- Удалить разделы пакетно (Поддерживается с v1.5.2):

```
ALTER TABLE [<db_name>.]<tbl_name>
DROP PARTITIONS [ IF EXISTS ]  { partition_name_list |
multi_range_partitions } [ FORCE ]

partition_name_list ::= ( <partition_name> [, ... ] )

multi_range_partitions ::=
    { START ("<start_date_value>") END ("<end_date_value>") EVERY (
INTERVAL <N> <time_unit> )
    | START ("<start_integer_value>") END ("<end_integer_value>") EVERY (
<granularity> ) } -- Значения столбца раздела все равно должны быть
заключены в двойные кавычки, даже если значения столбца раздела являются
целыми числами. Однако значения интервала в предложении EVERY не нужно
заключать в двойные кавычки.
```

Примечания для `multi_range_partitions`:

- Применяется только к Range разделению.
- Используемые параметры соответствуют параметрам в ADD PARTITION(S).
- Поддерживает только разделы с одним Partition Key.
- Удалить разделы с Common Partition Expression (Поддерживается с v1.5.2):

```
ALTER TABLE [<db_name>.]<tbl_name>
DROP PARTITIONS WHERE <expr>
```

Начиная с версии v1.5.2, Selena поддерживает удаление разделов с помощью Common Partition Expression. Вы можете указать предложение WHERE с выражением для фильтрации разделов для удаления.

- Выражение объявляет разделы, которые будут удалены. Разделы, соответствующие условию в выражении, будут удалены пакетно. Будьте осторожны при продолжении.
- Выражение может содержать только столбцы раздела и константы. Столбцы, не являющиеся столбцами раздела, не поддерживаются.
- Common Partition Expression применяется к List разделам и Range разделам по-разному:

- Для таблиц с List разделами Selena поддерживает удаление разделов, отфильтрованных Common Partition Expression.
- Для таблиц с Range разделами Selena может фильтровать и удалять разделы только с использованием возможности partition pruning FE. Разделы, соответствующие предикатам, которые не поддерживаются partition pruning, не могут быть отфильтрованы и удалены.

Пример:

```
-- Удалить данные старше трех последних месяцев. Столбец `dt` является столбцом раздела таблицы.
ALTER TABLE t1 DROP PARTITIONS WHERE dt < CURRENT_DATE() - INTERVAL 3 MONTH;
```

ПРИМЕЧАНИЕ

- Сохраните хотя бы один раздел для разделенных таблиц.
- Если FORCE не указан, вы можете восстановить удаленные разделы с помощью команды RECOVER в течение указанного периода (по умолчанию 1 день).
- Если FORCE указан, разделы будут удалены напрямую, независимо от того, есть ли незавершенные операции с разделами, и они не могут быть восстановлены. Таким образом, эта операция обычно не рекомендуется.

Добавить временный раздел

Синтаксис:

```
ALTER TABLE [<db_name>.]<tbl_name>
ADD TEMPORARY PARTITION [IF NOT EXISTS] <partition_name>
{ single_range_partition | multi_range_partitions | list_partitions }
[DISTRIBUTED BY HASH (k1[,k2 ...]) [BUCKETS num]]

-- Для получения подробной информации о single_range_partition и
multi_range_partitions см. раздел ADD PARTITION(S) на этой странице.

list_partitions::=
    PARTITION <partition_name> VALUES IN (value_list)

value_list ::=
    value_item [, ...]

value_item ::=
    { <value> | ( <value> [, ...] ) }
```

Использовать временный раздел для замены текущего раздела

Синтаксис:

```
ALTER TABLE [<db_name>.]<tbl_name>
REPLACE PARTITION <partition_name>
partition_desc ["key"="value"]
WITH TEMPORARY PARTITION
partition_desc ["key"="value"]
[PROPERTIES ("key"="value", ...)]
```

Удалить временный раздел

Синтаксис:

```
ALTER TABLE [<db_name>.]<tbl_name>
DROP TEMPORARY PARTITION <partition_name>
```

Изменить свойства раздела

Синтаксис

```
ALTER TABLE [<db_name>.]<tbl_name>
MODIFY PARTITION { <partition_name> | ( <partition1_name> [,
<partition2_name> ...] ) | (*) }
SET ("key" = "value", ...);
```

Использование

- Следующие свойства раздела могут быть изменены:
  - storage_medium
  - storage_cooldown_ttl или storage_cooldown_time
  - replication_num
- Для таблицы, которая имеет только один раздел, имя раздела совпадает с именем таблицы. Если таблица разделена на несколько разделов, вы можете использовать `(*)` для изменения свойств всех разделов, что более удобно.
- Выполните `SHOW PARTITIONS FROM <tbl_name>` для просмотра свойств раздела после изменения.

Modify the bucketing method and number of buckets (from v1.5.2)

Синтаксис:

```
ALTER TABLE [<db_name>.]<table_name>
[ partition_names ]
[ distribution_desc ]

partition_names ::=
    (PARTITION | PARTITIONS) ( <partition_name> [, <partition_name> ...] )

distribution_desc ::=
    DISTRIBUTED BY RANDOM [ BUCKETS <num> ] |
    DISTRIBUTED BY HASH ( <column_name> [, <column_name> ...] ) [ [ DEFAULT
] BUCKETS <num> ]
```

Пример:

Например, исходная таблица - это таблица Duplicate Key, где используется hash bucketing, а количество buckets автоматически устанавливается Selena.

```
CREATE TABLE IF NOT EXISTS details (
    event_time DATETIME NOT NULL COMMENT "datetime of event",
    event_type INT NOT NULL COMMENT "type of event",
    user_id INT COMMENT "id of user",
    device_code INT COMMENT "device code",
    channel INT COMMENT ""
)
DUPLICATE KEY(event_time, event_type)
PARTITION BY date_trunc('day', event_time)
DISTRIBUTED BY HASH(user_id);

-- Вставить данные за несколько дней
INSERT INTO details (event_time, event_type, user_id, device_code, channel)
VALUES
-- Данные 26 ноября
('2023-11-26 08:00:00', 1, 101, 12345, 2),
('2023-11-26 09:15:00', 2, 102, 54321, 3),
('2023-11-26 10:30:00', 1, 103, 98765, 1),
-- Данные 27 ноября
('2023-11-27 08:30:00', 1, 104, 11111, 2),
('2023-11-27 09:45:00', 2, 105, 22222, 3),
('2023-11-27 11:00:00', 1, 106, 33333, 1),
-- Данные 28 ноября
('2023-11-28 08:00:00', 1, 107, 44444, 2),
('2023-11-28 09:15:00', 2, 108, 55555, 3),
('2023-11-28 10:30:00', 1, 109, 66666, 1);
```

Изменить только метод bucketing

> **УВЕДОМЛЕНИЕ**
>
> - Изменение применяется ко всем разделам в таблице и не может быть применено только к конкретным разделам.
> - Хотя нужно изменить только метод bucketing, количество buckets все равно нужно указать в команде с помощью `BUCKETS <num>`. Если `BUCKETS <num>` не указан, это означает, что количество buckets автоматически определяется Selena.

- Метод bucketing изменяется на random bucketing с hash bucketing, и количество buckets остается автоматически установленным Selena.

```
ALTER TABLE details DISTRIBUTED BY RANDOM;
```

- Ключи для hash bucketing изменяются на `user_id, event_time` с `event_time, event_type`. И количество buckets остается автоматически установленным Selena.

```
ALTER TABLE details DISTRIBUTED BY HASH(user_id, event_time);
```

Изменить только количество buckets

> **УВЕДОМЛЕНИЕ**
>
> Хотя нужно изменить только количество buckets, метод bucketing все равно нужно указать в команде, например, `HASH(user_id)`.

- Изменить количество buckets для всех разделов на 10 с автоматически установленного Selena.

```
ALTER TABLE details DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

- Изменить количество buckets по умолчанию для таблицы на 10 с автоматически установленного Selena **без изменения количества для существующих разделов** (поддерживается начиная с v1.5.2 и v2.0.0).

```
ALTER TABLE details DISTRIBUTED BY HASH(user_id) DEFAULT BUCKETS 10;
```

> **УВЕДОМЛЕНИЕ**
>
> Вы не можете указать одновременно `partition_names` и `DEFAULT`.

- Изменить количество buckets для указанных разделов на 15 с автоматически установленного Selena.

```
ALTER TABLE details PARTITIONS (p20231127, p20231128) DISTRIBUTED BY
HASH(user_id) BUCKETS 15 ;
```

> **ПРИМЕЧАНИЕ**
>
> Имена разделов можно просмотреть, выполнив `SHOW PARTITIONS FROM <table_name>;`.

Изменить как метод bucketing, так и количество buckets

> **УВЕДОМЛЕНИЕ**
>
> Изменение применяется ко всем разделам в таблице и не может быть применено только к конкретным разделам.

- Изменить метод bucketing с hash bucketing на random bucketing и изменить количество buckets на 10 с автоматически установленного Selena.

```
ALTER TABLE details DISTRIBUTED BY RANDOM BUCKETS 10;
```

- Изменить ключ для hash bucketing и изменить количество buckets на 10 с автоматически установленного Selena. Ключ, используемый для hashing bucketing, изменяется на `user_id, event_time` с исходного `event_time, event_type`. Количество buckets изменяется на 10 с автоматически установленного Selena.

```
ALTER TABLE details DISTRIBUTED BY HASH(user_id, event_time) BUCKETS
10;
``
```

Modify columns (add/delete columns, change column order, modify column comment)

Добавить столбец в указанное место указанного индекса

Синтаксис:

```
ALTER TABLE [<db_name>.]<tbl_name>
ADD COLUMN column_name column_type [KEY | agg_type] [DEFAULT
"default_value"]
[AFTER column_name|FIRST]
[TO rollup_index_name]
[PROPERTIES ("key"="value", ...)]
```

Примечание:

1. Если вы добавляете столбец значения в таблицу Aggregate, вам нужно указать agg_type.

2. Если вы добавляете ключевой столбец в таблицу, отличную от Aggregate (например, таблицу Duplicate Key), вам нужно указать ключевое слово KEY.
3. Вы не можете добавить столбец, который уже существует в базовом индексе, в rollup. (Вы можете пересоздать rollup при необходимости.)

Добавить несколько столбцов в указанный индекс

Синтаксис:

- Добавить несколько столбцов

```
ALTER TABLE [<db_name>.]<tbl_name>
ADD COLUMN (column_name1 column_type [KEY | agg_type] DEFAULT
"default_value", ...)
[TO rollup_index_name]
[PROPERTIES ("key"="value", ...)]
```

- Добавить несколько столбцов и использовать AFTER для указания расположения добавленных столбцов

```
ALTER TABLE [<db_name>.]<tbl_name>
ADD COLUMN column_name1 column_type [KEY | agg_type] DEFAULT
"default_value" AFTER column_name,
ADD COLUMN column_name2 column_type [KEY | agg_type] DEFAULT
"default_value" AFTER column_name
[, ...]
[TO rollup_index_name]
[PROPERTIES ("key"="value", ...)]
```

Примечание:

1. Если вы добавляете столбец значения в таблицу Aggregate, вам нужно указать `agg_type`.

2. Если вы добавляете ключевой столбец в таблицу, отличную от Aggregate, вам нужно указать ключевое слово KEY.

3. Вы не можете добавить столбец, который уже существует в базовом индексе, в rollup. (Вы можете создать другой rollup при необходимости.)

Добавить generated столбец (from v1.5.2)

Синтаксис:

```
ALTER TABLE [<db_name>.]<tbl_name>
ADD COLUMN col_name data_type [NULL] AS generation_expr [COMMENT 'string']
```

Вы можете добавить generated столбец и указать его выражение. Generated столбец может использоваться для предварительного вычисления и хранения результатов выражений, что значительно ускоряет запросы с одинаковыми сложными выражениями. Начиная с версии v1.5.2, Selena поддерживает generated столбцы.

Удалить столбец из указанного индекса

Синтаксис:

```
ALTER TABLE [<db_name>.]<tbl_name>
DROP COLUMN column_name
[FROM rollup_index_name];
```

Примечание:

1. Вы не можете удалить столбец раздела.
2. Если столбец удален из базового индекса, он также будет удален, если он включен в rollup.

Изменить тип столбца, позицию, комментарий и другие свойства

Синтаксис:

```
ALTER TABLE [<db_name>.]<tbl_name>
MODIFY COLUMN <column_name>
[ column_type [ KEY | agg_type ] ] [ NULL | NOT NULL ]
[ DEFAULT "<default_value>"] [ COMMENT "<new_column_comment>" ]
[ AFTER <column_name> | FIRST ]
[ FROM rollup_index_name ]
[ PROPERTIES ("key"="value", ...) ]
```

Примечание:

1. Если вы изменяете столбец значения в агрегатных моделях, вам нужно указать agg_type.

2. Если вы изменяете ключевой столбец в неагрегатных моделях, вам нужно указать ключевое слово KEY.

3. Может быть изменен только тип столбца. Другие свойства столбца остаются такими, какими они являются в настоящее время. (т.е. другие свойства должны быть явно написаны в операторе в соответствии с исходным свойством, см. пример 8 в части column).

4. Столбец раздела не может быть изменен.

5. В настоящее время поддерживаются следующие типы преобразований (точность потери гарантируется пользователем).

    - Преобразование TINYINT/SMALLINT/INT/BIGINT в TINYINT/SMALLINT/INT/BIGINT/DOUBLE.
    - Преобразование TINYINT/SMALLINT/INT/BIGINT/LARGEINT/FLOAT/DOUBLE/DECIMAL в VARCHAR. VARCHAR поддерживает изменение максимальной длины.
    - Преобразование VARCHAR в TINYINT/SMALLINT/INT/BIGINT/LARGEINT/FLOAT/DOUBLE.
    - Преобразование VARCHAR в DATE (в настоящее время поддерживается шесть форматов: "%Y-%m-%d", "%y-%m-%d", "%Y%m%d", "%y%m%d", "%Y/%m/%d, "%y/%m/%d")
    - Преобразование DATETIME в DATE (сохраняется только информация о годе-месяце-дне, т.е. `2019-12-09 21:47:05` `<-->` `2019-12-09`)
    - Преобразование DATE в DATETIME (часы, минуты, секунды устанавливаются в ноль, например: `2019-12-09` `<-->` `2019-12-09 00:00:00`)
    - Преобразование FLOAT в DOUBLE
    - Преобразование INT в DATE (Если данные INT не удается преобразовать, исходные данные остаются прежними)
6. Преобразование из NULL в NOT NULL не поддерживается.

7. Вы можете изменить несколько свойств в одном предложении MODIFY COLUMN. Однако некоторые комбинации свойств не поддерживаются.

Изменить порядок столбцов указанного индекса

Синтаксис:

```
ALTER TABLE [<db_name>.]<tbl_name>
ORDER BY (column_name1, column_name2, ...)
[FROM rollup_index_name]
[PROPERTIES ("key"="value", ...)]
```

Примечание:

- Все столбцы в индексе должны быть написаны.
- Столбец значения указан после ключевого столбца.

Изменить ключ сортировки

Начиная с версии v1.5.2, ключи сортировки для таблиц с Primary Key могут быть изменены. v1.5.2 расширяет эту поддержку на таблицы Duplicate Key, таблицы Aggregate и таблицы Unique Key.

Ключи сортировки в таблицах Duplicate Key и таблицах Primary Key могут быть комбинацией любых столбцов сортировки. Ключи сортировки в таблицах Aggregate и таблицах Unique Key должны включать все ключевые столбцы, но порядок столбцов не обязательно должен быть таким же, как у ключевых столбцов.

Синтаксис:

```
ALTER TABLE [<db_name>.]<table_name>
[ order_desc ]

order_desc ::=
    ORDER BY <column_name> [, <column_name> ...]
```

Пример: изменить ключи сортировки в таблицах с Primary Key.

Например, исходная таблица - это таблица Primary Key, где ключ сортировки и первичный ключ связаны, что составляет `dt, order_id`.

```
create table orders (
    dt date NOT NULL,
    order_id bigint NOT NULL,
    user_id int NOT NULL,
    merchant_id int NOT NULL,
    good_id int NOT NULL,
    good_name string NOT NULL,
    price int NOT NULL,
    cnt int NOT NULL,
    revenue int NOT NULL,
    state tinyint NOT NULL
) PRIMARY KEY (dt, order_id)
PARTITION BY date_trunc('day', dt)
DISTRIBUTED BY HASH(order_id);
```

Отделить ключ сортировки от первичного ключа и изменить ключ сортировки на `dt, revenue, state`.

```
ALTER TABLE orders ORDER BY (dt, revenue, state);
```

Изменить столбец STRUCT для добавления или удаления поля

> *Beta* **BETA FEATURE**
>
> Рекомендации по использованию бета-функций

Начиная с v1.5.2, Selena поддерживает изменение столбца STRUCT для добавления или удаления поля, которое может быть вложенным или находиться внутри типа ARRAY.

Синтаксис:

```
-- Добавить поле
ALTER TABLE [<db_name>.]<tbl_name> MODIFY COLUMN <column_name>
ADD FIELD field_path field_desc

-- Удалить поле
ALTER TABLE [<db_name>.]<tbl_name> MODIFY COLUMN <column_name>
DROP FIELD field_path

field_path ::= [ { <field_name>. | [*]. } [ ... ] ]<field_name>

  -- Обратите внимание, что здесь `[*]` в целом является предопределенным
символом и представляет все элементы в поле ARRAY
  -- при добавлении или удалении поля в типе STRUCT, вложенном в тип ARRAY.
  -- Для получения подробной информации см. описание параметра и примеры
`field_path`.

field_desc ::= <field_type> [ AFTER <prior_field_name> | FIRST ]
```

Параметры:

- `field_path`: Поле для добавления или удаления. Это может быть простое имя поля, указывающее на поле верхнего уровня, например, `new_field_name`, или Column Access Path, который представляет вложенное поле, например, `lv1_k1.lv2_k2.[*].new_field_name`.
- `[*]`: Когда тип STRUCT вложен в тип ARRAY, `[*]` представляет все элементы в поле ARRAY. Используется для добавления или удаления поля во всех элементах STRUCT, вложенных в поле ARRAY.
- `prior_field_name`: Поле, предшествующее вновь добавленному полю. Используется в сочетании с ключевым словом AFTER для указания порядка нового поля. Вам не нужно указывать этот параметр, если используется ключевое слово FIRST, указывающее, что новое поле должно быть первым полем. Размерность `prior_field_name` определяется `field_path` (в частности, частью, предшествующей `new_field_name`, то есть `level1_k1.level2_k2.[*]`) и не нужно указывать явно.

Примеры `field_path`:

- Добавить или удалить подполе в поле STRUCT, вложенном в столбец STRUCT.

  Предположим, есть столбец `fx stuct<c1 int, c2 struct <v1 int, v2 int>>`. Синтаксис для добавления поля `v3` под `c2`:

```
ALTER TABLE tbl MODIFY COLUMN fx ADD FIELD c2.v3 INT
```

После операции столбец становится `fx stuct<c1 int, c2 struct <v1 int, v2 int, v3 int>>`.

- Добавить или удалить подполе в каждом поле STRUCT, вложенном в поле ARRAY.

  Предположим, есть столбец `fx struct<c1 int, c2 array<struct <v1 int, v2 int>>>`. Поле `c2` - это тип ARRAY, который содержит STRUCT с двумя полями `v1` и `v2`. Синтаксис для добавления поля `v3` во вложенный STRUCT:

```
ALTER TABLE tbl MODIFY COLUMN fx ADD FIELD c2.[*].v3 INT
```

  После операции столбец становится `fx struct<c1 int, c2 array<struct <v1 int, v2 int, v3 int>>>`.

Для получения дополнительных инструкций по использованию см. Пример - Столбец -14.

> **ПРИМЕЧАНИЕ**
>
> - В настоящее время эта функция поддерживается только в shared-nothing кластерах.
> - Таблица должна иметь включенное свойство `fast_schema_evolution`.
> - Изменение типа Value подполя MAP в типе STRUCT не поддерживается, независимо от того, является ли тип Value ARRAY, STRUCT или MAP.
> - Вновь добавленные поля не могут иметь значения по умолчанию или атрибуты, такие как Nullable. Они по умолчанию являются Nullable со значением по умолчанию null.
> - После использования этой функции прямое понижение версии кластера до версии, которая не поддерживает эту функцию, не разрешается.

Modify rollup

Создать rollup

Синтаксис:

```
ALTER TABLE [<db_name>.]<tbl_name>
ADD ROLLUP rollup_name (column_name1, column_name2, ...)
[FROM from_index_name]
[PROPERTIES ("key"="value", ...)]
```

PROPERTIES: Поддержка установки timeout time, время ожидания по умолчанию составляет один день.

Пример:

```
ALTER TABLE [<db_name>.]<tbl_name>
ADD ROLLUP r1(col1,col2) from r0;
```

Создать rollups пакетно

Синтаксис:

```
ALTER TABLE [<db_name>.]<tbl_name>
ADD ROLLUP [rollup_name (column_name1, column_name2, ...)
[FROM from_index_name]
[PROPERTIES ("key"="value", ...)],...];
```

Пример:

```
ALTER TABLE [<db_name>.]<tbl_name>
ADD ROLLUP r1(col1,col2) from r0, r2(col3,col4) from r0;
```

Примечание:

1. Если from_index_name не указан, создается из базового индекса по умолчанию.
2. Столбцы в таблице rollup должны быть существующими столбцами в from_index.
3. В свойствах пользователь может указать формат хранения. Подробности см. в CREATE TABLE.

Удалить rollup

Синтаксис:

```
ALTER TABLE [<db_name>.]<tbl_name>
DROP ROLLUP rollup_name [PROPERTIES ("key"="value", ...)];
```

Пример:

```
ALTER TABLE [<db_name>.]<tbl_name> DROP ROLLUP r1;
```

Пакетное удаление rollups

Синтаксис:

```
ALTER TABLE [<db_name>.]<tbl_name>
DROP ROLLUP [rollup_name [PROPERTIES ("key"="value", ...)],...];
```

Пример:

```
ALTER TABLE [<db_name>.]<tbl_name> DROP ROLLUP r1, r2;
```

Примечание: Вы не можете удалить базовый индекс.

Modify indexes

Вы можете создавать или удалять следующие индексы:

- Bitmap индекс
- N-Gram bloom filter индекс
- Full-Text инвертированный индекс
- Vector индекс

Создать индекс

Синтаксис:

```
ALTER TABLE [<db_name>.]<tbl_name>
ADD INDEX index_name (column) [USING { BITMAP | NGRAMBF | GIN | VECTOR } ]
[COMMENT '<comment>']
```

Для получения подробных инструкций и примеров по созданию этих индексов см. соответствующие руководства, перечисленные выше.

Удалить индекс

Синтаксис:

```
ALTER TABLE [<db_name>.]<tbl_name>
DROP INDEX index_name;
```

Modify table properties

Синтаксис:

```
ALTER TABLE [<db_name>.]<tbl_name>
SET ("key" = "value")
```

В настоящее время Selena поддерживает изменение следующих свойств таблицы:

- `replication_num`
- `default.replication_num`
- `default.storage_medium`
- Свойства, связанные с динамическим разделением
- `enable_persistent_index`
- `bloom_filter_columns`
- `colocate_with`
- `bucket_size` (поддерживается с 3.2)
- `base_compaction_forbidden_time_ranges` (поддерживается с v1.5.2)

> **ПРИМЕЧАНИЕ**
>
> - В большинстве случаев вам разрешено изменять только одно свойство за раз. Вы можете изменять несколько свойств одновременно только в том случае, если эти свойства имеют одинаковый префикс. В настоящее время поддерживаются только `dynamic_partition.` и `binlog.`.
> - Вы также можете изменить свойства, объединив их в вышеуказанную операцию со столбцом. См. следующие примеры.

Swap

Swap поддерживает атомарный обмен двух таблиц.

Синтаксис:

```
ALTER TABLE [<db_name>.]<tbl_name>
SWAP WITH <tbl_name>;
```

ПРИМЕЧАНИЕ

- Ограничения Unique Key и Foreign Key между таблицами OLAP будут проверяться во время Swap, чтобы убедиться, что ограничения двух обмениваемых таблиц согласованы. Ошибка будет возвращена, если обнаружены несоответствия. Если несоответствия не обнаружены, ограничения Unique Key и Foreign Key будут автоматически обменяны.
- Материализованные представления, которые зависят от обмениваемых таблиц, будут автоматически установлены в неактивное состояние, и их ограничения Unique Key и Foreign Key будут удалены и больше не будут доступны.

Manual compaction (from 3.1)

Selena использует механизм сжатия для объединения различных версий загруженных данных. Эта функция может объединять небольшие файлы в большие файлы, что эффективно повышает производительность запросов.

До версии v1.5.2 сжатие выполнялось двумя способами:

- Автоматическое сжатие системой: Сжатие выполняется на уровне BE в фоновом режиме. Пользователи не могут указать базу данных или таблицу для сжатия.
- Пользователи могут выполнять сжатие, вызывая HTTP-интерфейс.

Начиная с версии v1.5.2, Selena предлагает SQL-интерфейс для пользователей для ручного выполнения сжатия путем выполнения SQL-команд. Они могут выбрать конкретную таблицу или раздел для сжатия. Это обеспечивает большую гибкость и контроль над процессом сжатия.

Shared-data кластеры поддерживают эту функцию начиная с v1.5.2.

> **ПРИМЕЧАНИЕ**
>
> Начиная с версии v1.5.2, вы можете запретить Base Compaction в определенном временном диапазоне, используя свойство `base_compaction_forbidden_time_ranges`.

Синтаксис:

```
ALTER TABLE <tbl_name> [ BASE | CUMULATIVE ] COMPACT [ <partition_name> | (
<partition1_name> [, <partition2_name> ...] ) ]
```

То есть:

```
-- Выполнить сжатие всей таблицы.
ALTER TABLE <tbl_name> COMPACT

-- Выполнить сжатие одного раздела.
ALTER TABLE <tbl_name> COMPACT <partition_name>

-- Выполнить сжатие нескольких разделов.
ALTER TABLE <tbl_name> COMPACT (<partition1_name>[,<partition2_name>,...])

-- Выполнить cumulative сжатие.
ALTER TABLE <tbl_name> CUMULATIVE COMPACT (<partition1_name>[,
<partition2_name>,...])

-- Выполнить base сжатие.
ALTER TABLE <tbl_name> BASE COMPACT (<partition1_name>[,
<partition2_name>,...])
```

Таблица `be_compactions` в базе данных `information_schema` записывает результаты сжатия. Вы можете выполнить `SELECT * FROM information_schema.be_compactions;` для запроса версий данных после сжатия.

Drop Primary Key Persistent Index (From 3.3.9)

Синтаксис:

```
ALTER TABLE [<db_name>.]<tbl_name>
DROP PERSISTENT INDEX ON TABLETS(<tablet_id>[, <tablet_id>, ...]);
```

> **ПРИМЕЧАНИЕ**
>
> Selena поддерживает только удаление persistent индексов для облачных таблиц Primary Key в shared-data кластерах.

Примеры

Таблица

1. Изменить количество реплик по умолчанию для таблицы, которое используется как количество реплик по умолчанию для вновь добавляемых разделов.

```
ALTER TABLE example_db.my_table
SET ("default.replication_num" = "2");
```

2. Изменить фактическое количество реплик таблицы с одним разделом.

```
ALTER TABLE example_db.my_table
SET ("replication_num" = "3");
```

3. Изменить режим записи данных и репликации между репликами.

```
ALTER TABLE example_db.my_table
SET ("replicated_storage" = "false");
```

   Этот пример устанавливает режим записи данных и репликации между репликами на "leaderless replication", что означает, что данные записываются в несколько реплик одновременно без различения первичных и вторичных реплик. Для получения дополнительной информации см. параметр `replicated_storage` в CREATE TABLE.

Раздел

1. Добавить раздел и использовать метод bucketing по умолчанию. Существующий раздел - [MIN, 2013-01-01). Добавленный раздел - [2013-01-01, 2014-01-01).

```
ALTER TABLE example_db.my_table
ADD PARTITION p1 VALUES LESS THAN ("2014-01-01");
```

2. Добавить раздел и использовать новое количество buckets.

```
ALTER TABLE example_db.my_table
ADD PARTITION p1 VALUES LESS THAN ("2015-01-01")
DISTRIBUTED BY HASH(k1);
```

3. Добавить раздел и использовать новое количество реплик.

```
ALTER TABLE example_db.my_table
ADD PARTITION p1 VALUES LESS THAN ("2015-01-01")
("replication_num"="1");
```

4. Изменить количество реплик для раздела.

```
ALTER TABLE example_db.my_table
MODIFY PARTITION p1 SET("replication_num"="1");
```

5. Пакетное изменение количества реплик для указанных разделов.

```
ALTER TABLE example_db.my_table
MODIFY PARTITION (p1, p2, p4) SET("replication_num"="1");
```

6. Пакетное изменение носителя хранения всех разделов.

```
ALTER TABLE example_db.my_table
MODIFY PARTITION (*) SET("storage_medium"="HDD");
```

7. Удалить раздел.

```
ALTER TABLE example_db.my_table
DROP PARTITION p1;
```

8. Добавить раздел с верхней и нижней границами.

```
ALTER TABLE example_db.my_table
ADD PARTITION p1 VALUES [("2014-01-01"), ("2014-02-01"));
```

Rollup

1. Создать rollup `example_rollup_index` на основе базового индекса (k1,k2,k3,v1,v2). Используется column-based хранилище.

```
ALTER TABLE example_db.my_table
ADD ROLLUP example_rollup_index(k1, k3, v1, v2)
PROPERTIES("storage_type"="column");
```

2. Создать rollup `example_rollup_index2` на основе `example_rollup_index(k1,k3,v1,v2)`.

```
ALTER TABLE example_db.my_table
ADD ROLLUP example_rollup_index2 (k1, v1)
FROM example_rollup_index;
```

3. Создать rollup `example_rollup_index3` на основе базового индекса (k1, k2, k3, v1). Время ожидания rollup установлено на один час.

```
ALTER TABLE example_db.my_table
ADD ROLLUP example_rollup_index3(k1, k3, v1)
PROPERTIES("storage_type"="column", "timeout" = "3600");
```

4. Удалить rollup `example_rollup_index2`.

```
ALTER TABLE example_db.my_table
DROP ROLLUP example_rollup_index2;
```

Столбец

1. Добавить ключевой столбец `new_col` (неагрегатный столбец) после столбца `col1` в `example_rollup_index`.

```
ALTER TABLE example_db.my_table
ADD COLUMN new_col INT KEY DEFAULT "0" AFTER col1
TO example_rollup_index;
```

2. Добавить столбец значения `new_col` (неагрегатный столбец) после столбца `col1` в `example_rollup_index`.

```
ALTER TABLE example_db.my_table
ADD COLUMN new_col INT DEFAULT "0" AFTER col1
TO example_rollup_index;
```

3. Добавить ключевой столбец `new_col` (агрегатный столбец) после столбца `col1` в `example_rollup_index`.

```
ALTER TABLE example_db.my_table
ADD COLUMN new_col INT DEFAULT "0" AFTER col1
TO example_rollup_index;
```

4. Добавить столбец значения `new_col SUM` (агрегатный столбец) после столбца `col1` в `example_rollup_index`.

```
ALTER TABLE example_db.my_table
ADD COLUMN new_col INT SUM DEFAULT "0" AFTER col1
TO example_rollup_index;
```

5. Добавить несколько столбцов в `example_rollup_index` (агрегатный).

```
ALTER TABLE example_db.my_table
ADD COLUMN (col1 INT DEFAULT "1", col2 FLOAT SUM DEFAULT "2.3")
TO example_rollup_index;
```

6. Добавить несколько столбцов в `example_rollup_index` (агрегатный) и указать расположение добавленных столбцов с помощью `AFTER`.

```
ALTER TABLE example_db.my_table
ADD COLUMN col1 INT DEFAULT "1" AFTER `k1`,
ADD COLUMN col2 FLOAT SUM AFTER `v2`,
TO example_rollup_index;
```

7. Удалить столбец из `example_rollup_index`.

```
ALTER TABLE example_db.my_table
DROP COLUMN col2
FROM example_rollup_index;
```

8. Изменить тип столбца col1 базового индекса на BIGINT и поместить его после `col2`.

```
ALTER TABLE example_db.my_table
MODIFY COLUMN col1 BIGINT DEFAULT "1" AFTER col2;
```

9. Изменить максимальную длину столбца `val1` базового индекса на 64. Исходная длина - 32.

```
ALTER TABLE example_db.my_table
MODIFY COLUMN val1 VARCHAR(64) REPLACE DEFAULT "abc";
```

10. Изменить порядок столбцов в `example_rollup_index`. Исходный порядок столбцов: k1, k2, k3, v1, v2.

```
ALTER TABLE example_db.my_table
ORDER BY (k3,k1,k2,v2,v1)
FROM example_rollup_index;
```

11. Выполнить две операции (ADD COLUMN и ORDER BY) одновременно.

```
ALTER TABLE example_db.my_table
ADD COLUMN v2 INT MAX DEFAULT "0" AFTER k2 TO example_rollup_index,
ORDER BY (k3,k1,k2,v2,v1) FROM example_rollup_index;
```

12. Изменить bloomfilter столбцы таблицы.

```
ALTER TABLE example_db.my_table
SET ("bloom_filter_columns"="k1,k2,k3");
```

Эта операция также может быть объединена с вышеуказанной операцией со столбцом (обратите внимание, что синтаксис нескольких предложений немного отличается).

```
ALTER TABLE example_db.my_table
DROP COLUMN col2
PROPERTIES ("bloom_filter_columns"="k1,k2,k3");
```

13. Изменить тип данных нескольких столбцов в одном операторе.

```
ALTER TABLE example_db.my_table
MODIFY COLUMN k1 VARCHAR(100) KEY NOT NULL,
MODIFY COLUMN v2 DOUBLE DEFAULT "1" AFTER v1;
```

14. Добавить и удалить поля в данных типа STRUCT.

Предварительные условия: Создать таблицу и вставить строку данных.

```
CREATE TABLE struct_test(
    c0 INT,
    c1 STRUCT<v1 INT, v2 STRUCT<v4 INT, v5 INT>, v3 INT>,
    c2 STRUCT<v1 INT, v2 ARRAY<STRUCT<v3 INT, v4 STRUCT<v5 INT, v6
INT>>>>
)
DUPLICATE KEY(c0)
DISTRIBUTED BY HASH(`c0`) BUCKETS 1
PROPERTIES (
    "fast_schema_evolution" = "true"
);
INSERT INTO struct_test VALUES (
    1,
    ROW(1, ROW(2, 3), 4),
    ROW(5, [ROW(6, ROW(7, 8)), ROW(9, ROW(10, 11))])
);
```

```
mysql> SELECT * FROM struct_test\G
*************************** 1. row ***************************
c0: 1
c1: {"v1":1,"v2":{"v4":2,"v5":3},"v3":4}
c2: {"v1":5,"v2":[{"v3":6,"v4":{"v5":7,"v6":8}},{"v3":9,"v4":
{"v5":10,"v6":11}}]}
```

- Добавить новое поле в столбец типа STRUCT.

```
ALTER TABLE struct_test MODIFY COLUMN c1 ADD FIELD v4 INT AFTER v2;
```

```
mysql> SELECT * FROM struct_test\G
*************************** 1. row ***************************
c0: 1
c1: {"v1":1,"v2":{"v4":2,"v5":3},"v4":null,"v3":4}
c2: {"v1":5,"v2":[{"v3":6,"v4":{"v5":7,"v6":8}},{"v3":9,"v4":
{"v5":10,"v6":11}}]}
```

- Добавить новое поле во вложенный тип STRUCT.

```
ALTER TABLE struct_test MODIFY COLUMN c1 ADD FIELD v2.v6 INT FIRST;
```

```
mysql> SELECT * FROM struct_test\G
*************************** 1. row ***************************
c0: 1
c1: {"v1":1,"v2":{"v6":null,"v4":2,"v5":3},"v4":null,"v3":4}
c2: {"v1":5,"v2":[{"v3":6,"v4":{"v5":7,"v6":8}},{"v3":9,"v4":
{"v5":10,"v6":11}}]}
```

- Добавить новое поле в тип STRUCT в массиве.

```
ALTER TABLE struct_test MODIFY COLUMN c2 ADD FIELD v2.[*].v7 INT AFTER
v3;
```

```
mysql> SELECT * FROM struct_test\G
*************************** 1. row ***************************
c0: 1
c1: {"v1":1,"v2":{"v6":null,"v4":2,"v5":3},"v4":null,"v3":4}
c2: {"v1":5,"v2":[{"v3":6,"v7":null,"v4":{"v5":7,"v6":8}},
{"v3":9,"v7":null,"v4":{"v5":10,"v6":11}}]}
```

- Удалить поле из столбца типа STRUCT.

```
ALTER TABLE struct_test MODIFY COLUMN c1 DROP FIELD v3;
```

```
mysql> SELECT * FROM struct_test\G
*************************** 1. row ***************************
c0: 1
c1: {"v1":1,"v2":{"v6":null,"v4":2,"v5":3},"v4":null}
c2: {"v1":5,"v2":[{"v3":6,"v7":null,"v4":{"v5":7,"v6":8}},
{"v3":9,"v7":null,"v4":{"v5":10,"v6":11}}]}
```

- Удалить поле из вложенного типа STRUCT.

```
ALTER TABLE struct_test MODIFY COLUMN c1 DROP FIELD v2.v4;
```

```
mysql> SELECT * FROM struct_test\G
*************************** 1. row ***************************
c0: 1
c1: {"v1":1,"v2":{"v6":null,"v5":3},"v4":null}
c2: {"v1":5,"v2":[{"v3":6,"v7":null,"v4":{"v5":7,"v6":8}},
{"v3":9,"v7":null,"v4":{"v5":10,"v6":11}}]}
```

- Удалить поле из типа STRUCT в массиве.

```
ALTER TABLE struct_test MODIFY COLUMN c2 DROP FIELD v2.[*].v3;
```

```
mysql> SELECT * FROM struct_test\G
*************************** 1. row ***************************
c0: 1
c1: {"v1":1,"v2":{"v6":null,"v5":3},"v4":null}
c2: {"v1":5,"v2":[{"v7":null,"v4":{"v5":7,"v6":8}},{"v7":null,"v4":
{"v5":10,"v6":11}}]}
```

Свойство таблицы

1. Изменить свойство Colocate таблицы.

```
ALTER TABLE example_db.my_table
SET ("colocate_with" = "t1");
```

2. Изменить свойство динамического раздела таблицы.

```
ALTER TABLE example_db.my_table
SET ("dynamic_partition.enable" = "false");
```

Если вам нужно добавить свойства динамического раздела в таблицу без настроенных свойств динамического раздела, вам нужно указать все свойства динамического раздела.

```
ALTER TABLE example_db.my_table
SET (
    "dynamic_partition.enable" = "true",
    "dynamic_partition.time_unit" = "DAY",
    "dynamic_partition.end" = "3",
    "dynamic_partition.prefix" = "p",
    "dynamic_partition.buckets" = "32"
    );
```

3. Изменить свойство носителя хранения таблицы.

```
ALTER TABLE example_db.my_table SET("default.storage_medium"="SSD");
```

Переименование

1. Переименовать `table1` в `table2`.

```
ALTER TABLE table1 RENAME table2;
```

2. Переименовать rollup `rollup1` таблицы `example_table` в `rollup2`.

```
ALTER TABLE example_table RENAME ROLLUP rollup1 rollup2;
```

3. Переименовать раздел `p1` таблицы `example_table` в `p2`.

```
ALTER TABLE example_table RENAME PARTITION p1 p2;
```

Индекс

1. Создать bitmap индекс для столбца `siteid` в `table1`.

```
ALTER TABLE table1
ADD INDEX index_1 (siteid) USING BITMAP COMMENT 'site_id_bitmap';
```

2. Удалить bitmap индекс столбца `siteid` в `table1`.

```
ALTER TABLE table1
DROP INDEX index_1;
```

Swap

Атомарный обмен между `table1` и `table2`.

```
ALTER TABLE table1 SWAP WITH table2
```

Ручное сжатие

```
CREATE TABLE compaction_test(
    event_day DATE,
    pv BIGINT)
DUPLICATE KEY(event_day)
PARTITION BY date_trunc('month', event_day)
DISTRIBUTED BY HASH(event_day) BUCKETS 8
PROPERTIES("replication_num" = "3");

INSERT INTO compaction_test VALUES
('2023-02-14', 2),
('2033-03-01',2);
{'label':'insert_734648fa-c878-11ed-90d6-00163e0dcbfc', 'status':'VISIBLE',
'txnId':'5008'}

INSERT INTO compaction_test VALUES
('2023-02-14', 2),('2033-03-01',2);
{'label':'insert_85c95c1b-c878-11ed-90d6-00163e0dcbfc', 'status':'VISIBLE',
'txnId':'5009'}

ALTER TABLE compaction_test COMPACT;

ALTER TABLE compaction_test COMPACT p203303;

ALTER TABLE compaction_test COMPACT (p202302,p203303);

ALTER TABLE compaction_test CUMULATIVE COMPACT (p202302,p203303);

ALTER TABLE compaction_test BASE COMPACT (p202302,p203303);
```

Удаление Primary Key Persistent Index

Удалить persistent индекс на tablets `100` и `101` для таблицы Primary Key `db1.test_tbl` в shared-data кластере.

```
ALTER TABLE db1.test_tbl DROP PERSISTENT INDEX ON TABLETS (100, 101);
```

Ссылки

- CREATE TABLE
- SHOW CREATE TABLE
- SHOW TABLES
- SHOW ALTER TABLE
- DROP TABLE

CANCEL ALTER TABLE

CANCEL ALTER TABLE отменяет выполнение текущей операции ALTER TABLE, включая:

- Изменение столбцов.
- Оптимизация схемы таблицы (с v1.5.2), включая изменение метода распределения по bucket'ам и количества bucket'ов.
- Создание и удаление rollup индекса.

> **ПРИМЕЧАНИЕ**
>
> - Эта команда является синхронной операцией.
> - Для выполнения этой команды требуется привилегия `ALTER_PRIV` на таблице.
> - Эта команда поддерживает отмену только асинхронных операций с использованием ALTER TABLE (как указано выше) и не поддерживает отмену синхронных операций с использованием ALTER TABLE, таких как rename.

Синтаксис

```
CANCEL ALTER TABLE { COLUMN | OPTIMIZE | ROLLUP } FROM [db_name.]table_name
```

Параметры

- `{COLUMN | OPTIMIZE | ROLLUP}`

  - Если указано `COLUMN`, эта команда отменяет операции изменения столбцов.
  - Если указано `OPTIMIZE`, эта команда отменяет операции оптимизации схемы таблицы.
  - Если указано `ROLLUP`, эта команда отменяет операции добавления или удаления rollup индекса.
- `db_name`: опционально. Имя базы данных, к которой принадлежит таблица. Если этот параметр не указан, по умолчанию используется ваша текущая база данных.

- `table_name`: обязательно. Имя таблицы.

Примеры

1. Отменить операцию изменения столбцов для `example_table` в базе данных `example_db`.

```
CANCEL ALTER TABLE COLUMN FROM example_db.example_table;
```

2. Отменить операцию оптимизации схемы таблицы для `example_table` в базе данных `example_db`.

```
CANCEL ALTER TABLE OPTIMIZE FROM example_db.example_table;
```

3. Отменить операцию добавления или удаления rollup индекса для `example_table` в текущей базе данных.

```
CANCEL ALTER TABLE ROLLUP FROM example_table;
```

CREATE INDEX

Создает индексы.

Вы можете создать следующие индексы:

- Bitmap индекс
- N-Gram bloom filter индекс
- Full-Text inverted индекс
- Vector индекс

Для подробных инструкций и примеров создания этих индексов смотрите соответствующие руководства, перечисленные выше.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия ALTER на целевой таблице. Вы можете следовать инструкциям в GRANT, чтобы предоставить эту привилегию.

Синтаксис

```
CREATE INDEX index_name ON table_name (column_name)
[USING { BITMAP | NGRAMBF | GIN | VECTOR } ]
[(index_property)]
[COMMENT '<comment>']
```

Параметры

Параметр	Обязательно	Описание
index_name	Да	Имя индекса. Правила именования смотрите в Системных ограничениях.
table_name	Да	Имя таблицы.
column_name	Да	Имя столбца для построения индекса. Столбец может иметь только один индекс. Если у столбца уже есть индекс, вы не можете создать для него ещё один индекс.
USING	Нет	Тип создаваемого индекса. Допустимые значения: • BITMAP (По умолчанию) • NGRAMBF • GIN • VECTOR
index_property	Нет	Свойства создаваемого индекса. Для `NGRAMBF`, `GIN` и `VECTOR` необходимо указать соответствующие свойства. Подробные инструкции смотрите в соответствующих руководствах.
COMMENT	Нет	Комментарий к индексу.

Примеры

Создайте таблицу `sales_records` следующим образом:

```
CREATE TABLE sales_records
(
    record_id int,
    seller_id int,
    item_id int
)
DISTRIBUTED BY hash(record_id)
PROPERTIES (
    "replication_num" = "3"
);
```

Создайте bitmap индекс `index` на столбце `item_id` таблицы `sales_records`.

```
CREATE INDEX index ON sales_records (item_id) USING BITMAP COMMENT '';
```

Связанные SQL-команды

- SHOW INDEX
- DROP INDEX

CREATE TABLE

Создает новую таблицу в Selena.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия CREATE TABLE на целевой базе данных.

Синтаксис

```
CREATE [EXTERNAL] [TEMPORARY] TABLE [IF NOT EXISTS] [database.]table_name
(column_definition1[, column_definition2, ...]
[, index_definition1[, index_definition12,]])
[ENGINE = [olap|mysql|elasticsearch|hive|hudi|iceberg|jdbc]]
[key_desc]
[COMMENT "table comment"]
[partition_desc]
[distribution_desc]
[rollup_index]
[ORDER BY (column_name1,...)]
[PROPERTIES ("key"="value", ...)]
```

ПОДСКАЗКА

- Имя таблицы, имя partition, имя столбца и имя индекса, которые вы создаете, должны соответствовать соглашениям об именовании в Системных ограничениях.
- При указании имени базы данных, имени таблицы, имени столбца или имени partition обратите внимание, что некоторые литералы используются как зарезервированные ключевые слова в Selena. Не используйте эти ключевые слова напрямую в SQL-выражениях. Если вы хотите использовать такое ключевое слово в SQL-выражении, заключите его в пару обратных кавычек (`). См. Ключевые слова для этих зарезервированных ключевых слов.

Ключевые слова

`EXTERNAL`

ПРЕДУПРЕЖДЕНИЕ

Ключевое слово `EXTERNAL` устарело.

Мы рекомендуем использовать внешние каталоги для запроса данных из источников данных Hive, Iceberg, Hudi и JDBC вместо использования ключевого слова `EXTERNAL` для создания внешних таблиц.

ПОДСКАЗКА

Рекомендация

Начиная с v1.5.2, Selena поддерживает создание таблиц в формате Parquet в каталогах Iceberg и поддерживает загрузку данных в эти таблицы Iceberg в формате Parquet с помощью INSERT INTO.

Начиная с v1.5.2, Selena поддерживает создание таблиц в формате Parquet в каталогах Hive и поддерживает загрузку данных в эти таблицы Hive в формате Parquet с помощью INSERT INTO. Начиная с v1.5.2, Selena поддерживает создание таблиц в форматах ORC и Textfile в каталогах Hive и поддерживает загрузку данных в эти таблицы Hive в форматах ORC и Textfile с помощью INSERT INTO.

Если вы хотите использовать устаревшее ключевое слово `EXTERNAL`, разверните **Детали ключевого слова `EXTERNAL`**

▸ Детали ключевого слова `EXTERNAL`

`TEMPORARY`

Создает временную таблицу. Начиная с v1.5.2, Selena поддерживает создание временных таблиц в Default Catalog. Для получения дополнительной информации см. Временная таблица.

ПРИМЕЧАНИЕ

При создании временной таблицы необходимо установить `ENGINE` в `olap`.

Определение столбцов

```
col_name col_type [agg_type] [NULL | NOT NULL] [DEFAULT "default_value"]
[AUTO_INCREMENT] [AS generation_expr]
```

col_name

Обратите внимание, что обычно вы не можете создать столбец, имя которого начинается с `__op` или `__row`, потому что эти форматы имен зарезервированы для специальных

целей в Selena, и создание таких столбцов может привести к неопределенному поведению. Если вам действительно нужно создать такой столбец, установите динамический параметр FE `allow_system_reserved_names` в `TRUE`.

col_type

Конкретная информация о столбце, такая как типы и диапазоны:

- TINYINT (1 байт): Диапазон от -2^7 + 1 до 2^7 - 1.
- SMALLINT (2 байта): Диапазон от -2^15 + 1 до 2^15 - 1.
- INT (4 байта): Диапазон от -2^31 + 1 до 2^31 - 1.
- BIGINT (8 байт): Диапазон от -2^63 + 1 до 2^63 - 1.
- LARGEINT (16 байт): Диапазон от -2^127 + 1 до 2^127 - 1.
- FLOAT (4 байта): Поддерживает научную нотацию.
- DOUBLE (8 байт): Поддерживает научную нотацию.
- DECIMAL[(precision, scale)] (16 байт)
    - Значение по умолчанию: DECIMAL(10, 0)
    - precision: 1 ~ 38
    - scale: 0 ~ precision
    - Целая часть: precision - scale

      Научная нотация не поддерживается.
- DATE (3 байта): Диапазон от 0000-01-01 до 9999-12-31.
- DATETIME (8 байт): Диапазон от 0000-01-01 00:00:00 до 9999-12-31 23:59:59.
- CHAR[(length)]: Строка фиксированной длины. Диапазон: 1 ~ 255. Значение по умолчанию: 1.

- VARCHAR[(length)]: Строка переменной длины. Значение по умолчанию — 1. Единица измерения: байты. В версиях до Selena 2.1 диапазон значений `length` составляет 1–65533. [Предварительная версия] В Selena 2.1 и более поздних версиях диапазон значений `length` составляет 1–1048576.

- HLL (1~16385 байт): Для типа HLL нет необходимости указывать длину или значение по умолчанию. Длина будет контролироваться внутри системы в соответствии с агрегацией данных. Столбец HLL может быть запрошен или использован только с помощью hll_union_agg, Hll_cardinality и hll_hash.

- BITMAP: Тип Bitmap не требует указания длины или значения по умолчанию. Он представляет собой набор беззнаковых чисел bigint. Самый большой элемент может достигать 2^64 - 1.

agg_type

Тип агрегации. Если не указан, этот столбец является ключевым столбцом. Если указан, это столбец значений. Поддерживаются следующие типы агрегации:

- `SUM`, `MAX`, `MIN`, `REPLACE`
- `HLL_UNION` (только для типа `HLL`)
- `BITMAP_UNION` (только для `BITMAP`)
- `REPLACE_IF_NOT_NULL`: Это означает, что импортированные данные будут заменены только в том случае, если они имеют ненулевое значение. Если это нулевое значение, Selena сохранит исходное значение.

> **ПРИМЕЧАНИЕ**
>
> - Когда столбец типа агрегации BITMAP_UNION импортируется, его исходные типы данных должны быть TINYINT, SMALLINT, INT и BIGINT.
> - Если NOT NULL указан для столбца REPLACE_IF_NOT_NULL при создании таблицы, Selena все равно преобразует данные в NULL, не отправляя сообщение об ошибке пользователю. Это позволяет пользователю импортировать выбранные столбцы.

Этот тип агрегации применяется ТОЛЬКО к таблице Aggregate, у которой тип key_desc — AGGREGATE KEY. Начиная с v1.5.2, `REPLACE_IF_NOT_NULL` поддерживает столбцы типа BITMAP.

NULL | NOT NULL: Разрешено ли столбцу быть `NULL`. По умолчанию `NULL` указывается для всех столбцов в таблице, использующей ключ Duplicate Key, Aggregate или Unique Key. В таблице, использующей ключ Primary Key, по умолчанию столбцы значений указываются с `NULL`, тогда как ключевые столбцы указываются с `NOT NULL`. Если в исходных данных содержатся значения `NULL`, представьте их с помощью `\N`. Selena обрабатывает `\N` как `NULL` во время загрузки данных.

DEFAULT "default_value": значение по умолчанию для столбца. Когда вы загружаете данные в Selena, если исходное поле, сопоставленное со столбцом, пустое, Selena автоматически заполняет значение по умолчанию в столбце. Вы можете указать значение по умолчанию одним из следующих способов:

- **DEFAULT current_timestamp**: Используйте текущее время в качестве значения по умолчанию. Для получения дополнительной информации см. current_timestamp().
- **DEFAULT `<default_value>`**: Используйте заданное значение типа данных столбца в качестве значения по умолчанию. Например, если тип данных столбца — VARCHAR, вы можете указать строку VARCHAR, например beijing, в качестве значения по умолчанию, как показано в `DEFAULT "beijing"`. Обратите внимание, что значения по умолчанию не могут быть любого из следующих типов: ARRAY, BITMAP, JSON, HLL и BOOLEAN.
- **DEFAULT (<expr>)**: Используйте результат, возвращаемый заданной функцией, в качестве значения по умолчанию. Поддерживаются только выражения uuid() и uuid_numeric().

AUTO_INCREMENT: указывает столбец `AUTO_INCREMENT`. Типы данных столбцов `AUTO_INCREMENT` должны быть BIGINT. Автоинкрементные идентификаторы начинаются с 1 и увеличиваются с шагом 1. Для получения дополнительной информации о столбцах `AUTO_INCREMENT` см. AUTO_INCREMENT. Начиная с v1.5.2, Selena поддерживает столбцы `AUTO_INCREMENT`.

AS generation_expr: указывает вычисляемый столбец и его выражение. Вычисляемый столбец может использоваться для предварительного вычисления и хранения результатов выражений, что значительно ускоряет запросы с одинаковыми сложными выражениями. Начиная с v1.5.2, Selena поддерживает вычисляемые столбцы.

Определение индекса

```
INDEX index_name (col_name[, col_name, ...]) [USING BITMAP] COMMENT
'xxxxxx'
```

Для получения дополнительной информации об описаниях параметров и примечаниях по использованию см. Bitmap индексирование.

ENGINE

Значение по умолчанию: `olap`. Если этот параметр не указан, по умолчанию создается таблица OLAP (собственная таблица Selena).

Опциональные значения: `mysql`, `elasticsearch`, `hive`, `jdbc`, `iceberg` и `hudi`.

Ключ

Синтаксис:

```
key_type(k1[,k2 ...])
```

Данные упорядочиваются в указанных ключевых столбцах и имеют разные атрибуты для разных типов ключей:

- AGGREGATE KEY: Идентичное содержимое в ключевых столбцах будет агрегировано в столбцы значений в соответствии с указанным типом агрегации. Обычно применяется в бизнес-сценариях, таких как финансовые отчеты и многомерный анализ.

- UNIQUE KEY/PRIMARY KEY: Идентичное содержимое в ключевых столбцах будет заменено в столбцах значений в соответствии с последовательностью импорта. Может применяться для добавления, удаления, изменения и запроса по ключевым столбцам.

- DUPLICATE KEY: Идентичное содержимое в ключевых столбцах, которое также существует в Selena одновременно. Может использоваться для хранения детальных данных или данных без атрибутов агрегации.

> **ПРИМЕЧАНИЕ**
>
> DUPLICATE KEY — это тип по умолчанию. Данные будут упорядочены в соответствии с ключевыми столбцами.

> **ПРИМЕЧАНИЕ**
>
> Столбцы значений не нужно указывать типы агрегации, когда другой key_type используется для создания таблиц, за исключением AGGREGATE KEY.

COMMENT

Вы можете добавить комментарий к таблице при её создании, опционально. Обратите внимание, что COMMENT должен быть размещён после `key_desc`. В противном случае таблица не может быть создана.

Начиная с v1.5.2, вы можете изменить комментарий к таблице с помощью `ALTER TABLE <table_name> COMMENT = "new table comment"`.

Partition

Partition могут управляться следующими способами:

Динамическое создание partition

Динамическое разбиение на partition предоставляет управление временем жизни (TTL) для partition. Selena автоматически создает новые partition заранее и удаляет устаревшие partition для обеспечения свежести данных. Чтобы включить эту функцию, вы можете настроить свойства, связанные с динамическим разбиением на partition, при создании таблицы.

Создание partition по одному

Указать только верхнюю границу для partition

Синтаксис:

```
PARTITION BY RANGE ( <partitioning_column1> [, <partitioning_column2>, ...
] )
  PARTITION <partition1_name> VALUES LESS THAN ("
<upper_bound_for_partitioning_column1>" [ , "
<upper_bound_for_partitioning_column2>", ... ] )
  [ ,
  PARTITION <partition2_name> VALUES LESS THAN ("
<upper_bound_for_partitioning_column1>" [ , "
<upper_bound_for_partitioning_column2>", ... ] )
  , ... ]
)
```

ПРИМЕЧАНИЕ

Пожалуйста, используйте указанные ключевые столбцы и указанные диапазоны значений для разбиения на partition.

- Для соглашений об именовании partition см. Системные ограничения.
- До v1.5.2 столбцы для разбиения на partition по диапазону поддерживают только следующие типы: TINYINT, SMALLINT, INT, BIGINT, LARGEINT, DATE и DATETIME. Начиная с v1.5.2, три специальные функции времени могут использоваться в качестве столбцов для разбиения на partition по диапазону. Для подробного использования см. Распределение данных.
- Partition являются левозамкнутыми и правооткрытыми. Левая граница первого partition имеет минимальное значение.
- Значение NULL хранится только в partition, которые содержат минимальные значения. Когда partition, содержащий минимальное значение, удаляется, значения NULL больше не могут быть импортированы.
- Столбцы partition могут быть как одним, так и несколькими столбцами. Значения partition являются минимальными значениями по умолчанию.
- Когда в качестве столбца разбиения на partition указан только один столбец, вы можете установить `MAXVALUE` в качестве верхней границы для столбца разбиения на partition последнего partition.

```
PARTITION BY RANGE (pay_dt) (
  PARTITION p1 VALUES LESS THAN ("20210102"),
  PARTITION p2 VALUES LESS THAN ("20210103"),
  PARTITION p3 VALUES LESS THAN MAXVALUE
)
```

ПРИМЕЧАНИЕ

- Partition часто используются для управления данными, связанными со временем.
- Когда требуется откат данных, вы можете захотеть рассмотреть очистку первого partition для добавления partition позже, когда это необходимо.

Указать как нижнюю, так и верхнюю границы для partition

Синтаксис:

```
PARTITION BY RANGE ( <partitioning_column1> [, <partitioning_column2>, ...
] )
(
    PARTITION <partition_name1> VALUES [( "
<lower_bound_for_partitioning_column1>" [ , "
<lower_bound_for_partitioning_column2>", ... ] ), ( "
<upper_bound_for_partitioning_column1?" [ , "
<upper_bound_for_partitioning_column2>", ... ] ) )
    [,
    PARTITION <partition_name2> VALUES [( "
<lower_bound_for_partitioning_column1>" [ , "
<lower_bound_for_partitioning_column2>", ... ] ), ( "
<upper_bound_for_partitioning_column1>" [ , "
<upper_bound_for_partitioning_column2>", ... ] ) )
    , ...]
)
```

> **ПРИМЕЧАНИЕ**
>
> - Fixed Range более гибкий, чем LESS THAN. Вы можете настроить левую и правую partition.
> - Fixed Range такой же, как LESS THAN, в других аспектах.
> - Когда в качестве столбца разбиения на partition указан только один столбец, вы можете установить `MAXVALUE` в качестве верхней границы для столбца разбиения на partition последнего partition.

```
PARTITION BY RANGE (pay_dt) (
  PARTITION p202101 VALUES [("20210101"), ("20210201")),
  PARTITION p202102 VALUES [("20210201"), ("20210301")),
  PARTITION p202103 VALUES [("20210301"), (MAXVALUE))
)
```

Создание нескольких partition пакетно

Синтаксис

- Если столбец разбиения на partition имеет тип даты.

```
PARTITION BY RANGE (<partitioning_column>) (
    START ("<start_date>") END ("<end_date>") EVERY (INTERVAL <N>
<time_unit>)
)
```

- Если столбец разбиения на partition имеет целочисленный тип.

```
PARTITION BY RANGE (<partitioning_column>) (
    START ("<start_integer>") END ("<end_integer>") EVERY
(<partitioning_granularity>)
)
```

Описание

Вы можете указать начальные и конечные значения в `START()` и `END()` и единицу времени или гранулярность разбиения на partition в `EVERY()` для создания нескольких partition пакетно.

- До v1.5.2 столбцы для разбиения на partition по диапазону поддерживают только следующие типы: TINYINT, SMALLINT, INT, BIGINT, LARGEINT, DATE и DATETIME. Начиная с v1.5.2, три специальные функции времени могут использоваться в качестве столбцов для разбиения на partition по диапазону. Для подробного использования см. Распределение данных.
- Если столбец разбиения на partition имеет тип даты, вам нужно использовать ключевое слово `INTERVAL` для указания интервала времени. Вы можете указать единицу времени как hour (начиная с v1.5.2), day, week, month или year. Соглашения об именовании partition такие же, как и для динамических partition.

Для получения дополнительной информации см. Распределение данных.

Распределение

Selena поддерживает hash bucketing и random bucketing. Если вы не настроите bucketing, Selena использует random bucketing и автоматически устанавливает количество bucket'ов по умолчанию.

- Random bucketing (начиная с v1.5.2)

  Для данных в partition Selena распределяет данные случайным образом по всем bucket'ам, что не основано на конкретных значениях столбцов. И если вы хотите, чтобы Selena автоматически устанавливала количество bucket'ов, вам не нужно указывать какие-либо конфигурации bucketing. Если вы решите вручную указать количество bucket'ов, синтаксис следующий:

  ```
  DISTRIBUTED BY RANDOM BUCKETS <num>
  ```

  Однако обратите внимание, что производительность запросов, обеспечиваемая random bucketing, может быть не идеальной при запросе больших объемов данных и частом использовании определенных столбцов в качестве условных столбцов. В этом сценарии рекомендуется использовать hash bucketing. Поскольку нужно сканировать и вычислять только небольшое количество bucket'ов, это значительно улучшает производительность запросов.

  Меры предосторожности

- Вы можете использовать random bucketing только для создания таблиц Duplicate Key.
- Вы не можете указать Colocation Group для таблицы с random bucketing.
- Spark Load нельзя использовать для загрузки данных в таблицы с random bucketing.
- Начиная с Selena v1.5.2, вам не нужно устанавливать количество bucket'ов при создании таблицы. Selena автоматически устанавливает количество bucket'ов. Если вы хотите установить этот параметр, см. Установить количество bucket'ов.

Для получения дополнительной информации см. Random bucketing.

- Hash bucketing

  Синтаксис:

  ```
  DISTRIBUTED BY HASH (k1[,k2 ...]) [BUCKETS num]
  ```

  Данные в partition могут быть подразделены на bucket'ы на основе хеш-значений столбцов bucketing и количества bucket'ов. Мы рекомендуем выбирать столбец, который соответствует следующим двум требованиям, в качестве столбца bucketing.

  - Столбец с высокой кардинальностью, такой как ID
  - Столбец, который часто используется в качестве фильтра в запросах

  Если такого столбца не существует, вы можете определить столбец bucketing в соответствии со сложностью запросов.

  - Если запрос сложный, мы рекомендуем выбирать столбец с высокой кардинальностью в качестве столбца bucketing, чтобы обеспечить сбалансированное распределение данных между bucket'ами и улучшить использование ресурсов кластера.
  - Если запрос относительно простой, мы рекомендуем выбирать столбец, который часто используется в качестве условия запроса, в качестве столбца bucketing, чтобы повысить эффективность запросов.

  Если данные partition не могут быть равномерно распределены в каждый bucket с использованием одного столбца bucketing, вы можете выбрать несколько столбцов bucketing (максимум три). Для получения дополнительной информации см. Выбор столбцов bucketing.

Меры предосторожности:

- **При создании таблицы вы должны указать столбцы bucketing**.
- Значения столбцов bucketing не могут быть обновлены.
- Столбцы bucketing не могут быть изменены после их указания.
- Начиная с Selena v1.5.2, вам не нужно устанавливать количество bucket'ов при создании таблицы. Selena автоматически устанавливает количество bucket'ов. Если вы хотите установить этот параметр, см. Установить количество bucket'ов.

Rollup индекс

Вы можете создавать rollup'ы пакетно при создании таблицы.

Синтаксис:

```
ROLLUP (rollup_name (column_name1, column_name2, ...)
[FROM from_index_name]
[PROPERTIES ("key"="value", ...)],...)
```

ORDER BY

Начиная с v1.5.2, таблицы Primary Key поддерживают определение ключей сортировки с помощью `ORDER BY`. Начиная с v1.5.2, таблицы Duplicate Key, таблицы Aggregate и таблицы Unique Key поддерживают определение ключей сортировки с помощью `ORDER BY`.

Для получения дополнительных описаний ключей сортировки см. Ключи сортировки и префиксные индексы.

PROPERTIES

Хранилище и реплики

Если тип engine — `OLAP`, вы можете указать начальный носитель хранения (`storage_medium`), автоматическое время охлаждения хранилища (`storage_cooldown_time`) или временной интервал (`storage_cooldown_ttl`), и количество реплик (`replication_num`) при создании таблицы.

Область действия свойств: Если таблица имеет только один partition, свойства принадлежат таблице. Если таблица разделена на несколько partition, свойства

принадлежат каждому partition. И когда вам нужно настроить различные свойства для указанных partition, вы можете выполнить ALTER TABLE ... ADD PARTITION или ALTER TABLE ... MODIFY PARTITION после создания таблицы.

Установить начальный носитель хранения и автоматическое время охлаждения хранилища

```
PROPERTIES (
    "storage_medium" = "[SSD|HDD]",
    { "storage_cooldown_ttl" = "<num> { YEAR | MONTH | DAY | HOUR } "
    | "storage_cooldown_time" = "yyyy-MM-dd HH:mm:ss" }
)
```

Свойства

- `storage_medium`: начальный носитель хранения, который может быть установлен на `SSD` или `HDD`. Убедитесь, что тип носителя хранения, который вы явно указали, соответствует типам дисков BE для вашего кластера Selena, указанным в статическом параметре BE `storage_root_path`.

  Если параметр конфигурации FE `enable_strict_storage_medium_check` установлен в `true`, система строго проверяет тип диска BE при создании таблицы. Если носитель хранения, который вы указали в CREATE TABLE, не соответствует типу диска BE, возвращается ошибка "Failed to find enough host in all backends with storage medium is SSD|HDD." и создание таблицы завершается неудачей. Если `enable_strict_storage_medium_check` установлен в `false`, система игнорирует эту ошибку и принудительно создает таблицу. Однако дисковое пространство кластера может быть неравномерно распределено после загрузки данных.

  Начиная с v1.5.2, система автоматически определяет носитель хранения на основе типа диска BE, если `storage_medium` явно не указан.

  - Система автоматически устанавливает этот параметр в SSD в следующих сценариях:

    - Типы дисков, сообщаемые BE (`storage_root_path`), содержат только SSD.
    - Типы дисков, сообщаемые BE (`storage_root_path`), содержат как SSD, так и HDD. Обратите внимание, что начиная с v1.5.2, система устанавливает

`storage_medium` в SSD, когда `storage_root_path`, сообщаемые BE, содержат как SSD, так и HDD, и свойство `storage_cooldown_time` указано.

- Система автоматически устанавливает этот параметр в HDD в следующих сценариях:

  - Типы дисков, сообщаемые BE (`storage_root_path`), содержат только HDD.
  - Начиная с 2.3.10, 2.4.5, 2.5.4 и 3.0, система устанавливает `storage_medium` в HDD, когда `storage_root_path`, сообщаемые BE, содержат как SSD, так и HDD, и свойство `storage_cooldown_time` не указано.

- `storage_cooldown_ttl` или `storage_cooldown_time`: автоматическое время охлаждения хранилища или временной интервал. Автоматическое охлаждение хранилища относится к автоматической миграции данных с SSD на HDD. Эта функция эффективна только тогда, когда начальный носитель хранения — SSD.

  - `storage_cooldown_ttl`: **временной интервал** автоматического охлаждения хранилища для partition в этой таблице. Если вам нужно сохранить самые последние partition на SSD и автоматически охладить более старые partition на HDD после определенного временного интервала, вы можете использовать этот параметр. Автоматическое время охлаждения хранилища для каждого partition рассчитывается с использованием значения этого параметра плюс верхняя временная граница partition.

  Поддерживаемые значения: `<num> YEAR`, `<num> MONTH`, `<num> DAY` и `<num> HOUR`. `<num>` — неотрицательное целое число. Значение по умолчанию — null, что означает, что охлаждение хранилища не выполняется автоматически.

  Например, вы указываете значение как `"storage_cooldown_ttl"="1 DAY"` при создании таблицы, и существует partition `p20230801` с диапазоном `[2023-08-01 00:00:00,2023-08-02 00:00:00)`. Автоматическое время охлаждения хранилища для этого partition — `2023-08-03 00:00:00`, что составляет `2023-08-02 00:00:00 + 1 DAY`. Если вы указываете значение как `"storage_cooldown_ttl"="0 DAY"` при создании таблицы, автоматическое время охлаждения хранилища для этого partition — `2023-08-02 00:00:00`.

  - `storage_cooldown_time`: автоматическое время охлаждения хранилища (**абсолютное время**), когда таблица охлаждается с SSD на HDD. Указанное время должно быть позже текущего времени. Формат: "yyyy-MM-dd HH:mm:ss". Когда вам нужно настроить различные свойства для указанных partition, вы

можете выполнить ALTER TABLE ... ADD PARTITION или ALTER TABLE ... MODIFY PARTITION.

Использование

- Сравнение параметров, связанных с автоматическим охлаждением хранилища, выглядит следующим образом:

  - `storage_cooldown_ttl`: Свойство таблицы, которое указывает временной интервал автоматического охлаждения хранилища для partition в таблице. Система автоматически охлаждает partition в момент `значение этого параметра плюс верхняя временная граница partition`. Таким образом, автоматическое охлаждение хранилища выполняется на уровне partition, что более гибко.
  - `storage_cooldown_time`: Свойство таблицы, которое указывает автоматическое время охлаждения хранилища (**абсолютное время**) для этой таблицы. Кроме того, вы можете настроить различные свойства для указанных partition после создания таблицы.
  - `storage_cooldown_second`: Статический параметр FE, который указывает задержку автоматического охлаждения хранилища для всех таблиц в кластере.
- Свойство таблицы `storage_cooldown_ttl` или `storage_cooldown_time` имеет приоритет над статическим параметром FE `storage_cooldown_second`.

- При настройке этих параметров необходимо указать `"storage_medium = "SSD"`.

- Если вы не настроите эти параметры, автоматическое охлаждение хранилища не будет выполняться автоматически.

- Выполните `SHOW PARTITIONS FROM <table_name>` для просмотра автоматического времени охлаждения хранилища для каждого partition.

Ограничения

- Expression и List partitioning не поддерживаются.
- Столбец partition должен быть типа даты.
- Несколько столбцов partition не поддерживаются.
- Таблицы Primary Key не поддерживаются.

Установить количество реплик для каждого tablet в partition

`replication_num`: количество реплик для каждой таблицы в partition. Число по умолчанию: `3`.

```
PROPERTIES (
    "replication_num" = "<num>"
)
```

Bloom filter индексы

Если тип Engine — `olap`, вы можете указать столбец для применения bloom filter индексов.

Следующие ограничения применяются при использовании bloom filter индекса:

- Вы можете создавать bloom filter индексы для всех столбцов таблицы Duplicate Key или Primary Key. Для таблицы Aggregate или Unique Key вы можете создавать bloom filter индексы только для ключевых столбцов.
- Столбцы TINYINT, FLOAT, DOUBLE и DECIMAL не поддерживают создание bloom filter индексов.
- Bloom filter индексы могут только улучшить производительность запросов, содержащих операторы `in` и `=`, такие как `Select xxx from table where x in {}` и `Select xxx from table where column = xxx`. Более дискретные значения в этом столбце приведут к более точным запросам.

Для получения дополнительной информации см. Bloom filter индексирование

```
PROPERTIES (
    "bloom_filter_columns"="k1,k2,k3"
)
```

Colocate Join

Если вы хотите использовать атрибуты Colocate Join, укажите это в `properties`.

```
PROPERTIES (
    "colocate_with"="table1"
)
```

Динамические partition

Если вы хотите использовать атрибуты динамического разбиения на partition, пожалуйста, укажите это в properties.

```
PROPERTIES (
    "dynamic_partition.enable" = "true|false",
    "dynamic_partition.time_unit" = "DAY|WEEK|MONTH",
    "dynamic_partition.start" = "${integer_value}",
    "dynamic_partition.end" = "${integer_value}",
    "dynamic_partition.prefix" = "${string_value}",
    "dynamic_partition.buckets" = "${integer_value}"
```

`PROPERTIES`

Параметр	Обязательно	Описание
dynamic_partition.enable	Нет	Включить ли динамическое разбиение на partition. Допустимые значения: `TRUE` и `FALSE`. Значение по умолчанию: `TRUE`.
dynamic_partition.time_unit	Да	Временная гранулярность для динамически созданных partition. Это обязательный параметр. Допустимые значения: `DAY`, `WEEK` и `MONTH`. Временная гранулярность определяет формат суффикса для динамически созданных partition. - Если значение `DAY`, формат суффикса для динамически созданных partition — `yyyyMMdd`. Пример суффикса имени partition — `20200321`. - Если значение `WEEK`, формат суффикса для динамически созданных partition — `yyyy_ww`, например `2020_13` для 13-й недели 2020 года. - Если значение `MONTH`, формат суффикса для динамически созданных partition — `yyyyMM`, например `202003`.
dynamic_partition.start	Нет	Начальное смещение динамического разбиения на partition. Значение этого параметра должно быть отрицательным целым числом. Partition до этого смещения будут удалены на основе текущего дня, недели или месяца, который определяется `dynamic_partition.time_unit`. Значение по умолчанию — `Integer.MIN_VALUE`, а именно -2147483648, что означает, что исторические partition не будут удалены.

Параметр	Обязательно	Описание
dynamic_partition.end	Да	Конечное смещение динамического разбиения на partition. Значение этого параметра должно быть положительным целым числом. Partition от текущего дня, недели или месяца до конечного смещения будут созданы заранее.
dynamic_partition.prefix	Нет	Префикс, добавляемый к именам динамических partition. Значение по умолчанию: `p`.
dynamic_partition.buckets	Нет	Количество bucket'ов на динамический partition. Значение по умолчанию совпадает с количеством bucket'ов, определяемым зарезервированным словом `BUCKETS`, или автоматически устанавливаемым Selena.

> **ПРИМЕЧАНИЕ**
>
> Когда столбец partition имеет тип INT, его формат должен быть `yyyyMMdd`, независимо от гранулярности времени partition.

Размер bucket с random bucketing

Начиная с v1.5.2, для таблиц, настроенных с random bucketing, вы можете указать размер bucket с помощью параметра `bucket_size` в `PROPERTIES` при создании таблицы, чтобы включить динамическое увеличение количества bucket'ов по требованию. Единица измерения: B.

```
PROPERTIES (
    "bucket_size" = "1073741824"
)
```

Алгоритм сжатия данных

Вы можете указать алгоритм сжатия данных для таблицы, добавив свойство `compression` при создании таблицы.

Допустимые значения `compression`:

- `LZ4`: алгоритм LZ4.

- `ZSTD`: алгоритм Zstandard.
- `ZLIB`: алгоритм zlib.
- `SNAPPY`: алгоритм Snappy.

Начиная с v1.5.2, Selena поддерживает указание уровня сжатия для формата сжатия zstd при создании таблицы.

Синтаксис:

```
PROPERTIES ("compression" = "zstd(<compression_level>)")
```

`compression_level`: уровень сжатия для формата сжатия ZSTD. Тип: Integer. Диапазон: [1,22]. По умолчанию: `3` (Рекомендуется). Чем больше число, тем выше коэффициент сжатия. Чем выше уровень сжатия, тем больше времени требуется на сжатие и распаковку.

Пример:

```
PROPERTIES ("compression" = "zstd(3)")
```

Для получения дополнительной информации о том, как выбрать подходящий алгоритм сжатия данных, см. Сжатие данных.

Кворум записи для загрузки данных

Если ваш кластер Selena имеет несколько реплик данных, вы можете установить разный кворум записи для таблиц, то есть сколько реплик должно вернуть успешную загрузку, прежде чем Selena сможет определить, что задача загрузки успешна. Вы можете указать кворум записи, добавив свойство `write_quorum` при создании таблицы. Это свойство поддерживается начиная с v1.5.2.

Допустимые значения `write_quorum`:

- `MAJORITY`: Значение по умолчанию. Когда **большинство** реплик данных возвращают успешную загрузку, Selena возвращает успешную задачу загрузки. В противном случае Selena возвращает неудачную задачу загрузки.

- `ONE`: Когда **одна** из реплик данных возвращает успешную загрузку, Selena возвращает успешную задачу загрузки. В противном случае Selena возвращает неудачную задачу загрузки.
- `ALL`: Когда **все** реплики данных возвращают успешную загрузку, Selena возвращает успешную задачу загрузки. В противном случае Selena возвращает неудачную задачу загрузки.

ПРЕДУПРЕЖДЕНИЕ

- Установка низкого кворума записи для загрузки увеличивает риск недоступности данных и даже потери. Например, вы загружаете данные в таблицу с одним кворумом записи в кластере Selena с двумя репликами, и данные были успешно загружены только в одну реплику. Несмотря на то, что Selena определяет задачу загрузки как успешную, существует только одна выжившая реплика данных. Если сервер, который хранит tablet'ы загруженных данных, выходит из строя, данные в этих tablet'ах становятся недоступными. И если диск сервера поврежден, данные теряются.
- Selena возвращает статус задачи загрузки только после того, как все реплики данных вернули статус. Selena не вернет статус задачи загрузки, когда есть реплики, статус загрузки которых неизвестен. В реплике тайм-аут загрузки также считается неудачной загрузкой.

Режим записи и репликации данных реплики

Если ваш кластер Selena имеет несколько реплик данных, вы можете указать параметр `replicated_storage` в `PROPERTIES` для настройки режима записи и репликации данных между репликами.

- `true` (по умолчанию в v1.5.2 и более поздних версиях) указывает на "single leader replication", что означает, что данные записываются только в основную реплику. Другие реплики синхронизируют данные из основной реплики. Этот режим значительно снижает затраты на CPU, вызванные записью данных в несколько реплик. Он поддерживается начиная с v1.5.2.
- `false` (по умолчанию в v1.5.2) указывает на "leaderless replication", что означает, что данные напрямую записываются в несколько реплик, без различения основных и вторичных реплик. Затраты на CPU умножаются на количество реплик.

В большинстве случаев использование значения по умолчанию обеспечивает лучшую производительность записи данных. Если вы хотите изменить режим записи и репликации данных между репликами, выполните команду ALTER TABLE. Пример:

```
ALTER TABLE example_db.my_table
SET ("replicated_storage" = "false");
```

Ограничения unique и foreign key для Delta Join

Чтобы включить перезапись запросов в сценарии View Delta Join, вы должны определить ограничения Unique Key `unique_constraints` и ограничения Foreign Key `foreign_key_constraints` для таблицы, которая будет объединена в Delta Join. См. Асинхронное материализованное представление - Перезапись запросов в сценарии View Delta Join для получения дополнительной информации.

```
PROPERTIES (
    "unique_constraints" = "<unique_key>[, ...]",
    "foreign_key_constraints" = "
    (<child_column>[, ...])
    REFERENCES
    [catalog_name].[database_name].<parent_table_name>(<parent_column>[,
...])
    [;...]
    "
)
```

- `child_column`: Foreign Key таблицы. Вы можете определить несколько `child_column`.
- `catalog_name`: имя каталога, в котором находится таблица для объединения. По умолчанию используется каталог по умолчанию, если этот параметр не указан.
- `database_name`: имя базы данных, в которой находится таблица для объединения. Используется текущая база данных, если этот параметр не указан.
- `parent_table_name`: имя таблицы для объединения.
- `parent_column`: столбец для объединения. Они должны быть Primary Keys или Unique Keys соответствующих таблиц.

ПРЕДУПРЕЖДЕНИЕ

- `unique_constraints` и `foreign_key_constraints` используются только для перезаписи запросов. Проверки ограничений Foreign Key не гарантируются при загрузке данных в таблицу. Вы должны убедиться, что данные, загружаемые в таблицу, соответствуют ограничениям.
- Первичные ключи таблицы Primary Key или уникальные ключи таблицы Unique Key по умолчанию являются соответствующими `unique_constraints`. Вам не нужно устанавливать это вручную.
- `child_column` в `foreign_key_constraints` таблицы должен ссылаться на `unique_key` в `unique_constraints` другой таблицы.
- Количество `child_column` и `parent_column` должно совпадать.
- Типы данных `child_column` и соответствующего `parent_column` должны совпадать.

Cloud-native таблицы для кластеров shared-data

Чтобы использовать ваш кластер Selena Shared-data, вы должны создать cloud-native таблицы со следующими свойствами:

```
PROPERTIES (
    "storage_volume" = "<storage_volume_name>",
    "datacache.enable" = "{ true | false }",
    "datacache.partition_duration" = "<string_value>",
    "file_bundling" = "{ true | false }"
)
```

- `storage_volume`: Имя тома хранилища, используемого для хранения cloud-native таблицы, которую вы хотите создать. Если это свойство не указано, используется том хранилища по умолчанию. Это свойство поддерживается начиная с v1.5.2.

- `datacache.enable`: Включить ли кеш локального диска. По умолчанию: `true`.

  - Когда это свойство установлено в `true`, данные, которые должны быть загружены, одновременно записываются в объектное хранилище и локальный диск (в качестве кеша для ускорения запросов).
  - Когда это свойство установлено в `false`, данные загружаются только в объектное хранилище.

> **ПРИМЕЧАНИЕ**
>
> Чтобы включить кеш локального диска, вы должны указать каталог диска в параметре конфигурации BE `storage_root_path`.

- `datacache.partition_duration`: Срок действия горячих данных. Когда кеш локального диска включен, все данные загружаются в кеш. Когда кеш заполнен, Selena удаляет менее недавно использованные данные из кеша. Когда запросу необходимо просканировать удаленные данные, Selena проверяет, находятся ли данные в пределах срока действия. Если данные находятся в пределах срока действия, Selena снова загружает данные в кеш. Если данные не находятся в пределах срока действия, Selena не загружает их в кеш. Это свойство представляет собой строковое значение, которое может быть указано со следующими единицами: `YEAR`, `MONTH`, `DAY` и `HOUR`, например, `7 DAY` и `12 HOUR`. Если оно не указано, все данные кешируются как горячие данные.

> **ПРИМЕЧАНИЕ**
>
> Это свойство доступно только тогда, когда `datacache.enable` установлено в `true`.

- `file_bundling` (Опционально): Включить ли оптимизацию File Bundling для cloud-native таблицы. Поддерживается начиная с v2.0.0. Когда эта функция включена (установка в `true`), система автоматически объединяет файлы данных, генерируемые операциями загрузки, Compaction или Publish, тем самым снижая затраты на API, вызванные частым доступом к системе внешнего хранилища.

ПРИМЕЧАНИЕ

- File Bundling доступен только для кластеров shared-data с Selena v2.0.0 или более поздней версией.
- File Bundling включен по умолчанию для таблиц, созданных в v2.0.0 или более поздней версии, управляется конфигурацией FE `enable_file_bundling` (По умолчанию: true).
- После включения File Bundling вы можете понизить версию кластера только до v1.5.2 или более поздней версии. Если вы хотите понизить версию до версий до v1.5.2, вы должны сначала удалить таблицы, для которых включена File Bundling.
- File Bundling остается отключенным по умолчанию для существующих таблиц после обновления кластера до v2.0.0.
- Вы можете вручную включить File Bundling для существующих таблиц, используя оператор ALTER TABLE со следующими ограничениями:
  - Вы не можете включить File Bundling для таблиц с Rollup Indexes, созданными в версиях до v2.0.0. Вы можете удалить и пересоздать индексы в v2.0.0 или более поздней версии, а затем включить File Bundling для таблиц.

  - Вы не можете изменять свойство `file_bundling` **повторно** в течение определенного периода времени. В противном случае система вернет ошибку. Вы можете проверить, доступно ли свойство `file_bundling` для изменения, выполнив следующий SQL-запрос:

    ```
    SELECT METADATA_SWITCH_VERSION FROM
    information_schema.partitions_meta WHERE TABLE_NAME =
    '<table_name>';
    ```

    Вам разрешено изменять свойство `file_bundling` только тогда, когда возвращается `0`. Ненулевые значения указывают на то, что версия данных, соответствующая `METADATA_SWITCH_VERSION`, еще не была возвращена механизмом GC. Вы должны подождать, пока версия данных не будет возвращена.

    Вы можете сократить этот интервал, установив более низкое значение для динамической конфигурации FE

> `lake_autovacuum_grace_period_minutes`. Однако не забудьте сбросить конфигурацию на исходное значение после изменения свойства `file_bundling`.

Быстрая эволюция схемы

`fast_schema_evolution`: Включить ли быструю эволюцию схемы для таблицы. Допустимые значения: `TRUE` или `FALSE` (по умолчанию). Включение быстрой эволюции схемы может увеличить скорость изменения схемы и уменьшить использование ресурсов при добавлении или удалении столбцов. В настоящее время это свойство может быть включено только при создании таблицы и не может быть изменено с помощью ALTER TABLE после создания таблицы.

> **ПРИМЕЧАНИЕ**
>
> - Быстрая эволюция схемы поддерживается для кластеров shared-nothing начиная с v1.5.2.
> - Быстрая эволюция схемы поддерживается для кластеров shared-data начиная с v1.5.2 и включена по умолчанию. Вам не нужно указывать это свойство при создании cloud-native таблиц в кластерах shared-data. Динамический параметр FE `enable_fast_schema_evolution` (По умолчанию: true) управляет этим поведением.

Запрет Base Compaction

`base_compaction_forbidden_time_ranges`: Временной диапазон, в течение которого Base Compaction запрещена для таблицы. Когда это свойство установлено, система выполняет Base Compaction на подходящих tablet'ах только за пределами указанного временного диапазона. Это свойство поддерживается начиная с v1.5.2.

> **ПРИМЕЧАНИЕ**
>
> Убедитесь, что количество загрузок данных в таблицу не превышает 500 в течение периода, когда Base Compaction запрещена.

Значение `base_compaction_forbidden_time_ranges` следует синтаксису Quartz cron и поддерживает только эти поля: `<minute> <hour> <day-of-the-month> <month> <day-of-`

the-week>, где <minute> должна быть *.

```
crontab_param_value ::= [ "" | crontab ]

crontab ::= * <hour> <day-of-the-month> <month> <day-of-the-week>
```

- Когда это свойство не установлено или установлено в "" (пустая строка), Base Compaction не запрещена ни в какое время.
- Когда это свойство установлено в * * * * *, Base Compaction всегда запрещена.
- Другие значения следуют синтаксису Quartz cron.
  - Независимое значение указывает единицу времени поля. Например, 8 в поле <hour> означает 8:00-8:59.
  - Диапазон значений указывает временной диапазон поля. Например, 8-9 в поле <hour> означает 8:00-9:59.
  - Несколько диапазонов значений, разделенных запятыми, указывают несколько временных диапазонов поля.
  - <day of the week> имеет начальное значение 1 для воскресенья, а 7 означает субботу.

Пример:

```
-- Запретить Base Compaction с 8:00 до 9:00 вечера каждый день.
'base_compaction_forbidden_time_ranges' = '* 8-20 * * *'

-- Запретить Base Compaction с 0:00 до 5:00 утра и с 9:00 вечера до 11:00
вечера каждый день.
'base_compaction_forbidden_time_ranges' = '* 0-4,21-22 * * *'

-- Запретить Base Compaction с понедельника по пятницу (то есть разрешить
его в субботу и воскресенье).
'base_compaction_forbidden_time_ranges' = '* * * * 2-6'

-- Запретить Base Compaction с 8:00 до 9:00 вечера каждый рабочий день (то
есть с понедельника по пятницу).
'base_compaction_forbidden_time_ranges' = '* 8-20 * * 2-6'
```

Указать Common Partition Expression TTL

Начиная с v1.5.2, собственные таблицы Selena поддерживают Common Partition Expression TTL.

`partition_retention_condition`: Выражение, которое объявляет partition, которые должны быть сохранены динамически. Partition, которые не соответствуют условию в выражении, будут регулярно удаляться.

- Выражение может содержать только столбцы partition и константы. Столбцы, не являющиеся столбцами partition, не поддерживаются.
- Common Partition Expression применяется к List partition и Range partition по-разному:
  - Для таблиц с List partition Selena поддерживает удаление partition, отфильтрованных Common Partition Expression.
  - Для таблиц с Range partition Selena может только фильтровать и удалять partition, используя возможность обрезки partition FE. Partition, соответствующие предикатам, которые не поддерживаются обрезкой partition, не могут быть отфильтрованы и удалены.

Пример:

```
-- Сохранить данные за последние три месяца. Столбец `dt` является столбцом partition таблицы.
"partition_retention_condition" = "dt >= CURRENT_DATE() - INTERVAL 3 MONTH"
```

Чтобы отключить эту функцию, вы можете использовать оператор ALTER TABLE, чтобы установить это свойство в пустую строку:

```
ALTER TABLE tbl SET('partition_retention_condition' = '');
```

Установить свойства Flat JSON на уровне таблицы

В v1.5.2 Selena представила функцию Flat JSON для улучшения эффективности запросов данных JSON и уменьшения сложности использования JSON. Эта функция контролировалась определенными элементами конфигурации BE и системными переменными. В результате она могла быть включена (или отключена) только глобально.

Начиная с v2.0.0, вы можете устанавливать свойства, связанные с Flat JSON, на уровне таблицы.

```
PROPERTIES (
    "flat_json.enable" = "{ true | false }",
    "flat_json.null.factor" = "",
    "flat_json.sparsity.factor" = "",
    "flat_json.column.max" = ""
)
```

- `flat_json.enable` (Опционально): Включить ли функцию Flat JSON. После включения этой функции вновь загруженные данные JSON будут автоматически сглажены, что улучшит производительность запросов JSON.
- `flat_json.null.factor` (Опционально): Пороговая доля значений NULL в столбце. Столбец не будет извлечен Flat JSON, если его доля значений NULL выше этого порога. Этот параметр действует только тогда, когда `flat_json.enable` установлен в `true`. Значение по умолчанию: `0.3`.
- `flat_json.sparsity.factor` (Опционально): Пороговая доля столбцов с одинаковым именем. Столбец не будет извлечен Flat JSON, если доля столбцов с одинаковым именем ниже этого значения. Этот параметр действует только тогда, когда `flat_json.enable` установлен в `true`. Значение по умолчанию: `0.3`.
- `flat_json.column.max` (Опционально): Максимальное количество подполей, которые могут быть извлечены Flat JSON. Этот параметр действует только тогда, когда `flat_json.enable` установлен в `true`. Значение по умолчанию: `100`.

Примеры

Aggregate таблица с Hash bucketing и columnar хранилищем

```
CREATE TABLE example_db.table_hash
(
    k1 TINYINT,
    k2 DECIMAL(10, 2) DEFAULT "10.5",
    v1 CHAR(10) REPLACE,
    v2 INT SUM
)
ENGINE=olap
AGGREGATE KEY(k1, k2)
COMMENT "my first selena table"
DISTRIBUTED BY HASH(k1)
PROPERTIES ("storage_type"="column");
```

Aggregate таблица с установленным носителем хранения и временем охлаждения

```
CREATE TABLE example_db.table_hash
(
    k1 BIGINT,
    k2 LARGEINT,
    v1 VARCHAR(2048) REPLACE,
    v2 SMALLINT SUM DEFAULT "10"
)
ENGINE=olap
UNIQUE KEY(k1, k2)
DISTRIBUTED BY HASH (k1, k2)
PROPERTIES(
    "storage_type"="column",
    "storage_medium" = "SSD",
    "storage_cooldown_time" = "2015-06-04 00:00:00"
);
```

Duplicate Key таблица с Range partition, Hash bucketing, column-based хранилищем, носителем хранения и временем охлаждения

LESS THAN

```
CREATE TABLE example_db.table_range
(
    k1 DATE,
    k2 INT,
    k3 SMALLINT,
    v1 VARCHAR(2048),
    v2 DATETIME DEFAULT "2014-02-04 15:36:00"
)
ENGINE=olap
DUPLICATE KEY(k1, k2, k3)
PARTITION BY RANGE (k1)
(
    PARTITION p1 VALUES LESS THAN ("2014-01-01"),
    PARTITION p2 VALUES LESS THAN ("2014-06-01"),
    PARTITION p3 VALUES LESS THAN ("2014-12-01")
)
DISTRIBUTED BY HASH(k2)
PROPERTIES(
    "storage_medium" = "SSD",
    "storage_cooldown_time" = "2015-06-04 00:00:00"
);
```

Примечание:

Это выражение создаст три partition данных:

```
( {    MIN     },   {"2014-01-01"} )
[ {"2014-01-01"},   {"2014-06-01"} )
[ {"2014-06-01"},   {"2014-12-01"} )
```

Данные за пределами этих диапазонов не будут загружены.

Fixed Range

```
CREATE TABLE table_range
(
    k1 DATE,
    k2 INT,
    k3 SMALLINT,
    v1 VARCHAR(2048),
    v2 DATETIME DEFAULT "2014-02-04 15:36:00"
)
ENGINE=olap
DUPLICATE KEY(k1, k2, k3)
PARTITION BY RANGE (k1, k2, k3)
(
    PARTITION p1 VALUES [("2014-01-01", "10", "200"), ("2014-01-01", "20",
"300")),
    PARTITION p2 VALUES [("2014-06-01", "100", "200"), ("2014-07-01",
"100", "300"))
)
DISTRIBUTED BY HASH(k2)
PROPERTIES(
    "storage_medium" = "SSD"
);
```

Внешняя таблица MySQL

```
CREATE EXTERNAL TABLE example_db.table_mysql
(
    k1 DATE,
    k2 INT,
    k3 SMALLINT,
    k4 VARCHAR(2048),
    k5 DATETIME
)
ENGINE=mysql
PROPERTIES
(
    "host" = "127.0.0.1",
    "port" = "8239",
    "user" = "mysql_user",
    "password" = "mysql_passwd",
    "database" = "mysql_db_test",
    "table" = "mysql_table_test"
)
```

Таблица со столбцами HLL

```
CREATE TABLE example_db.example_table
(
    k1 TINYINT,
    k2 DECIMAL(10, 2) DEFAULT "10.5",
    v1 HLL HLL_UNION,
    v2 HLL HLL_UNION
)
ENGINE=olap
AGGREGATE KEY(k1, k2)
DISTRIBUTED BY HASH(k1)
PROPERTIES ("storage_type"="column");
```

Таблица, использующая тип агрегации BITMAP_UNION

Исходный тип данных столбцов `v1` и `v2` должен быть TINYINT, SMALLINT или INT.

```
CREATE TABLE example_db.example_table
(
    k1 TINYINT,
    k2 DECIMAL(10, 2) DEFAULT "10.5",
    v1 BITMAP BITMAP_UNION,
    v2 BITMAP BITMAP_UNION
)
ENGINE=olap
AGGREGATE KEY(k1, k2)
DISTRIBUTED BY HASH(k1)
PROPERTIES ("storage_type"="column");
```

© ООО «Платформа Данных Селена», 2026

Таблицы, поддерживающие Colocate Join

```
CREATE TABLE `t1`
(
     `id` int(11) COMMENT "",
    `value` varchar(8) COMMENT ""
)
ENGINE=OLAP
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`)
PROPERTIES
(
    "colocate_with" = "t1"
);

CREATE TABLE `t2`
(
    `id` int(11) COMMENT "",
    `value` varchar(8) COMMENT ""
)
ENGINE=OLAP
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`)
PROPERTIES
(
    "colocate_with" = "t1"
);
```

Таблица с bitmap индексом

```
CREATE TABLE example_db.table_hash
(
    k1 TINYINT,
    k2 DECIMAL(10, 2) DEFAULT "10.5",
    v1 CHAR(10) REPLACE,
    v2 INT SUM,
    INDEX k1_idx (k1) USING BITMAP COMMENT 'xxxxxx'
)
ENGINE=olap
AGGREGATE KEY(k1, k2)
COMMENT "my first selena table"
DISTRIBUTED BY HASH(k1)
PROPERTIES ("storage_type"="column");
```

Таблица с динамическими partition

Функция динамического разбиения на partition должна быть включена ("dynamic_partition.enable" = "true") в конфигурации FE. Для получения дополнительной информации см. Настройка динамических partition.

Этот пример создает partition на следующие три дня и удаляет partition, созданные три дня назад. Например, если сегодня 2020-01-08, будут созданы partition со следующими именами: p20200108, p20200109, p20200110, p20200111, а их диапазоны:

```
[types: [DATE]; keys: [2020-01-08]; ..types: [DATE]; keys: [2020-01-09]; )
[types: [DATE]; keys: [2020-01-09]; ..types: [DATE]; keys: [2020-01-10]; )
[types: [DATE]; keys: [2020-01-10]; ..types: [DATE]; keys: [2020-01-11]; )
[types: [DATE]; keys: [2020-01-11]; ..types: [DATE]; keys: [2020-01-12]; )
```

```
CREATE TABLE example_db.dynamic_partition
(
    k1 DATE,
    k2 INT,
    k3 SMALLINT,
    v1 VARCHAR(2048),
    v2 DATETIME DEFAULT "2014-02-04 15:36:00"
)
ENGINE=olap
DUPLICATE KEY(k1, k2, k3)
PARTITION BY RANGE (k1)
(
    PARTITION p1 VALUES LESS THAN ("2014-01-01"),
    PARTITION p2 VALUES LESS THAN ("2014-06-01"),
    PARTITION p3 VALUES LESS THAN ("2014-12-01")
)
DISTRIBUTED BY HASH(k2)
PROPERTIES(
    "storage_medium" = "SSD",
    "dynamic_partition.enable" = "true",
    "dynamic_partition.time_unit" = "DAY",
    "dynamic_partition.start" = "-3",
    "dynamic_partition.end" = "3",
    "dynamic_partition.prefix" = "p",
    "dynamic_partition.buckets" = "10"
);
```

Таблица с несколькими partition, созданными пакетно, и целочисленным столбцом в качестве столбца разбиения на partition

В следующем примере столбец разбиения на partition `datekey` имеет тип INT. Все partition создаются одним простым предложением partition `START ("1") END ("5") EVERY (1)`. Диапазон всех partition начинается с `1` и заканчивается на `5` с гранулярностью partition `1`:

> **ПРИМЕЧАНИЕ**
>
> Значения столбца разбиения на partition в **START()** и **END()** должны быть заключены в кавычки, в то время как гранулярность partition в **EVERY()** не нужно заключать в кавычки.

```
CREATE TABLE site_access (
    datekey INT,
    site_id INT,
    city_code SMALLINT,
    user_name VARCHAR(32),
    pv BIGINT DEFAULT '0'
)
ENGINE=olap
DUPLICATE KEY(datekey, site_id, city_code, user_name)
PARTITION BY RANGE (datekey) (START ("1") END ("5") EVERY (1)
)
DISTRIBUTED BY HASH(site_id)
PROPERTIES ("replication_num" = "3");
```

Внешняя таблица Hive

Прежде чем создавать внешнюю таблицу Hive, вы должны создать ресурс и базу данных Hive. Для получения дополнительной информации см. Внешняя таблица.

```
CREATE EXTERNAL TABLE example_db.table_hive
(
    k1 TINYINT,
    k2 VARCHAR(50),
    v INT
)
ENGINE=hive
PROPERTIES
(
    "resource" = "hive0",
    "database" = "hive_db_name",
    "table" = "hive_table_name"
);
```

Таблица Primary Key с конкретным ключом сортировки

Предположим, вам нужно анализировать поведение пользователей в режиме реального времени с таких измерений, как адрес пользователей и время последней активности. При создании таблицы вы можете определить столбец `user_id` в качестве первичного ключа и определить комбинацию столбцов `address` и `last_active` в качестве ключа сортировки.

```
create table users (
    user_id bigint NOT NULL,
    name string NOT NULL,
    email string NULL,
    address string NULL,
    age tinyint NULL,
    sex tinyint NULL,
    last_active datetime,
    property0 tinyint NOT NULL,
    property1 tinyint NOT NULL,
    property2 tinyint NOT NULL,
    property3 tinyint NOT NULL
)
PRIMARY KEY (`user_id`)
DISTRIBUTED BY HASH(`user_id`)
ORDER BY(`address`,`last_active`)
PROPERTIES(
    "replication_num" = "3",
    "enable_persistent_index" = "true"
);
```

Разделенная временная таблица

```
CREATE TEMPORARY TABLE example_db.temp_table
(
    k1 DATE,
    k2 INT,
    k3 SMALLINT,
    v1 VARCHAR(2048),
    v2 DATETIME DEFAULT "2014-02-04 15:36:00"
)
ENGINE=olap
DUPLICATE KEY(k1, k2, k3)
PARTITION BY RANGE (k1)
(
    PARTITION p1 VALUES LESS THAN ("2014-01-01"),
    PARTITION p2 VALUES LESS THAN ("2014-06-01"),
    PARTITION p3 VALUES LESS THAN ("2014-12-01")
)
DISTRIBUTED BY HASH(k2);
```

Таблица со свойствами Flat JSON

```
CREATE TABLE example_db.example_table
(
    k1 DATE,
    k2 INT,
    v1 VARCHAR(2048),
    v2 JSON
)
ENGINE=olap
DUPLICATE KEY(k1, k2)
DISTRIBUTED BY HASH(k2)
PROPERTIES (
    "flat_json.enable" = "true",
    "flat_json.null.factor" = "0.5",
    "flat_json.sparsity.factor" = "0.5",
    "flat_json.column.max" = "50"
);
```

Ссылки

- SHOW CREATE TABLE
- SHOW TABLES

- USE
- ALTER TABLE

CREATE TABLE AS SELECT

Используйте выражение CREATE TABLE AS SELECT (CTAS) для синхронного или асинхронного запроса таблицы и создания новой таблицы на основе результата запроса, а затем для вставки результата запроса в новую таблицу.

Вы можете отправить асинхронную CTAS задачу с помощью SUBMIT TASK.

Синтаксис

- Синхронный запрос таблицы и создание новой таблицы на основе результата запроса, а затем вставка результата запроса в новую таблицу.

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] [database.]table_name
[column_name1 [, column_name2, ...]]
[index_definition1 [, index_definition2, ...]]
[key_desc]
[COMMENT "table comment"]
[partition_desc]
[distribution_desc]
[ORDER BY (column_name1 [, column_name2, ...])]
[PROPERTIES ("key"="value", ...)]
AS SELECT query
[ ... ]
```

- Асинхронный запрос таблицы и создание новой таблицы на основе результата запроса, а затем вставка результата запроса в новую таблицу.

```
SUBMIT [/*+ SET_VAR(key=value) */] TASK [[database.]<task_name>]AS
CREATE TABLE [IF NOT EXISTS] [database.]table_name
[column_name1 [, column_name2, ...]]
[index_definition1 [, index_definition2, ...]]
[key_desc]
[COMMENT "table comment"]
[partition_desc]
[distribution_desc]
[ORDER BY (column_name1 [, column_name2, ...])]
[PROPERTIES ("key"="value", ...)] AS SELECT query
[ ... ]
```

Параметры

Параметр	Обязательный	Описание
TEMPORARY	Нет	Создает временную таблицу. Начиная с v1.5.2, Selena поддерживает создание временных таблиц в Default Catalog. Дополнительную информацию см. в разделе Временная таблица. В настоящее время Selena не поддерживает создание временных таблиц с помощью асинхронных задач через SUBMIT TASK.
column_name	Нет	Имя столбца в новой таблице. Вам не нужно указывать тип данных для столбца. Selena автоматически назначает подходящий тип данных для столбца. Selena преобразует данные FLOAT и DOUBLE в данные DECIMAL(38,9). Selena также преобразует данные CHAR, VARCHAR и STRING в данные VARCHAR(65533).
index_definition	Нет	Начиная с v1.5.2, для новой таблицы можно создать bitmap index. Синтаксис: `INDEX index_name (col_name[, col_name, ...]) [USING BITMAP] COMMENT 'xxxxxx'`. Дополнительную информацию об описании параметров и примечаниях по использованию см. в разделе Bitmap indexes.
key_desc	Нет	Синтаксис: `key_type ( <col_name1> [, <col_name2> , ...])`. **Параметры**: • `key_type`: тип ключа новой таблицы. Допустимые значения: `DUPLICATE KEY` и `PRIMARY KEY`. Значение по умолчанию: `DUPLICATE KEY`. • `col_name`: столбец для формирования ключа.
COMM	Нет	Комментарий к новой таблице.

Параметр	Обязательный	Описание
ENT		
partition_desc	Нет	Метод партиционирования новой таблицы. По умолчанию, если вы не указываете этот параметр, новая таблица не имеет партиций. Дополнительную информацию о партиционировании см. в разделе CREATE TABLE.
distribution_desc	Нет	Метод бакетирования новой таблицы. Если вы не указываете этот параметр, столбец bucket по умолчанию устанавливается как столбец с наибольшей кардинальностью, собранной оптимизатором на основе стоимости (CBO). Количество bucket по умолчанию равно 10. Если CBO не собирает информацию о кардинальности, столбец bucket по умолчанию устанавливается как первый столбец в новой таблице. Дополнительную информацию о бакетировании см. в разделе CREATE TABLE.
ORDER BY	Нет	Начиная с v1.5.2, для новой таблицы можно указать sort key, если новая таблица является Primary Key таблицей. Sort key может быть комбинацией любых столбцов. Primary Key таблица — это таблица, для которой при создании таблицы указан `PRIMARY KEY (xxx)`.
Properties	Нет	Свойства новой таблицы.
AS SELECT query	Да	Результат запроса. Вы можете указать столбцы в `... AS SELECT query`, например, `... AS SELECT a, b, c FROM table_a;`. В этом примере `a`, `b` и `c` обозначают имена столбцов запрашиваемой таблицы. Если вы не указываете имена столбцов новой таблицы, имена столбцов новой таблицы также будут `a`, `b` и `c`. Вы можете указать выражения в `... AS SELECT query`, например, `... AS SELECT a+1 AS x, b+2 AS y, c*c AS z FROM table_a;`. В этом примере `a+1`, `b+2` и `c*c` обозначают имена столбцов запрашиваемой таблицы, а `x`, `y` и `z` обозначают имена столбцов новой таблицы. Примечание: Количество столбцов в новой таблице должно совпадать с количеством столбцов, указанных в выражении SELECT. Рекомендуется использовать имена столбцов, которые легко идентифицировать.

Примечания по использованию

- Выражение CTAS может создать новую таблицу только при соблюдении следующих требований:
  - `ENGINE` - это `OLAP`.

  - По умолчанию таблица является Duplicate Key таблицей. Вы также можете указать её как Primary Key таблицу в `key_desc`.
  - Sort keys — это первые три столбца, и объем хранения типов данных этих трех столбцов не превышает 36 байт.
- Если выполнение выражения CTAS завершится неудачей по таким причинам, как перезапуск FE, может возникнуть одна из следующих проблем:
  - Новая таблица успешно создана, но не содержит данных.
  - Новая таблица не создана.
- После создания новой таблицы, если для вставки данных в новую таблицу используется несколько методов (таких как INSERT INTO), метод, который первым завершит операцию INSERT, вставит свои данные в новую таблицу.
- После создания новой таблицы вам необходимо вручную предоставить разрешения на эту таблицу пользователям.
- Если вы не указываете имя для Task при асинхронном запросе таблицы и создании новой таблицы на основе результата запроса, Selena автоматически генерирует имя для Task.

Примеры

Пример 1: Синхронный запрос таблицы `order` и создание новой таблицы `order_new` на основе результата запроса, а затем вставка результата запроса в новую таблицу.

```
CREATE TABLE order_new
AS SELECT * FROM order;
```

Пример 2: Синхронный запрос столбцов `k1`, `k2` и `k3` в таблице `order` и создание новой таблицы `order_new` на основе результата запроса, а затем вставка результата запроса в новую таблицу. Кроме того, установить имена столбцов новой таблицы как `a`, `b` и `c`.

```
CREATE TABLE order_new (a, b, c)
AS SELECT k1, k2, k3 FROM order;
```

или

```
CREATE TABLE order_new
AS SELECT k1 AS a, k2 AS b, k3 AS c FROM order;
```

Пример 3: Синхронный запрос наибольшего значения столбца `salary` в таблице `employee` и создание новой таблицы `employee_new` на основе результата запроса, а затем вставка результата запроса в новую таблицу. Кроме того, установить имя столбца новой таблицы как `salary_max`.

```
CREATE TABLE employee_new
AS SELECT MAX(salary) AS salary_max FROM employee;
```

После вставки данных, запросите новую таблицу.

```
SELECT * FROM employee_new;

+------------+
| salary_max |
+------------+
|   10000    |
+------------+
```

Пример 4: Синхронный запрос столбцов `customer_id` и `first_name` в таблице `customers` и создание новой таблицы `customers_new` на основе результата запроса, а затем вставка результата запроса в новую таблицу. Кроме того, установить имена столбцов новой таблицы как `customer_id_new` и `first_name_new`. Также построить bitmap index для столбца `customer_id_new` в новой таблице.

```
CREATE TABLE customers_new
(   customer_id_new,
    first_name_new,
    INDEX idx_bitmap_customer_id (customer_id_new) USING BITMAP
)
AS SELECT customer_id,first_name FROM customers;
```

Пример 5: Синхронный запрос таблицы `customers` и создание новой таблицы `customers_new` на основе результата запроса, а затем вставка результата запроса в новую таблицу. Кроме того, указать новую таблицу как Primary Key таблицу и указать её sort key как `first_name` и `last_name`.

```
CREATE TABLE customers_pk
PRIMARY KEY (customer_id)
ORDER BY (first_name,last_name)
AS SELECT  * FROM customers;
```

Пример 6: Использование CTAS для создания Primary Key таблицы. Обратите внимание, что количество строк данных в Primary Key таблице может быть меньше, чем в результате запроса. Это происходит потому, что Primary Key таблица хранит только самую последнюю строку данных среди группы строк, которые имеют одинаковый primary key.

```
CREATE TABLE employee_new
PRIMARY KEY(order_id)
AS SELECT
    order_list.order_id,
    sum(goods.price) as total
FROM order_list INNER JOIN goods ON goods.item_id1 = order_list.item_id2
GROUP BY order_id;
```

Пример 7: Синхронный запрос четырех таблиц, включая `lineorder`, `customer`, `supplier` и `part`, и создание новой таблицы `lineorder_flat` на основе результата запроса, а затем вставка результата запроса в новую таблицу. Кроме того, указать метод партиционирования и метод бакетирования для новой таблицы.

```sql
CREATE TABLE lineorder_flat
PARTITION BY RANGE(`LO_ORDERDATE`)
(
    START ("1993-01-01") END ("1999-01-01") EVERY (INTERVAL 1 YEAR)
)
DISTRIBUTED BY HASH(`LO_ORDERKEY`) AS SELECT
    l.LO_ORDERKEY AS LO_ORDERKEY,
    l.LO_LINENUMBER AS LO_LINENUMBER,
    l.LO_CUSTKEY AS LO_CUSTKEY,
    l.LO_PARTKEY AS LO_PARTKEY,
    l.LO_SUPPKEY AS LO_SUPPKEY,
    l.LO_ORDERDATE AS LO_ORDERDATE,
    l.LO_ORDERPRIORITY AS LO_ORDERPRIORITY,
    l.LO_SHIPPRIORITY AS LO_SHIPPRIORITY,
    l.LO_QUANTITY AS LO_QUANTITY,
    l.LO_EXTENDEDPRICE AS LO_EXTENDEDPRICE,
    l.LO_ORDTOTALPRICE AS LO_ORDTOTALPRICE,
    l.LO_DISCOUNT AS LO_DISCOUNT,
    l.LO_REVENUE AS LO_REVENUE,
    l.LO_SUPPLYCOST AS LO_SUPPLYCOST,
    l.LO_TAX AS LO_TAX,
    l.LO_COMMITDATE AS LO_COMMITDATE,
    l.LO_SHIPMODE AS LO_SHIPMODE,
    c.C_NAME AS C_NAME,
    c.C_ADDRESS AS C_ADDRESS,
    c.C_CITY AS C_CITY,
    c.C_NATION AS C_NATION,
    c.C_REGION AS C_REGION,
    c.C_PHONE AS C_PHONE,
    c.C_MKTSEGMENT AS C_MKTSEGMENT,
    s.S_NAME AS S_NAME,
    s.S_ADDRESS AS S_ADDRESS,
    s.S_CITY AS S_CITY,
    s.S_NATION AS S_NATION,
    s.S_REGION AS S_REGION,
    s.S_PHONE AS S_PHONE,
    p.P_NAME AS P_NAME,
    p.P_MFGR AS P_MFGR,
    p.P_CATEGORY AS P_CATEGORY,
    p.P_BRAND AS P_BRAND,
    p.P_COLOR AS P_COLOR,
    p.P_TYPE AS P_TYPE,
    p.P_SIZE AS P_SIZE,
    p.P_CONTAINER AS P_CONTAINER FROM lineorder AS l
INNER JOIN customer AS c ON c.C_CUSTKEY = l.LO_CUSTKEY
```

```
INNER JOIN supplier AS s ON s.S_SUPPKEY = l.LO_SUPPKEY
INNER JOIN part AS p ON p.P_PARTKEY = l.LO_PARTKEY;
```

Пример 8: Асинхронный запрос таблицы `order_detail` и создание новой таблицы `order_statistics` на основе результата запроса, а затем вставка результата запроса в новую таблицу.

```
SUBMIT TASK AS CREATE TABLE order_statistics AS SELECT COUNT(*) as count
FROM order_detail;

+-------------------------------------------+-----------+
| TaskName                                  | Status    |
+-------------------------------------------+-----------+
| ctas-df6f7930-e7c9-11ec-abac-bab8ee315bf2 | SUBMITTED |
+-------------------------------------------+-----------+
```

Проверьте информацию о Task.

```
SELECT * FROM INFORMATION_SCHEMA.tasks;

-- Информация о Task

TASK_NAME: ctas-df6f7930-e7c9-11ec-abac-bab8ee315bf2
CREATE_TIME: 2022-06-14 14:07:06
SCHEDULE: MANUAL
DATABASE: default_cluster:test
DEFINITION: CREATE TABLE order_statistics AS SELECT COUNT(*) as cnt FROM
order_detail
EXPIRE_TIME: 2022-06-17 14:07:06
```

Проверьте состояние TaskRun.

```
SELECT * FROM INFORMATION_SCHEMA.task_runs;

-- Состояние TaskRun

QUERY_ID: 37bd2b63-eba8-11ec-8d41-bab8ee315bf2
TASK_NAME: ctas-df6f7930-e7c9-11ec-abac-bab8ee315bf2
CREATE_TIME: 2022-06-14 14:07:06
FINISH_TIME: 2022-06-14 14:07:07
STATE: SUCCESS
DATABASE:
DEFINITION: CREATE TABLE order_statistics AS SELECT COUNT(*) as cnt FROM
order_detail
EXPIRE_TIME: 2022-06-17 14:07:06
ERROR_CODE: 0
ERROR_MESSAGE: NULL
```

Запросите новую таблицу, когда состояние TaskRun будет `SUCCESS`.

```
SELECT * FROM order_statistics;
```

CREATE TABLE LIKE

CREATE TABLE LIKE создает идентичную пустую таблицу на основе определения другой таблицы. Определение включает определение столбцов, partition и свойства таблицы. Вы можете скопировать внешнюю таблицу, такую как MySQL.

v1.5.2 позволяет вам указать другой метод партиционирования, метод бакетирования и свойства для новой таблицы, отличные от исходной таблицы.

> **ПОДСКАЗКА**
>
> Эта операция требует привилегии CREATE TABLE в базе данных, в которой вы хотите создать таблицу, и привилегии `SELECT` в исходной таблице, на основе которой создается таблица.

Синтаксис

- Синтаксис, поддерживаемый в версиях до v1.5.2.

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [database.]<table_name>
LIKE [database.]<source_table_name>
```

- v1.5.2 поддерживает указание свойств для новой таблицы.

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [database.]<table_name>
[partition_desc]
[distribution_desc]
[PROPERTIES ("key" = "value",...)]
LIKE [database.]<source_table_name>
```

Параметры

- `TEMPORARY`: Создает временную таблицу. Начиная с v1.5.2, Selena поддерживает создание временных таблиц в Default Catalog. Дополнительную информацию см. в разделе Временная таблица.
- `database`: база данных.
- `table_name`: имя таблицы, которую вы хотите создать. Соглашения об именовании см. в разделе Системные ограничения.
- `source_table_name`: имя исходной таблицы, которую вы хотите скопировать.
-
  - `partition_desc`: метод партиционирования. Дополнительную информацию см. в разделе CREATE TABLE.
- `distribution_desc`: метод бакетирования. Дополнительную информацию см. в разделе CREATE TABLE.
- `PROPERTIES`: свойства таблицы. Поддерживаются все свойства таблицы. Дополнительную информацию см. в разделе ALTER TABLE.

Примеры

Предположим, в базе данных `test1` есть таблица `orders`.

```
create table orders (
    dt date NOT NULL,
    order_id bigint NOT NULL,
    user_id int NOT NULL,
    merchant_id int NOT NULL,
    good_id int NOT NULL,
    good_name string NOT NULL,
    price int NOT NULL,
    cnt int NOT NULL,
    revenue int NOT NULL,
    state tinyint NOT NULL
) PRIMARY KEY (dt, order_id)
PARTITION BY RANGE(`dt`) (
    PARTITION p20210820 VALUES [('2021-08-20'), ('2021-08-21')),
    PARTITION p20210821 VALUES [('2021-08-21'), ('2021-08-22')),
    PARTITION p20210929 VALUES [('2021-09-29'), ('2021-09-30')),
    PARTITION p20210930 VALUES [('2021-09-30'), ('2021-10-01'))
) DISTRIBUTED BY HASH(order_id)
PROPERTIES (
    "replication_num" = "3",
    "enable_persistent_index" = "true"
);
```

Пример 1: В базе данных `test1` создайте пустую таблицу `order_1`, которая имеет такую же структуру таблицы, как и `orders`.

```
CREATE TABLE test1.order_1 LIKE test1.orders;
```

```
show create table order_1\G
*************************** 1. row ***************************
       Table: order_1
Create Table: CREATE TABLE `order_1` (
  `dt` date NOT NULL COMMENT "",
  `order_id` bigint(20) NOT NULL COMMENT "",
  `user_id` int(11) NOT NULL COMMENT "",
  `merchant_id` int(11) NOT NULL COMMENT "",
  `good_id` int(11) NOT NULL COMMENT "",
  `good_name` varchar(65533) NOT NULL COMMENT "",
  `price` int(11) NOT NULL COMMENT "",
  `cnt` int(11) NOT NULL COMMENT "",
  `revenue` int(11) NOT NULL COMMENT "",
  `state` tinyint(4) NOT NULL COMMENT ""
) ENGINE=OLAP
PRIMARY KEY(`dt`, `order_id`)
PARTITION BY RANGE(`dt`)
(PARTITION p20210820 VALUES [("2021-08-20"), ("2021-08-21")),
PARTITION p20210821 VALUES [("2021-08-21"), ("2021-08-22")),
PARTITION p20210929 VALUES [("2021-09-29"), ("2021-09-30")),
PARTITION p20210930 VALUES [("2021-09-30"), ("2021-10-01")))
DISTRIBUTED BY HASH(`order_id`)
PROPERTIES (
"replication_num" = "3",
"in_memory" = "false",
"enable_persistent_index" = "true",
"replicated_storage" = "true",
"compression" = "LZ4"
);
```

Пример 2: Создайте пустую таблицу `order_2` на основе `orders` и укажите свойства для `order_2`.

```
CREATE TABLE order_2
PARTITION BY date_trunc('day',dt)
DISTRIBUTED BY hash(dt)
PROPERTIES ("replication_num" = "1")
LIKE orders;
```

```
show create table order_2\G
*************************** 1. row ***************************
       Table: order_2
Create Table: CREATE TABLE `order_2` (
  `dt` date NOT NULL COMMENT "",
  `order_id` bigint(20) NOT NULL COMMENT "",
  `user_id` int(11) NOT NULL COMMENT "",
  `merchant_id` int(11) NOT NULL COMMENT "",
  `good_id` int(11) NOT NULL COMMENT "",
  `good_name` varchar(65533) NOT NULL COMMENT "",
  `price` int(11) NOT NULL COMMENT "",
  `cnt` int(11) NOT NULL COMMENT "",
  `revenue` int(11) NOT NULL COMMENT "",
  `state` tinyint(4) NOT NULL COMMENT ""
) ENGINE=OLAP
PRIMARY KEY(`dt`, `order_id`)
PARTITION BY RANGE(date_trunc('day', dt))
()
DISTRIBUTED BY HASH(`dt`)
PROPERTIES (
"replication_num" = "1",
"in_memory" = "false",
"enable_persistent_index" = "true",
"replicated_storage" = "true",
"compression" = "LZ4"
);
```

Пример 3: Создайте пустую таблицу `table2`, которая имеет такую же структуру таблицы, как внешняя таблица MySQL `table1`.

```
CREATE TABLE test1.table2 LIKE test1.table1
```

DELETE

Удаляет строки данных из таблицы на основе указанных условий. Таблица может быть партиционированной или не партиционированной.

Для Duplicate Key таблиц, Aggregate таблиц и Unique Key таблиц вы можете удалять данные из указанных партиций. Начиная с v1.5.2, Primary Key таблицы поддерживают полную семантику `DELETE...WHERE`, которая позволяет удалять строки данных на основе primary key, любого столбца или результатов подзапроса. Начиная с v1.5.2, Selena

обогащает семантику `DELETE...WHERE` многотабличными соединениями и общими табличными выражениями (CTE). Если вам нужно соединить Primary Key таблицы с другими таблицами в базе данных, вы можете ссылаться на эти другие таблицы в предложении USING или CTE.

Примечания по использованию

- У вас должны быть привилегии на таблицу и базу данных, для которых вы хотите выполнить DELETE.
- Частые операции DELETE не рекомендуются. При необходимости выполняйте такие операции в непиковые часы.
- Операция DELETE удаляет только данные в таблице. Таблица остается. Чтобы удалить таблицу, выполните DROP TABLE.
- Чтобы предотвратить случайное удаление данных во всей таблице, вы должны указать предложение WHERE в выражении DELETE.
- Удаленные строки не очищаются немедленно. Они помечаются как "удаленные" и временно сохраняются в Segment. Физически строки удаляются только после завершения слияния версий данных (compaction).
- Эта операция также удаляет данные материализованных представлений, которые ссылаются на эту таблицу.
- Вы можете использовать системную переменную `insert_timeout` для установки времени ожидания для операций DELETE.

Duplicate Key таблицы, Aggregate таблицы и Unique Key таблицы

Синтаксис

```
DELETE FROM  [<db_name>.]<table_name> [PARTITION <partition_name>]
WHERE
column_name1 op { value | value_list } [ AND column_name2 op { value |
value_list } ...]
```

Параметры

Параметр	Обязательный	Описание
`db_name`	Нет	База данных, к которой принадлежит целевая таблица. Если этот параметр не указан, по умолчанию используется текущая база данных.

Параметр	Обязательный	Описание
`table_name`	Да	Таблица, из которой вы хотите удалить данные.
`partition_name`	Нет	Партиция, из которой вы хотите удалить данные.
`column_name`	Да	Столбец, который вы хотите использовать в качестве условия DELETE. Вы можете указать один или несколько столбцов.
`op`	Да	Оператор, используемый в условии DELETE. Поддерживаемые операторы: `=`, `>`, `<`, `>=`, `<=`, `!=`, `IN` и `NOT IN`.

Ограничения и примечания по использованию

- Для Duplicate Key таблиц вы можете использовать **любой столбец** в качестве условия DELETE. Для Aggregate таблиц и Unique Key таблиц в качестве условия DELETE могут использоваться только **key столбцы**.
- Условия, которые вы указываете, должны быть в отношении AND. Если вы хотите указать условия в отношении OR, вы должны указать условия в отдельных выражениях DELETE.
- Для Duplicate Key таблиц, Aggregate таблиц и Unique Key таблиц выражение DELETE не поддерживает использование результатов подзапроса в качестве условий.

Воздействие

После выполнения выражения DELETE производительность запросов вашего кластера может ухудшиться на некоторый период времени (до завершения compaction). Степень ухудшения зависит от количества условий, которые вы указываете. Большее количество условий означает более высокую степень ухудшения.

Примеры

Создайте таблицу и вставьте данные

Следующий пример создает партиционированную Duplicate Key таблицу.

```
CREATE TABLE `my_table` (
    `date` date NOT NULL,
    `k1` int(11) NOT NULL COMMENT "",
    `k2` varchar(65533) NULL DEFAULT "" COMMENT "")
DUPLICATE KEY(`date`)
PARTITION BY RANGE(`date`)
(
    PARTITION p1 VALUES [('2022-03-11'), ('2022-03-12')),
    PARTITION p2 VALUES [('2022-03-12'), ('2022-03-13'))
)
DISTRIBUTED BY HASH(`date`)
PROPERTIES
("replication_num" = "3");

INSERT INTO `my_table` VALUES
('2022-03-11', 3, 'abc'),
('2022-03-11', 2, 'acb'),
('2022-03-11', 4, 'abc'),
('2022-03-12', 2, 'bca'),
('2022-03-12', 4, 'cba'),
('2022-03-12', 5, 'cba');
```

Запросите данные

```
select * from my_table order by date;
+------------+------+------+
| date       | k1   | k2   |
+------------+------+------+
| 2022-03-11 |    3 | abc  |
| 2022-03-11 |    2 | acb  |
| 2022-03-11 |    4 | abc  |
| 2022-03-12 |    2 | bca  |
| 2022-03-12 |    4 | cba  |
| 2022-03-12 |    5 | cba  |
+------------+------+------+
```

Удалите данные

Удалите данные из указанной партиции

Удалите строки, значения `k1` которых равны `3`, из партиции `p1`.

```
DELETE FROM my_table PARTITION p1
WHERE k1 = 3;

-- Результат запроса показывает, что строки со значениями `k1` равными `3`
удалены.

select * from my_table partition (p1);
+------------+------+------+
| date       | k1   | k2   |
+------------+------+------+
| 2022-03-11 |    2 | acb  |
| 2022-03-11 |    4 | abc  |
+------------+------+------+
```

Удалите данные из указанной партиции с использованием AND

Удалите строки, значения `k1` которых больше или равны `3` и значения `k2` которых равны `"abc"`, из партиции `p1`.

```
DELETE FROM my_table PARTITION p1
WHERE k1 >= 3 AND k2 = "abc";

select * from my_table partition (p1);
+------------+------+------+
| date       | k1   | k2   |
+------------+------+------+
| 2022-03-11 |    2 | acb  |
+------------+------+------+
```

Удалите данные из всех партиций

Удалите строки, значения `k2` которых равны `"abc"` или `"cba"`, из всех партиций.

```
DELETE FROM my_table
WHERE  k2 in ("abc", "cba");

select * from my_table order by date;
+------------+------+------+
| date       | k1   | k2   |
+------------+------+------+
| 2022-03-11 |    2 | acb  |
| 2022-03-12 |    2 | bca  |
+------------+------+------+
```

Primary Key таблицы

Начиная с v1.5.2, Primary Key таблицы поддерживают полную семантику `DELETE...WHERE`, которая позволяет удалять строки данных на основе primary key, любого столбца или подзапроса.

Синтаксис

```
[ WITH <with_query> [, ...] ]
DELETE FROM <table_name>
[ USING <from_item> [, ...] ]
[ WHERE <where_condition> ]
```

Параметры

Параметр	Обязательный	Описание
`with_query`	Нет	Один или несколько CTE, на которые можно ссылаться по имени в выражении DELETE. CTE — это временные наборы результатов, которые могут улучшить читаемость сложных выражений.
`table_name`	Да	Таблица, из которой вы хотите удалить данные.

Параметр	Обязательный	Описание
`from_item`	Нет	Одна или несколько других таблиц в базе данных. Эти таблицы могут быть соединены с обрабатываемой таблицей на основе условия, указанного в предложении WHERE. На основе набора результатов запроса соединения Selena удаляет совпадающие строки из обрабатываемой таблицы. Например, если предложение USING — это `USING t1 WHERE t0.pk = t1.pk;`, Selena преобразует табличное выражение в предложении USING в `t0 JOIN t1 ON t0.pk=t1.pk;` при выполнении выражения DELETE.
`where_condition`	Да	Условие, на основе которого вы хотите удалить строки. Могут быть удалены только строки, соответствующие условию WHERE. Этот параметр является обязательным, поскольку он помогает предотвратить случайное удаление всей таблицы. Если вы хотите удалить всю таблицу, вы можете использовать 'WHERE true'.

Ограничения и примечания по использованию

- Primary Key таблицы не поддерживают удаление данных из указанной партиции, например, `DELETE FROM <table_name> PARTITION <partition_id> WHERE <where_condition>`.
- Поддерживаются следующие операторы сравнения: `=`, `>`, `<`, `>=`, `<=`, `!=`, `IN`, `NOT IN`.
- Поддерживаются следующие логические операторы: `AND` и `OR`.
- Вы не можете использовать выражение DELETE для выполнения одновременных операций DELETE или для удаления данных при загрузке данных. Если вы выполняете такие операции, атомарность, согласованность, изоляция и долговечность (ACID) транзакций могут не быть обеспечены.

Примеры

Создайте таблицу и вставьте данные

Создайте Primary Key таблицу с именем `score_board`:

```
CREATE TABLE `score_board` (
  `id` int(11) NOT NULL COMMENT "",
  `name` varchar(65533) NULL DEFAULT "" COMMENT "",
  `score` int(11) NOT NULL DEFAULT "0" COMMENT ""
) ENGINE=OLAP
PRIMARY KEY(`id`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`id`)
PROPERTIES (
"replication_num" = "3",
"storage_format" = "DEFAULT",
"enable_persistent_index" = "true"
);

INSERT INTO score_board VALUES
(0, 'Jack', 21),
(1, 'Bob', 21),
(2, 'Stan', 21),
(3, 'Sam', 22);
```

Запросите данные

Выполните следующее выражение для вставки данных в таблицу `score_board`:

```
select * from score_board;
+------+------+------+
| id   | name | score|
+------+------+------+
|    0 | Jack |   21 |
|    1 | Bob  |   21 |
|    2 | Stan |   21 |
|    3 | Sam  |   22 |
+------+------+------+
4 rows in set (0.00 sec)
```

Удалите данные

Удалите данные по primary key

Вы можете указать primary key в выражении DELETE, поэтому Selena не нужно сканировать всю таблицу.

Удалите строки, значения `id` которых равны `0`, из таблицы `score_board`.

```
DELETE FROM score_board WHERE id = 0;

select * from score_board;
+------+------+------+
| id   | name | score|
+------+------+------+
|    1 | Bob  |   21 |
|    2 | Stan |   21 |
|    3 | Sam  |   22 |
+------+------+------+
```

Удалите данные по условию

Пример 1: Удалите строки, значения `score` которых равны `22`, из таблицы `score_board`.

```
DELETE FROM score_board WHERE score = 22;

select * from score_board;
+------+------+------+
| id   | name | score|
+------+------+------+
|    0 | Jack |   21 |
|    1 | Bob  |   21 |
|    2 | Stan |   21 |
+------+------+------+
```

Пример 2: Удалите строки, значения `score` которых меньше `22`, из таблицы `score_board`.

```
DELETE FROM score_board WHERE score < 22;

select * from score_board;
+------+------+------+
| id   | name | score|
+------+------+------+
|    3 | Sam  |   22 |
+------+------+------+
```

Пример 3: Удалите строки, значения `score` которых меньше `22` и значения `name` которых не равны `Bob`, из таблицы `score_board`.

```
DELETE FROM score_board WHERE score < 22 and name != "Bob";

select * from score_board;
+------+------+------+
| id   | name | score|
+------+------+------+
|    1 | Bob  |   21 |
|    3 | Sam  |   22 |
+------+------+------+
2 rows in set (0.00 sec)
```

Удалите данные по результату подзапроса

Вы можете вложить один или несколько подзапросов в выражение `DELETE` и использовать результаты подзапроса в качестве условий.

Перед удалением данных выполните следующее выражение для создания другой таблицы с именем `users`:

```
CREATE TABLE `users` (
  `uid` int(11) NOT NULL COMMENT "",
  `name` varchar(65533) NOT NULL COMMENT "",
  `country` varchar(65533) NULL COMMENT ""
) ENGINE=OLAP
PRIMARY KEY(`uid`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`uid`)
PROPERTIES (
"replication_num" = "3",
"storage_format" = "DEFAULT",
"enable_persistent_index" = "true"
);
```

Вставьте данные в таблицу `users`:

```
INSERT INTO users VALUES
(0, "Jack", "China"),
(2, "Stan", "USA"),
(1, "Bob", "China"),
(3, "Sam", "USA");
```

```
select * from users;
+------+------+---------+
| uid  | name | country |
+------+------+---------+
|    0 | Jack | China   |
|    1 | Bob  | China   |
|    2 | Stan | USA     |
|    3 | Sam  | USA     |
+------+------+---------+
4 rows in set (0.00 sec)
```

Вложите подзапрос, чтобы найти строки, значения `country` которых равны `China`, из таблицы `users`, и удалите строки, которые имеют те же значения `name`, что и строки, возвращенные из подзапроса, из таблицы `score_board`. Вы можете использовать один из следующих методов для достижения этой цели:

- Метод 1

```
DELETE FROM score_board
WHERE name IN (select name from users where country = "China");

select * from score_board;
+------+------+------+
| id   | name | score|
+------+------+------+
|    2 | Stan |   21 |
|    3 | Sam  |   22 |
+------+------+------+
```

- Метод 2

```
DELETE FROM score_board
WHERE EXISTS (select name from users
              where score_board.name = users.name and country = "China");

select * from score_board;
+------+------+-------+
| id   | name | score |
+------+------+-------+
|    2 | Stan |    21 |
|    3 | Sam  |    22 |
+------+------+-------+
2 rows in set (0.00 sec)
```

Удалите данные с использованием многотабличного соединения или CTE

Чтобы удалить все фильмы, произведенные продюсером "foo", вы можете выполнить следующее выражение:

```
DELETE FROM films USING producers
WHERE producer_id = producers.id
    AND producers.name = 'foo';
```

Вы также можете использовать CTE для переписывания приведенного выше выражения, чтобы улучшить читаемость.

```
WITH foo_producers as (
    SELECT * from producers
    where producers.name = 'foo'
)
DELETE FROM films USING foo_producers
WHERE producer_id = foo_producers.id;
```

Ссылки

SHOW DELETE: запрашивает исторические операции DELETE, которые были успешно выполнены для Duplicate Key таблиц, Aggregate таблиц и Unique Key таблиц.

DESC

DESC используется для просмотра схем таблиц:

- Просмотр схемы таблицы, хранящейся в вашем Selena cluster, вместе с типом sort key и materialized view таблицы.
- Просмотр схемы таблицы, хранящейся в следующих внешних источниках данных, таких как Apache Hive™. Обратите внимание, что вы можете выполнить эту операцию только в Selena 2.4 и более поздних версиях.

Синтаксис

```
DESC[RIBE] { [[<catalog_name>.]<db_name>.]<table_name> [ALL] |
FILES(files_loading_properties) }
```

Параметры

Параметр	Обязательный	Описание
catalog_name	Нет	Имя внутреннего catalog или внешнего catalog. • Если вы установите значение параметра равным имени внутреннего catalog, которым является `default_catalog`, вы можете просмотреть схему таблицы, хранящейся в вашем Selena cluster. • Если вы установите значение параметра равным имени внешнего catalog, вы можете просмотреть схему таблицы, хранящейся во внешнем источнике данных.
db_name	Нет	Имя базы данных.
table_name	Да	Имя таблицы.
ALL	Нет	• Если это ключевое слово указано, вы можете просмотреть тип sort key, materialized view и схему таблицы, хранящейся в вашем Selena cluster. Если это ключевое слово не указано, вы просматриваете только схему таблицы. • Не указывайте это ключевое слово при просмотре схемы таблицы, хранящейся во внешнем источнике данных.
FILES	Нет	Табличная функция FILES(). Начиная с версии 3.3.4, вы можете использовать DESC с FILES() для просмотра информации о схеме файлов, хранящихся в удалённом хранилище. Подробную информацию см. в Справочнике функций - FILES().

Вывод

```
+-----------+---------------+-------+------+------+-----+---------+-------+
| IndexName | IndexKeysType | Field | Type | Null | Key | Default | Extra |
+-----------+---------------+-------+------+------+-----+---------+-------+
```

В следующей таблице описаны параметры, возвращаемые этим оператором.

Параметр	Описание
IndexName	Имя таблицы. Если вы просматриваете схему таблицы, хранящейся во внешнем источнике данных, этот параметр не возвращается.
IndexKeysType	Тип sort key таблицы. Если вы просматриваете схему таблицы, хранящейся во внешнем источнике данных, этот параметр не возвращается.
Field	Имя столбца.
Type	Тип данных столбца.
Null	Могут ли значения столбца быть NULL. • `yes`: указывает, что значения могут быть NULL. • `no`: указывает, что значения не могут быть NULL.
Key	Используется ли столбец в качестве sort key. • `true`: указывает, что столбец используется в качестве sort key. • `false`: указывает, что столбец не используется в качестве sort key.
Default	Значение по умолчанию для типа данных столбца. Если тип данных не имеет значения по умолчанию, возвращается NULL.

Extra	• Если вы просматриваете схему таблицы, хранящейся в вашем Selena cluster, это поле отображает следующую информацию о столбце:
	  ◦ Агрегатная функция, используемая столбцом, такая как `SUM` и `MIN`.
	  ◦ Создан ли bloom filter index для столбца. Если да, значение `Extra` будет `BLOOM_FILTER`.
	• Если вы просматриваете схему таблицы, хранящейся во внешних источниках данных, это поле отображает, является ли столбец partition column. Если столбец является partition column, значение `Extra` будет `partition key`.

> Примечание: Для информации о том, как materialized view отображается в выводе, см. Пример 2.

Примеры

Пример 1: Просмотр схемы `example_table`, хранящейся в вашем Selena cluster.

```
DESC example_table;
```

Или

```
DESC default_catalog.example_db.example_table;
```

Вывод предыдущих операторов следующий.

```
+-------+---------------+------+-------+---------+-------+
| Field | Type          | Null | Key   | Default | Extra |
+-------+---------------+------+-------+---------+-------+
| k1    | TINYINT       | Yes  | true  | NULL    |       |
| k2    | DECIMAL(10,2) | Yes  | true  | 10.5    |       |
| k3    | CHAR(10)      | Yes  | false | NULL    |       |
| v1    | INT           | Yes  | false | NULL    |       |
+-------+---------------+------+-------+---------+-------+
```

Пример 2: Просмотр схемы, типа sort key и materialized view таблицы `sales_records`, хранящейся в вашем Selena cluster. В следующем примере одна materialized view `store_amt` создана на основе `sales_records`.

```
DESC db1.sales_records ALL;

+---------------+---------------+-----------+--------+------+-------+------
---+-------+
| IndexName     | IndexKeysType | Field     | Type   | Null | Key   |
Default | Extra |
+---------------+---------------+-----------+--------+------+-------+------
---+-------+
| sales_records | DUP_KEYS      | record_id | INT    | Yes  | true  | NULL
|       |
|               |               | seller_id | INT    | Yes  | true  | NULL
|       |
|               |               | store_id  | INT    | Yes  | true  | NULL
|       |
|               |               | sale_date | DATE   | Yes  | false | NULL
| NONE  |
|               |               | sale_amt  | BIGINT | Yes  | false | NULL
| NONE  |
|               |               |           |        |      |       |
|       |
| store_amt     | AGG_KEYS      | store_id  | INT    | Yes  | true  | NULL
|       |
|               |               | sale_amt  | BIGINT | Yes  | false | NULL
| SUM   |
+---------------+---------------+-----------+--------+------+-------+------
---+-------+
```

Пример 3: Просмотр схемы `hive_table`, хранящейся в вашем Hive cluster.

```
DESC hive_catalog.hive_db.hive_table;

+-------+----------------+------+-------+---------+---------------+
| Field | Type           | Null | Key   | Default | Extra         |
+-------+----------------+------+-------+---------+---------------+
| id    | INT            | Yes  | false | NULL    |               |
| name  | VARCHAR(65533) | Yes  | false | NULL    |               |
| date  | DATE           | Yes  | false | NULL    | partition key |
+-------+----------------+------+-------+---------+---------------+
```

Пример 4: Просмотр схемы parquet файла `lineorder`, хранящегося в AWS S3.

> **ПРИМЕЧАНИЕ**

> Для файлов, хранящихся в удалённом хранилище, DESC возвращает только три поля: `Field`, `Type` и `Null`.

```
DESC FILES(
    "path" = "s3://inserttest/lineorder.parquet",
    "format" = "parquet",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
    "aws.s3.region" = "us-west-2"
);

+------------------+------------------+------+
| Field            | Type             | Null |
+------------------+------------------+------+
| lo_orderkey      | int              | YES  |
| lo_linenumber    | int              | YES  |
| lo_custkey       | int              | YES  |
| lo_partkey       | int              | YES  |
| lo_suppkey       | int              | YES  |
| lo_orderdate     | int              | YES  |
| lo_orderpriority | varchar(1048576) | YES  |
| lo_shippriority  | int              | YES  |
| lo_quantity      | int              | YES  |
| lo_extendedprice | int              | YES  |
| lo_ordtotalprice | int              | YES  |
| lo_discount      | int              | YES  |
| lo_revenue       | int              | YES  |
| lo_supplycost    | int              | YES  |
| lo_tax           | int              | YES  |
| lo_commitdate    | int              | YES  |
| lo_shipmode      | varchar(1048576) | YES  |
+------------------+------------------+------+
17 rows in set (0.05 sec)
```

Ссылки

- CREATE DATABASE
- SHOW CREATE DATABASE
- USE
- SHOW DATABASES
- DROP DATABASE

DROP INDEX

DROP INDEX используется для удаления указанного index в таблице. В настоящее время в этой версии поддерживается только bitmap index.

> **ПОДСКАЗКА**
>
> Эта операция требует привилегии ALTER для целевой таблицы. Вы можете следовать инструкциям в GRANT, чтобы предоставить эту привилегию.

Синтаксис:

```
DROP INDEX index_name ON [db_name.]table_name
```

DROP TABLE

DROP TABLE используется для удаления таблицы.

Синтаксис

```
DROP [TEMPORARY] TABLE [IF EXISTS] [db_name.]table_name [FORCE]
```

Примечание:

- После удаления таблицы вы можете использовать оператор RECOVER для восстановления таблицы в течение указанного периода (по умолчанию 1 день).
- Если выполняется DROP Table FORCE, таблица будет удалена напрямую и не может быть восстановлена без проверки наличия незавершённых активностей в базе данных. В целом, эта операция не рекомендуется.
- После удаления временные таблицы не могут быть восстановлены с помощью RECOVER.

Примеры

1. Удалить таблицу.

```
DROP TABLE my_table;
```

2. Если она существует, то удалить таблицу в указанной базе данных.

```
DROP TABLE IF EXISTS example_db.my_table;
```

3. Принудительно удалить таблицу и очистить её данные на диске.

```
DROP TABLE my_table FORCE;
```

Ссылки

- CREATE TABLE
- SHOW TABLES
- SHOW CREATE TABLE
- ALTER TABLE
- SHOW ALTER TABLE

REFRESH EXTERNAL TABLE

REFRESH EXTERNAL TABLE обновляет метаданные, кэшированные в Selena. Метаданные берутся из таблиц в data lakes. Эта команда используется в следующих сценариях:

- **External table**: При использовании Hive external table или Hudi external table для запроса данных в Apache Hive™ или Apache Hudi вы можете выполнить эту команду, чтобы обновить метаданные Hive таблицы или Hudi таблицы, кэшированные в Selena.
- **External catalog**: При использовании external catalog (кроме JDBC Catalog) для запроса данных в соответствующем источнике данных вы можете выполнить эту команду, чтобы обновить метаданные, кэшированные в Selena.

> **NOTE**
>
> Только пользователи с привилегией ALTER на соответствующую external table могут выполнять эту операцию.

Базовые концепции

- **Hive external table**: создается и хранится в Selena. Вы можете использовать её для запроса данных Hive.

- **Hudi external table**: создается и хранится в Selena. Вы можете использовать её для запроса данных Hudi.
- **Hive table**: создается и хранится в Hive.
- **Hudi table**: создается и хранится в Hudi.

Синтаксис и параметры

Ниже описаны синтаксис и параметры для различных случаев:

- External table

```
REFRESH EXTERNAL TABLE table_name
[PARTITION ('partition_name', ...)]
```

Параметр	Обязательный	Описание
table_name	Да	Имя Hive external table или Hudi external table.
partition_name	Нет	Имена partition Hive таблицы или Hudi таблицы. Указание этого параметра обновляет метаданные partition Hive таблицы и Hudi таблицы, кэшированные в Selena.

- External catalog

```
REFRESH EXTERNAL TABLE [external_catalog.][db_name.]table_name
[PARTITION ('partition_name', ...)]
```

Параметр	Обязательный	Описание
external_catalog	Нет	Имя external catalog, поддерживает Hive, Hudi, Delta Lake и MaxCompute (начиная с v1.5.2) catalogs.
db_name	Нет	Имя базы данных, где находится целевая таблица.
table_name	Да	Имя таблицы.
partition_name	Нет	Имена partition. Указание этого параметра обновляет метаданные partition целевой таблицы, кэшированные в Selena.

Примечания по использованию

Только пользователи с привилегией `ALTER_PRIV` могут выполнять эту команду для обновления метаданных Hive таблиц и Hudi таблиц, кэшированных в Selena.

Примеры

Примеры использования в различных случаях:

External table

Пример 1: Обновить кэшированные метаданные соответствующей Hive таблицы в Selena, указав external table `hive1`.

```
REFRESH EXTERNAL TABLE hive1;
```

Пример 2: Обновить кэшированные метаданные partition соответствующей Hudi таблицы в Selena, указав external table `hudi1` и partition в этой таблице.

```
REFRESH EXTERNAL TABLE hudi1
PARTITION ('date=2022-12-20', 'date=2022-12-21');
```

External catalog

Пример 1: Обновить метаданные `hive_table`, кэшированные в Selena.

```
REFRESH EXTERNAL TABLE hive_catalog.hive_db.hive_table;
```

Или

```
USE hive_catalog.hive_db;
REFRESH EXTERNAL TABLE hive_table;
```

Пример 2: Обновить метаданные второго уровня partition `p2` в `hive_table`, кэшированные в Selena.

```
USE hive_catalog.hive_db;
REFRESH EXTERNAL TABLE hive_table PARTITION ('p1=${date}/p2=${hour}');
```

Пример 3: Обновить метаданные partition `hudi_table`, кэшированные в Selena.

```
REFRESH EXTERNAL TABLE hudi_catalog.hudi_db.hudi_table
PARTITION ('date=2022-12-20', 'date=2022-12-21');
```

SELECT

SELECT запрашивает данные из одной или нескольких таблиц, views или materialized views. Оператор SELECT обычно состоит из следующих частей:

- WITH
- WHERE и операторы
- GROUP BY
- HAVING
- UNION
- INTERSECT
- EXCEPT/MINUS
- ORDER BY
- LIMIT
- OFFSET
- Joins
- Subqueries
- DISTINCT
- Alias

SELECT может работать как независимый оператор или как часть других операторов. Результат выполнения SELECT может быть использован в качестве входных данных для других операторов.

Синтаксис запросов Selena в основном соответствует стандарту SQL92. Вот краткое описание поддерживаемого использования SELECT.

> **NOTE**
>
> Для запроса данных из таблиц, views или materialized views во внутренней таблице Selena вы должны иметь привилегию SELECT на эти объекты. Для запроса данных из таблиц, views или materialized views во внешнем источнике данных вы должны иметь привилегию USAGE на соответствующий external catalog.

WITH

Часть, которая может быть добавлена перед оператором SELECT для определения псевдонима для сложного выражения, на которое ссылаются несколько раз внутри SELECT.

Аналогично CRATE VIEW, но имена таблиц и столбцов, определенные в части, не сохраняются после завершения запроса и не конфликтуют с именами в реальной таблице или VIEW.

Преимущества использования части WITH:

Удобно и легко поддерживать, сокращает дублирование внутри запросов.

Упрощает чтение и понимание SQL кода, абстрагируя наиболее сложные части запроса в отдельные блоки.

Примеры:

```
-- Define one subquery at the outer level, and another at the inner level
as part of the
-- initial stage of the UNION ALL query.

with t1 as (select 1),t2 as (select 2)
select * from t1 union all select * from t2;
```

Join

Join операции объединяют данные из двух или более таблиц и затем возвращают набор результатов из некоторых столбцов из некоторых из них.

Selena поддерживает self joins, cross joins, inner joins, outer joins, semi joins и anti joins. Outer joins включают left joins, right joins и full joins.

Синтаксис:

```
SELECT select_list FROM
table_or_subquery1 [INNER] JOIN table_or_subquery2 |
table_or_subquery1 {LEFT [OUTER] | RIGHT [OUTER] | FULL [OUTER]} JOIN
table_or_subquery2 |
table_or_subquery1 {LEFT | RIGHT} SEMI JOIN table_or_subquery2 |
table_or_subquery1 {LEFT | RIGHT} ANTI JOIN table_or_subquery2 |
[ ON col1 = col2 [AND col3 = col4 ...] |
USING (col1 [, col2 ...]) ]
[other_join_clause ...]
[ WHERE where_clauses ]
```

```
SELECT select_list FROM
table_or_subquery1, table_or_subquery2 [, table_or_subquery3 ...]
[other_join_clause ...]
WHERE
col1 = col2 [AND col3 = col4 ...]
```

```
SELECT select_list FROM
table_or_subquery1 CROSS JOIN table_or_subquery2
[other_join_clause ...]
[ WHERE where_clauses ]
```

Self Join

Selena поддерживает self-joins, которые являются самосоединениями. Например, разные столбцы одной и той же таблицы соединяются.

На самом деле нет специального синтаксиса для идентификации self-join. Условия с обеих сторон join в self-join исходят из одной и той же таблицы.

Нам нужно назначить им разные псевдонимы.

Примеры:

```
SELECT lhs.id, rhs.parent, lhs.c1, rhs.c2 FROM tree_data lhs, tree_data rhs
WHERE lhs.id = rhs.parent;
```

Cross Join

Cross join может производить много результатов, поэтому cross join следует использовать с осторожностью.

Даже если вам нужно использовать cross join, необходимо использовать условия фильтрации и убедиться, что возвращается меньше результатов. Пример:

```
SELECT * FROM t1, t2;

SELECT * FROM t1 CROSS JOIN t2;
```

Inner Join

Inner join — это наиболее известный и часто используемый join. Возвращает результаты из столбцов, запрошенных из двух похожих таблиц, соединенных, если столбцы обеих таблиц содержат одинаковое значение.

Если имя столбца обеих таблиц совпадает, нам нужно использовать полное имя (в форме table_name.column_name) или использовать псевдоним для имени столбца.

Примеры:

Следующие три запроса эквивалентны.

```
SELECT t1.id, c1, c2 FROM t1, t2 WHERE t1.id = t2.id;

SELECT t1.id, c1, c2 FROM t1 JOIN t2 ON t1.id = t2.id;

SELECT t1.id, c1, c2 FROM t1 INNER JOIN t2 ON t1.id = t2.id;
```

Outer Join

Outer join возвращает левую или правую таблицу или все строки обеих. Если нет соответствующих данных в другой таблице, устанавливается NULL. Пример:

```
SELECT * FROM t1 LEFT OUTER JOIN t2 ON t1.id = t2.id;

SELECT * FROM t1 RIGHT OUTER JOIN t2 ON t1.id = t2.id;

SELECT * FROM t1 FULL OUTER JOIN t2 ON t1.id = t2.id;
```

Эквивалентные и неэквивалентные join

Обычно пользователи используют чаще всего эквивалентные join, которые требуют, чтобы оператор условия join был знаком равенства.

Неэквивалентные join можно использовать с условиями join !=. Неэквивалентные join производят большое количество результатов и могут превысить лимит памяти во время вычисления.

Используйте с осторожностью. Неэквивалентные join поддерживают только inner join. Пример:

```
SELECT t1.id, c1, c2 FROM t1 INNER JOIN t2 ON t1.id = t2.id;

SELECT t1.id, c1, c2 FROM t1 INNER JOIN t2 ON t1.id > t2.id;
```

Semi Join

Left semi join возвращает только те строки в левой таблице, которые соответствуют данным в правой таблице, независимо от того, сколько строк соответствует данным в правой таблице.

Эта строка левой таблицы возвращается не более одного раза. Right semi join работает аналогично, за исключением того, что возвращаемые данные являются правой таблицей.

Примеры:

```
SELECT t1.c1, t1.c2, t1.c2 FROM t1 LEFT SEMI JOIN t2 ON t1.id = t2.id;
```

Anti Join

Left anti join возвращает только строки из левой таблицы, которые не соответствуют правой таблице.

Right anti join меняет это сравнение, возвращая только строки из правой таблицы, которые не соответствуют левой таблице. Пример:

```
SELECT t1.c1, t1.c2, t1.c2 FROM t1 LEFT ANTI JOIN t2 ON t1.id = t2.id;
```

Equi-join и Non-equi-join

Различные joins, поддерживаемые Selena, могут быть классифицированы как equi-joins и non-equi-joins в зависимости от условий join, указанных в joins.

Equi-joins	Self joins, cross joins, inner joins, outer joins, semi joins и anti joins
Non-equi-joins	cross joins, inner joins, left semi joins, left anti joins и outer joins

- Equi-joins

  Equi-join использует условие join, в котором два элемента join объединяются оператором `=`. Пример: `a JOIN b ON a.id = b.id`.

- Non-equi-joins

  Non-equi-join использует условие join, в котором два элемента join объединяются оператором сравнения, таким как `<`, `<=`, `>`, `>=` или `<>`. Пример: `a JOIN b ON a.id < b.id`. Non-equi-joins работают медленнее, чем equi-joins. Мы рекомендуем проявлять осторожность при использовании non-equi-joins.

  Следующие два примера показывают, как выполнять non-equi-joins:

  ```
  SELECT t1.id, c1, c2
  FROM t1
  INNER JOIN t2 ON t1.id < t2.id;

  SELECT t1.id, c1, c2
  FROM t1
  LEFT JOIN t2 ON t1.id > t2.id;
  ```

Join с частью USING

Начиная с v2.0.0, Selena поддерживает указание условий join через часть `USING` в дополнение к `ON`. Это помогает упростить equi-joins со столбцами одинакового имени.

Например: `SELECT * FROM t1 JOIN t2 USING (id)`.

Различия между версиями:

- **Версии до v2.0.0**

  `USING` рассматривается как синтаксический сахар и внутренне преобразуется в условие `ON`. Результат будет включать столбцы USING из обеих левой и правой таблиц как отдельные столбцы, и квалификаторы псевдонимов таблиц (например, `t1.id`) были разрешены при ссылке на столбцы USING.

  Пример:

  ```
  SELECT t1.id, t2.id FROM t1 JOIN t2 USING (id);  -- Returns two separate id columns
  ```

- **v2.0.0 и позже**

  Selena реализует семантику `USING` в соответствии со стандартом SQL. Ключевые особенности включают:

  - Поддерживаются все типы join, включая `FULL OUTER JOIN`.
  - Столбцы USING появляются как один объединенный столбец в результатах. Для FULL OUTER JOIN используется семантика `COALESCE(left.col, right.col)`.
  - Квалификаторы псевдонимов таблиц (например, `t1.id`) больше не поддерживаются для столбцов USING. Необходимо использовать неквалифицированные имена столбцов (например, `id`).
  - Для результата `SELECT *` порядок столбцов: `[USING columns, left non-USING columns, right non-USING columns]`.

  Пример:

  ```
  SELECT t1.id FROM t1 JOIN t2 USING (id);        -- ✗ Error: Column 'id' is ambiguous
  SELECT id FROM t1 JOIN t2 USING (id);           -- □ Correct: Returns a single coalesced 'id' column
  SELECT * FROM t1 FULL OUTER JOIN t2 USING (id); -- □ FULL OUTER JOIN is supported
  ```

Эти изменения приводят поведение Selena в соответствие с базами данных, совместимыми со стандартом SQL.

ASOF Join

ASOF Join — это тип временного или основанного на диапазоне join, обычно используемого в аналитике временных рядов. Он позволяет объединять две таблицы на основе равенства определенных ключей и условия неравенства по времени или полям последовательности, например, `t1.time >= t2.time`. ASOF Join выбирает наиболее недавнюю соответствующую строку из правой таблицы для каждой строки в левой таблице. Поддерживается начиная с v2.0.0.

В реальных сценариях аналитика, связанная с данными временных рядов, часто сталкивается со следующими проблемами:

- Несовпадение времени сбора данных (например, разные времена выборки датчиков)
- Небольшие расхождения между временем возникновения события и временем записи
- Необходимость найти ближайшую историческую запись для данного временной метки

Традиционные equi joins (INNER Join) часто приводят к значительной потере данных при обработке таких данных, в то время как inequality joins могут привести к проблемам производительности. ASOF Join был разработан для решения этих конкретных проблем.

ASOF Joins обычно используются в следующих случаях:

- **Анализ финансового рынка**
  - Сопоставление цен акций с объемами торговли
  - Согласование данных с разных рынков
  - Сопоставление эталонных данных для ценообразования производных инструментов
- **Обработка данных IoT**
  - Согласование потоков данных нескольких датчиков
  - Корреляция изменений состояния устройств
  - Интерполяция данных временных рядов
- **Анализ логов**
  - Корреляция системных событий с действиями пользователей
  - Сопоставление логов различных сервисов

- Анализ неисправностей и отслеживание проблем

Синтаксис:

```
SELECT [select_list]
FROM left_table [AS left_alias]
ASOF LEFT JOIN right_table [AS right_alias]
    ON equality_condition
    AND asof_condition
[WHERE ...]
[ORDER BY ...]
```

- `ASOF LEFT JOIN`: Выполняет неэквивалентный join на основе ближайшего соответствия по времени или последовательности. ASOF LEFT JOIN возвращает все строки из левой таблицы, заполняя несоответствующие строки правой таблицы значением NULL.
- `equality_condition`: Стандартное условие равенства (например, сопоставление символов ticker или идентификаторов).
- `asof_condition`: Условие диапазона, обычно записываемое как `left.time >= right.time`, указывающее на поиск наиболее недавних записей `right.time`, не превышающих `left.time`.

> **ПРИМЕЧАНИЕ**
>
> В `asof_condition` поддерживаются только типы DATE и DATETIME. И поддерживается только одно `asof_condition`.

Пример:

```
SELECT *
FROM holdings h ASOF LEFT JOIN prices p
ON h.ticker = p.ticker
AND h.when >= p.when
ORDER BY ALL;
```

Ограничения:

- В настоящее время поддерживаются только Inner Join (по умолчанию) и Left Outer Join.
- В `asof_condition` поддерживаются только типы DATE и DATETIME.
- Поддерживается только одно `asof_condition`.

ORDER BY

Часть ORDER BY оператора SELECT сортирует набор результатов, сравнивая значения из одного или нескольких столбцов.

ORDER BY — это трудоемкая и ресурсоемкая операция, поскольку все результаты должны быть отправлены на один узел для слияния, прежде чем результаты могут быть отсортированы. Сортировка потребляет больше ресурсов памяти, чем запрос без ORDER BY.

Поэтому, если вам нужны только первые `N` результатов из отсортированного набора результатов, вы можете использовать часть LIMIT, которая уменьшает использование памяти и сетевые накладные расходы. Если часть LIMIT не указана, по умолчанию возвращаются первые 65535 результатов.

Синтаксис:

```
ORDER BY <column_name>
    [ASC | DESC]
    [NULLS FIRST | NULLS LAST]
```

`ASC` указывает, что результаты должны быть возвращены в порядке возрастания. `DESC` указывает, что результаты должны быть возвращены в порядке убывания. Если порядок не указан, ASC (возрастание) является порядком по умолчанию. Пример:

```
select * from big_table order by tiny_column, short_column desc;
```

Порядок сортировки для значений NULL: `NULLS FIRST` указывает, что значения NULL должны быть возвращены перед значениями, отличными от NULL. `NULLS LAST` указывает, что значения NULL должны быть возвращены после значений, отличных от NULL.

Примеры:

```
select * from sales_record order by employee_id nulls first;
```

GROUP BY

Часть GROUP BY часто используется с агрегатными функциями. Столбцы, указанные в части GROUP BY, не будут участвовать в операции агрегирования.

Синтаксис

```
SELECT
...
aggregate_function() [ FILTER ( where boolean_expression ) ]
...
FROM ...
[ ... ]
GROUP BY [
    , ... |
    GROUPING SETS [, ...] (  groupSet [ , groupSet [ , ... ] ] ) |
    ROLLUP(expr  [ , expr [ , ... ] ]) |
    CUBE(expr  [ , expr [ , ... ] ])
    ]
[ ... ]
```

Параметры

- `FILTER` может использоваться вместе с агрегатными функциями. Только отфильтрованные строки будут участвовать в вычислении агрегатной функции.

  > **NOTE**
  >
  > - Часть FILTER поддерживается только в функциях AVG, COUNT, MAX, MIN, SUM, ARRAY_AGG и ARRAY_AGG_DISTINCT.
  > - Часть FILTER не поддерживается для COUNT DISTINCT.
  > - Когда указана часть FILTER, части ORDER BY не разрешены внутри функций ARRAY_AGG и ARRAY_AGG_DISTINCT.

- `GROUPING SETS`, `CUBE` и `ROLLUP` являются расширениями части GROUP BY. В части GROUP BY они могут использоваться для достижения групповой агрегации нескольких наборов. Результаты эквивалентны результату UNION нескольких частей GROUP BY.

Примеры

Пример 1: `FILTER`

Следующие два запроса эквивалентны.

```
SELECT
  COUNT(*) AS total_users,
  SUM(CASE WHEN gender = 'M' THEN 1 ELSE 0 END) AS male_users,
  SUM(CASE WHEN gender = 'F' THEN 1 ELSE 0 END) AS female_users
FROM users;
```

```
SELECT
  COUNT(*) AS total_users,
  COUNT(*) FILTER (WHERE gender = 'M') AS male_users,
  COUNT(*) FILTER (WHERE gender = 'F') AS female_users
FROM users;
```

Пример 2: `GROUPING SETS`, `CUBE` и `ROLLUP`

`ROLLUP(a,b,c)` эквивалентен следующей части `GROUPING SETS`.

```
GROUPING SETS (
(a,b,c),
(a,b  ),
(a    ),
(     )
)
```

`CUBE (a, b, c)` эквивалентен следующей части `GROUPING SETS`.

```
GROUPING SETS (
( a, b, c ),
( a, b    ),
( a,    c ),
( a       ),
(    b, c ),
(    b    ),
(       c ),
(         )
)
```

Тестирование на реальном наборе данных.

```
SELECT * FROM t;
+------+------+------+
| k1   | k2   | k3   |
+------+------+------+
| a    | A    |    1 |
| a    | A    |    2 |
| a    | B    |    1 |
| a    | B    |    3 |
| b    | A    |    1 |
| b    | A    |    4 |
| b    | B    |    1 |
| b    | B    |    5 |
+------+------+------+
8 rows in set (0.01 sec)

SELECT k1, k2, SUM(k3) FROM t
GROUP BY GROUPING SETS ( (k1, k2), (k2), (k1), ( ) );
+------+------+-----------+
| k1   | k2   | sum(`k3`) |
+------+------+-----------+
| b    | B    |         6 |
| a    | B    |         4 |
| a    | A    |         3 |
| b    | A    |         5 |
| NULL | B    |        10 |
| NULL | A    |         8 |
| a    | NULL |         7 |
| b    | NULL |        11 |
| NULL | NULL |        18 |
+------+------+-----------+
9 rows in set (0.06 sec)

SELECT k1, k2, GROUPING_ID(k1,k2), SUM(k3) FROM t
GROUP BY GROUPING SETS ((k1, k2), (k1), (k2), ());
+------+------+---------------+----------------+
| k1   | k2   | grouping_id(k1,k2) | sum(`k3`) |
+------+------+---------------+----------------+
| a    | A    |             0 |              3 |
| a    | B    |             0 |              4 |
| a    | NULL |             1 |              7 |
| b    | A    |             0 |              5 |
| b    | B    |             0 |              6 |
| b    | NULL |             1 |             11 |
| NULL | A    |             2 |              8 |
| NULL | B    |             2 |             10 |
```

```
| NULL | NULL |              3 |              18 |
+------+------+----------------+-----------------+
9 rows in set (0.02 sec)
```

HAVING

Часть HAVING не фильтрует данные строк в таблице, но фильтрует результаты агрегатных функций.

Как правило, HAVING используется с агрегатными функциями (такими как COUNT(), SUM(), AVG(), MIN(), MAX()) и частями GROUP BY.

Примеры:

```
select tiny_column, sum(short_column)
from small_table
group by tiny_column
having sum(short_column) = 1;
```

```
+-------------+---------------------+
|tiny_column  | sum('short_column') |
+-------------+---------------------+
|         2   |        1            |
+-------------+---------------------+

1 row in set (0.07 sec)
```

```
select tiny_column, sum(short_column)
from small_table
group by tiny_column
having tiny_column > 1;
```

```
+-------------+---------------------+
|tiny_column  | sum('short_column') |
+-------------+---------------------+
|      2      |          1          |
+-------------+---------------------+

1 row in set (0.07 sec)
```

LIMIT

Части LIMIT используются для ограничения максимального количества возвращаемых строк. Установка максимального количества возвращаемых строк может помочь Selena оптимизировать использование памяти.

Эта часть в основном используется в следующих сценариях:

Возвращает результат запроса top-N.

Подумайте о том, что включено в таблицу ниже.

Размер набора результатов запроса необходимо ограничить из-за большого объема данных в таблице или из-за того, что часть where не фильтрует слишком много данных.

Инструкции по использованию: Значение части LIMIT должно быть числовой литеральной константой.

Примеры:

```
mysql> select tiny_column from small_table limit 1;

+-------------+
|tiny_column  |
+-------------+
|     1       |
+-------------+

1 row in set (0.02 sec)
```

```
mysql> select tiny_column from small_table limit 10000;

+-------------+
|tiny_column  |
+-------------+
|      1      |
|      2      |
+-------------+

2 rows in set (0.01 sec)
```

OFFSET

Часть OFFSET заставляет набор результатов пропустить первые несколько строк и вернуть следующие результаты напрямую.

Набор результатов по умолчанию начинается со строки 0, поэтому offset 0 и без offset возвращают одинаковые результаты.

Как правило, части OFFSET необходимо использовать с частями ORDER BY и LIMIT, чтобы быть действительными.

Примеры:

```
mysql> select varchar_column from big_table order by varchar_column limit
3;

+----------------+
| varchar_column |
+----------------+
|    beijing     |
|    chongqing   |
|    tianjin     |
+----------------+

3 rows in set (0.02 sec)
```

```
mysql> select varchar_column from big_table order by varchar_column limit 1
offset 0;

+----------------+
|varchar_column  |
+----------------+
|     beijing    |
+----------------+

1 row in set (0.01 sec)
```

```
mysql> select varchar_column from big_table order by varchar_column limit 1
offset 1;

+----------------+
|varchar_column  |
+----------------+
|    chongqing   |
+----------------+

1 row in set (0.01 sec)
```

```
mysql> select varchar_column from big_table order by varchar_column limit 1
offset 2;

+----------------+
|varchar_column  |
+----------------+
|     tianjin    |
+----------------+

1 row in set (0.02 sec)
```

Примечание: Разрешается использовать синтаксис offset без order by, но offset не имеет смысла в этом случае.

В этом случае берется только значение limit, а значение offset игнорируется. Таким образом, без order by.

Offset превышает максимальное количество строк в наборе результатов и по-прежнему является результатом. Рекомендуется пользователям использовать offset с order by.

UNION

Объединяет результаты нескольких запросов.

Синтаксис:

```
query_1 UNION [DISTINCT | ALL] query_2
```

- DISTINCT (по умолчанию) возвращает только уникальные строки. UNION эквивалентен UNION DISTINCT.
- ALL объединяет все строки, включая дубликаты. Поскольку дедупликация требует много памяти, запросы, использующие UNION ALL, выполняются быстрее и потребляют меньше памяти. Для лучшей производительности используйте UNION ALL.

> **NOTE**
>
> Каждый оператор запроса должен возвращать одинаковое количество столбцов, и столбцы должны иметь совместимые типы данных.

Примеры:

Создайте таблицы `select1` и `select2`.

```
CREATE TABLE select1(
    id          INT,
    price       INT
    )
DISTRIBUTED BY HASH(id);

INSERT INTO select1 VALUES
    (1,2),
    (1,2),
    (2,3),
    (5,6),
    (5,6);

CREATE TABLE select2(
    id          INT,
    price       INT
    )
DISTRIBUTED BY HASH(id);

INSERT INTO select2 VALUES
    (2,3),
    (3,4),
    (5,6),
    (7,8);
```

Пример 1: Вернуть все идентификаторы в двух таблицах, включая дубликаты.

```
mysql> (select id from select1) union all (select id from select2) order by
id;

+------+
| id   |
+------+
|    1 |
|    1 |
|    2 |
|    2 |
|    3 |
|    5 |
|    5 |
|    5 |
|    7 |
+------+
11 rows in set (0.02 sec)
```

Пример 2: Вернуть все уникальные идентификаторы в двух таблицах. Следующие два оператора эквивалентны.

```
mysql> (select id from select1) union (select id from select2) order by id;

+------+
| id   |
+------+
|    1 |
|    2 |
|    3 |
|    5 |
|    7 |
+------+
6 rows in set (0.01 sec)

mysql> (select id from select1) union distinct (select id from select2)
order by id;
+------+
| id   |
+------+
|    1 |
|    2 |
|    3 |
|    5 |
|    7 |
+------+
5 rows in set (0.02 sec)
```

Пример 3: Вернуть первые три идентификатора среди всех уникальных идентификаторов в двух таблицах. Следующие два оператора эквивалентны.

```
mysql> (select id from select1) union distinct (select id from select2)
order by id
limit 3;
++------+
| id   |
+------+
|    1 |
|    2 |
|    3 |
+------+
4 rows in set (0.11 sec)

mysql> select * from (select id from select1 union distinct select id from
select2) as t1
order by id
limit 3;
+------+
| id   |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set (0.01 sec)
```

INTERSECT

Вычисляет пересечение результатов нескольких запросов, то есть результаты, которые появляются во всех наборах результатов. Эта часть возвращает только уникальные строки среди наборов результатов. Ключевое слово ALL не поддерживается.

Синтаксис:

```
query_1 INTERSECT [DISTINCT] query_2
```

> **NOTE**
>
> - INTERSECT эквивалентен INTERSECT DISTINCT.
> - Каждый оператор запроса должен возвращать одинаковое количество столбцов, и столбцы должны иметь совместимые типы данных.

Примеры:

Используются две таблицы из UNION.

Вернуть уникальные комбинации `(id, price)`, которые являются общими для обеих таблиц. Следующие два оператора эквивалентны.

```
mysql> (select id, price from select1) intersect (select id, price from
select2)
order by id;

+------+-------+
| id   | price |
+------+-------+
|    2 |     3 |
|    5 |     6 |
+------+-------+

mysql> (select id, price from select1) intersect distinct (select id, price
from select2)
order by id;

+------+-------+
| id   | price |
+------+-------+
|    2 |     3 |
|    5 |     6 |
+------+-------+
```

EXCEPT/MINUS

Возвращает уникальные результаты левого запроса, которых не существует в правом запросе. EXCEPT эквивалентен MINUS.

Синтаксис:

```
query_1 {EXCEPT | MINUS} [DISTINCT] query_2
```

> **NOTE**

> - EXCEPT эквивалентен EXCEPT DISTINCT. Ключевое слово ALL не поддерживается.
> - Каждый оператор запроса должен возвращать одинаковое количество столбцов, и столбцы должны иметь совместимые типы данных.

Примеры:

Используются две таблицы из UNION.

Вернуть уникальные комбинации `(id, price)` в `select1`, которые не могут быть найдены в `select2`.

```
mysql> (select id, price from select1) except (select id, price from
select2)
order by id;
+------+-------+
| id   | price |
+------+-------+
|    1 |     2 |
+------+-------+

mysql> (select id, price from select1) minus (select id, price from
select2)
order by id;
+------+-------+
| id   | price |
+------+-------+
|    1 |     2 |
+------+-------+
```

DISTINCT

Ключевое слово DISTINCT дедуплицирует набор результатов. Пример:

```
-- Returns the unique values from one column.
select distinct tiny_column from big_table limit 2;

-- Returns the unique combinations of values from multiple columns.
select distinct tiny_column, int_column from big_table limit 2;
```

DISTINCT можно использовать с агрегатными функциями (обычно функциями count), и count (distinct) используется для вычисления того, сколько различных комбинаций содержится в одном или нескольких столбцах.

```
-- Counts the unique values from one column.
select count(distinct tiny_column) from small_table;
```

```
+-------------------------------+
| count(DISTINCT 'tiny_column') |
+-------------------------------+
|              2                |
+-------------------------------+
1 row in set (0.06 sec)
```

```
-- Counts the unique combinations of values from multiple columns.
select count(distinct tiny_column, int_column) from big_table limit 2;
```

Selena поддерживает использование distinct в нескольких агрегатных функциях одновременно.

```
-- Count the unique value from multiple aggregation function separately.
select count(distinct tiny_column, int_column), count(distinct
varchar_column) from big_table;
```

Subquery

Подзапросы классифицируются на два типа с точки зрения релевантности:

- Некоррелированный подзапрос получает свои результаты независимо от своего внешнего запроса.
- Коррелированный подзапрос требует значений из своего внешнего запроса.

Некоррелированный подзапрос

Некоррелированные подзапросы поддерживают [NOT] IN и EXISTS.

Пример:

```
SELECT x FROM t1 WHERE x [NOT] IN (SELECT y FROM t2);

SELECT * FROM t1 WHERE (x,y) [NOT] IN (SELECT x,y FROM t2 LIMIT 2);

SELECT x FROM t1 WHERE EXISTS (SELECT y FROM t2 WHERE y = 1);
```

Начиная с v1.5.2, вы можете указать несколько полей в части WHERE `SELECT... FROM... WHERE... [NOT] IN`, например, `WHERE (x,y)` во втором операторе SELECT.

Коррелированный подзапрос

Связанные подзапросы поддерживают [NOT] IN и [NOT] EXISTS.

Пример:

```
SELECT * FROM t1 WHERE x [NOT] IN (SELECT a FROM t2 WHERE t1.y = t2.b);

SELECT * FROM t1 WHERE [NOT] EXISTS (SELECT a FROM t2 WHERE t1.y = t2.b);
```

Подзапросы также поддерживают скалярные подзапросы. Его можно разделить на нерелевантный скалярный подзапрос, связанный скалярный подзапрос и скалярный подзапрос в качестве параметра общей функции.

Примеры:

1. Некоррелированный скалярный подзапрос с предикатом = знак. Например, вывести информацию о человеке с самой высокой заработной платой.

   ```
   SELECT name FROM table WHERE salary = (SELECT MAX(salary) FROM table);
   ```

2. Некоррелированные скалярные подзапросы с предикатами `>`, `<` и т.д. Например, вывести информацию о людях, которым платят больше среднего.

   ```
   SELECT name FROM table WHERE salary > (SELECT AVG(salary) FROM table);
   ```

3. Связанные скалярные подзапросы. Например, вывести информацию о самой высокой зарплате для каждого отдела.

```
SELECT name FROM table a WHERE salary = (SELECT MAX(salary) FROM table
b WHERE b.Department= a.Department);
```

4. Скалярные подзапросы используются в качестве параметров обычных функций.

```
SELECT name FROM table WHERE salary = abs((SELECT MAX(salary) FROM
table));
```

Where и операторы

SQL операторы — это серия функций, используемых для сравнения и широко используемых в частях where операторов select.

Арифметический оператор

Арифметические операторы обычно появляются в выражениях, содержащих левый, правый и чаще всего левый операнды

+и-: можно использовать как единичный или как 2-арный оператор. При использовании в качестве единичного оператора, такого как +1, -2.5 или -col_name, что означает, что значение умножается на +1 или -1.

Таким образом, единичный оператор + возвращает неизменное значение, а единичный оператор - изменяет знаковые биты этого значения.

Пользователи могут наложить два единичных оператора, такие как +5 (возвращая положительное значение), -+2 или +2 (возвращая отрицательное значение), но пользователи не могут использовать два последовательных знака -.

Потому что -- интерпретируется как комментарий в следующем операторе (когда пользователь может использовать два знака, требуется пробел или скобка между двумя знаками, такими как -(-2) или - -2, что фактически приводит к +2).

Когда + или - является бинарным оператором, например, 2+2, 3+1.5 или col1+col2, это означает, что левое значение добавляется или вычитается из правого значения. Оба левое и правое значения должны быть числовыми типами.

и/: представляют умножение и деление соответственно. Операнды с обеих сторон должны быть типами данных. Когда два числа умножаются.

Меньшие операнды могут быть продвинуты при необходимости (например, SMALLINT в INT или BIGINT), и результат выражения будет продвинут до следующего большего типа.

Например, TINYINT, умноженный на INT, произведет результат типа BIGINT. Когда два числа умножаются, оба операнда и результаты выражения интерпретируются как типы DOUBLE, чтобы избежать потери точности.

Если пользователь хочет преобразовать результат выражения в другой тип, его необходимо преобразовать с помощью функции CAST.

%: Оператор модуляции. Возвращает остаток от деления левого операнда на правый операнд. Оба левый и правый операнды должны быть целыми числами.

&, |и ^: Побитовый оператор возвращает результат побитовых операций AND, побитовых OR, побитовых XOR над двумя операндами. Оба операнда требуют целочисленный тип.

Если типы двух операндов побитового оператора несовместимы, операнды меньшего типа продвигаются к операндам большего типа, и выполняются соответствующие побитовые операции.

Несколько арифметических операторов могут появляться в выражении, и пользователь может заключить соответствующее арифметическое выражение в скобки. Арифметические операторы обычно не имеют соответствующих математических функций для выражения тех же функций, что и арифметические операторы.

Например, у нас нет функции MOD() для представления оператора %. И наоборот, математические функции не имеют соответствующих арифметических операторов. Например, функция возведения в степень POW() не имеет соответствующего оператора возведения в степень **.

Пользователи могут узнать, какие арифметические функции мы поддерживаем, в разделе "Математические функции".

Оператор Between

В части where выражения могут сравниваться как с верхними, так и с нижними границами. Если выражение больше или равно нижней границе и меньше или равно верхней границе, результат сравнения истинен.

Синтаксис:

```
expression BETWEEN lower_bound AND upper_bound
```

Тип данных: Обычно выражение вычисляется в числовой тип, который также поддерживает другие типы данных. Если вы должны убедиться, что и нижняя, и верхняя границы являются сопоставимыми символами, вы можете использовать функцию cast().

Инструкции по использованию: Если операнд имеет тип string, обратите внимание, что длинная строка, начинающаяся с верхней границы, не будет соответствовать верхней границе, которая больше верхней границы. Например, "between'A'and'M' не будет соответствовать 'MJ'.

Если вам нужно убедиться, что выражение работает правильно, вы можете использовать такие функции, как upper(), lower(), substr(), trim().

Пример:

```
select c1 from t1 where month between 1 and 6;
```

Операторы сравнения

Операторы сравнения используются для сравнения двух значений. `=`, `!=`, `>=` применяются ко всем типам данных.

Операторы `<>` и `!=` эквивалентны, что указывает на то, что два значения не равны.

Оператор In

Оператор In сравнивает коллекцию VALUE и возвращает TRUE, если он может соответствовать любому из элементов в коллекции.

Параметры и коллекции VALUE должны быть сопоставимыми. Все выражения, использующие оператор IN, могут быть записаны как эквивалентные сравнения, связанные с OR, но синтаксис IN проще, точнее и легче для оптимизации Selena.

Примеры:

```
select * from small_table where tiny_column in (1,2);
```

Оператор Like

Этот оператор используется для сравнения со строкой. '_' (подчеркивание) соответствует одному символу, '%' соответствует нескольким символам. Параметр должен соответствовать полной строке. Как правило, размещение '%' в конце строки является более практичным.

Примеры:

```
mysql> select varchar_column from small_table where varchar_column like
'm%';

+----------------+
|varchar_column  |
+----------------+
|     milan      |
+----------------+

1 row in set (0.02 sec)
```

```
mysql> select varchar_column from small_table where varchar_column like
'm____';

+----------------+
| varchar_column |
+----------------+
|    milan       |
+----------------+

1 row in set (0.01 sec)
```

Логический оператор

Логические операторы возвращают значение BOOL, включая единичные и множественные операторы, каждый из которых обрабатывает параметры, которые являются выражениями, возвращающими значения BOOL. Поддерживаемые операторы:

AND: 2-арный оператор, оператор AND возвращает TRUE, если параметры слева и справа оба вычисляются как TRUE.

OR: 2-арный оператор, который возвращает TRUE, если один из параметров слева и справа вычисляется как TRUE. Если оба параметра FALSE, оператор OR возвращает FALSE.

NOT: Единичный оператор, результат инвертирования выражения. Если параметр TRUE, оператор возвращает FALSE; Если параметр FALSE, оператор возвращает TRUE.

Примеры:

```
mysql> select true and true;

+-------------------+
| (TRUE) AND (TRUE) |
+-------------------+
|         1         |
+-------------------+

1 row in set (0.00 sec)
```

```
mysql> select true and false;

+--------------------+
| (TRUE) AND (FALSE) |
+--------------------+
|         0          |
+--------------------+

1 row in set (0.01 sec)
```

```
mysql> select true or false;

+-------------------+
| (TRUE) OR (FALSE) |
+-------------------+
|        1          |
+-------------------+

1 row in set (0.01 sec)
```

```
mysql> select not true;

+----------+
| NOT TRUE |
+----------+
|     0    |
+----------+

1 row in set (0.01 sec)
```

Оператор регулярного выражения

Определяет, соответствует ли регулярное выражение. Использование стандартных регулярных выражений POSIX, '^' соответствует первой части строки, '$' соответствует концу строки.

"." соответствует любому отдельному символу, "*" соответствует нулю или более опциям, "+" соответствует одной или более опциям, "?" означает жадное представление и так далее. Регулярные выражения должны соответствовать полным значениям, а не только частям строк.

Если вы хотите сопоставить среднюю часть, передняя часть регулярного выражения может быть записана как '^.' или '.'. '^' и '$' обычно опускаются. Оператор RLIKE и оператор REGEXP являются синонимами.

Оператор '|' является опциональным оператором. Регулярные выражения с обеих сторон '|' должны удовлетворять только одному условию стороны. Оператор '|' и регулярные выражения с обеих сторон обычно должны быть заключены в ().

Примеры:

```
mysql> select varchar_column from small_table where varchar_column regexp
'(mi|MI).*';

+----------------+
| varchar_column |
+----------------+
|     milan      |
+----------------+

1 row in set (0.01 sec)
```

```
mysql> select varchar_column from small_table where varchar_column regexp
'm.*';

+----------------+
| varchar_column |
+----------------+
|     milan      |
+----------------+

1 row in set (0.01 sec)
```

Alias

Когда вы пишете имена таблиц, столбцов или выражений, содержащих столбцы в запросе, вы можете назначить им псевдоним. Псевдонимы обычно короче и лучше запоминаются, чем исходные имена.

Когда нужен псевдоним, вы можете просто добавить часть AS после имени таблицы, столбца и выражения в списке select или списке from. Ключевое слово AS является необязательным. Вы также можете указать псевдонимы непосредственно после исходного имени без использования AS.

Если псевдоним или другой идентификатор имеет то же имя, что и внутреннее ключевое слово Selena, вам нужно заключить имя в пару обратных кавычек, например, `rank`.

Псевдонимы чувствительны к регистру, но псевдонимы столбцов и псевдонимы выражений не чувствительны к регистру.

Примеры:

```
select tiny_column as name, int_column as sex from big_table;

select sum(tiny_column) as total_count from big_table;

select one.tiny_column, two.int_column from small_table one, <br/>
big_table two where one.tiny_column = two.tiny_column;
```

PIVOT

Эта функция поддерживается начиная с v1.5.2.

Операция PIVOT — это расширенная функция в SQL, которая позволяет вам преобразовывать строки в столбцы в таблице, что особенно полезно для создания сводных таблиц. Это пригодится при работе с отчетами баз данных или аналитикой, особенно когда вам нужно суммировать или категоризировать данные для представления.

На самом деле, PIVOT — это синтаксический сахар, который может упростить написание оператора запроса, такого как `sum(case when ... then ... end)`.

Синтаксис

```
pivot:
SELECT ...
FROM ...
PIVOT (
  aggregate_function(<expr>) [[AS] alias] [, aggregate_function(<expr>)
[[AS] alias] ...]
  FOR <pivot_column>
  IN (<pivot_value>)
)

pivot_column:
<column_name>
| (<column_name> [, <column_name> ...])

pivot_value:
<literal> [, <literal> ...]
| (<literal>, <literal> ...) [, (<literal>, <literal> ...)]
```

Параметры

В операции PIVOT вам нужно указать несколько ключевых компонентов:

- aggregate_function(): Агрегатная функция, такая как SUM, AVG, COUNT и т.д., используемая для суммирования данных.
- alias: Псевдоним для агрегированного результата, делающий результат более понятным.
- FOR pivot_column: Указывает имя столбца, для которого будет выполнено преобразование строк в столбцы.
- IN (pivot_value): Указывает конкретные значения pivot_column, которые будут преобразованы в столбцы.

Примеры

```
create table t1 (c0 int, c1 int, c2 int, c3 int);
SELECT * FROM t1 PIVOT (SUM(c1) AS sum_c1, AVG(c2) AS avg_c2 FOR c3 IN (1,
2, 3, 4, 5));
-- The result is equivalent to the following query:
SELECT SUM(CASE WHEN c3 = 1 THEN c1 ELSE NULL END) AS sum_c1_1,
       AVG(CASE WHEN c3 = 1 THEN c2 ELSE NULL END) AS avg_c2_1,
       SUM(CASE WHEN c3 = 2 THEN c1 ELSE NULL END) AS sum_c1_2,
       AVG(CASE WHEN c3 = 2 THEN c2 ELSE NULL END) AS avg_c2_2,
       SUM(CASE WHEN c3 = 3 THEN c1 ELSE NULL END) AS sum_c1_3,
       AVG(CASE WHEN c3 = 3 THEN c2 ELSE NULL END) AS avg_c2_3,
       SUM(CASE WHEN c3 = 4 THEN c1 ELSE NULL END) AS sum_c1_4,
       AVG(CASE WHEN c3 = 4 THEN c2 ELSE NULL END) AS avg_c2_4,
       SUM(CASE WHEN c3 = 5 THEN c1 ELSE NULL END) AS sum_c1_5,
       AVG(CASE WHEN c3 = 5 THEN c2 ELSE NULL END) AS avg_c2_5
FROM t1
GROUP BY c0;
```

EXCLUDE

Эта функция поддерживается начиная с версии 4.0.

Ключевое слово EXCLUDE используется для исключения указанных столбцов из результатов запроса, упрощая SQL операторы, когда определенные столбцы могут быть проигнорированы. Это особенно удобно при работе с таблицами, содержащими большое количество столбцов, избегая необходимости явно перечислять все столбцы, которые должны быть сохранены.

Синтаксис

```
SELECT
  * EXCLUDE (<column_name> [, <column_name> ...])
  | <table_alias>.* EXCLUDE (<column_name> [, <column_name> ...])
FROM ...
```

Параметры

- **`* EXCLUDE`** Выбирает все столбцы с подстановочным знаком `*`, за которым следует `EXCLUDE` и список имен столбцов для исключения.
- **`<table_alias>.* EXCLUDE`** Когда существует псевдоним таблицы, это позволяет исключить определенные столбцы из этой таблицы (должно использоваться с псевдонимом).
- **`<column_name>`** Имя(имена) столбца для исключения. Несколько столбцов разделяются запятыми. Столбцы должны существовать в таблице; в противном случае будет возвращена ошибка.

Примеры

Базовое использование

```
-- Create test_table.
CREATE TABLE test_table (
  id INT,
  name VARCHAR(50),
  age INT,
  email VARCHAR(100)
) DUPLICATE KEY(id);

-- Exclude a single column (age).
SELECT * EXCLUDE (age) FROM test_table;
-- Above is equivalent to:
SELECT id, name, email FROM test_table;

-- Exclude multiple columns (name, email).
SELECT * EXCLUDE (name, email) FROM test_table;
-- Above is equivalent to:
SELECT id, age FROM test_table;

-- Exclude columns using a table alias.
SELECT test_table.* EXCLUDE (email) FROM test_table;
-- Above is equivalent to:
SELECT id, name, age FROM test_table;
```

SHOW ALTER TABLE

SHOW ALTER TABLE показывает выполнение текущих операций ALTER TABLE, включая:

- Изменение столбцов.
- Оптимизацию схемы таблицы (начиная с v1.5.2), включая изменение метода bucketing и количества buckets.
- Создание и удаление rollup index.

Синтаксис

- Показать выполнение операций изменения столбцов или оптимизации схемы таблицы.

```
SHOW ALTER TABLE { COLUMN | OPTIMIZE } [FROM db_name] [WHERE
TableName|CreateTime|FinishTime|State] [ORDER BY] [LIMIT]
```

- Показать выполнение операций добавления или удаления rollup index.

```
SHOW ALTER TABLE ROLLUP [FROM db_name]
```

Параметры

- `{COLUMN | OPTIMIZE | ROLLUP}`:
  - Если указан `COLUMN`, эта команда показывает операции изменения столбцов.
  - Если указан `OPTIMIZE`, эта команда показывает операции оптимизации структуры таблицы.
  - Если указан `ROLLUP`, эта команда показывает операции добавления или удаления rollup index.
- `db_name`: опционально. Если `db_name` не указан, используется текущая база данных по умолчанию.

Примеры

1. Показать выполнение операций изменения столбцов, оптимизации схемы таблицы и создания или удаления rollup index в текущей базе данных.

```
SHOW ALTER TABLE COLUMN;
SHOW ALTER TABLE OPTIMIZE;
SHOW ALTER TABLE ROLLUP;
```

2. Показать выполнение операций, связанных с изменением столбцов, оптимизацией схемы таблицы и созданием или удалением rollup index в указанной базе данных.

```
SHOW ALTER TABLE COLUMN FROM example_db;
SHOW ALTER TABLE OPTIMIZE FROM example_db;
SHOW ALTER TABLE ROLLUP FROM example_db;
```

3. Показать выполнение последней операции изменения столбцов или оптимизации схемы таблицы в указанной таблице.

```
SHOW ALTER TABLE COLUMN WHERE TableName = "table1" ORDER BY CreateTime
DESC LIMIT 1;
SHOW ALTER TABLE OPTIMIZE WHERE TableName = "table1" ORDER BY
CreateTime DESC LIMIT 1;
```

Связанные команды

- CREATE TABLE
- ALTER TABLE
- SHOW TABLES
- SHOW CREATE TABLE

SHOW CREATE TABLE

Возвращает команду CREATE TABLE, которая использовалась для создания указанной таблицы.

> **ПРИМЕЧАНИЕ**
>
> В версиях до v1.5.2 команда SHOW CREATE TABLE требует наличия привилегии `SELECT_PRIV` на таблицу. Начиная с v1.5.2, команда SHOW CREATE TABLE требует наличия привилегии `SELECT` на таблицу.

Начиная с v1.5.2, вы можете использовать команду SHOW CREATE TABLE для просмотра команд CREATE TABLE для таблиц, управляемых внешним catalog и хранящихся в Apache Hive™, Apache Iceberg, Apache Hudi или Delta Lake.

Начиная с v1.5.2, Selena может автоматически устанавливать количество buckets (BUCKETS) при создании таблицы или добавлении partition. Вам больше не нужно вручную устанавливать количество buckets. Подробную информацию см. в разделе set the number of buckets.

- Если вы указали количество buckets при создании таблицы, вывод SHOW CREATE TABLE отобразит количество buckets.
- Если вы не указали количество buckets при создании таблицы, вывод SHOW CREATE TABLE не отобразит количество buckets. Вы можете выполнить SHOW PARTITIONS, чтобы просмотреть количество buckets для каждого partition.

В версиях до v1.5.2 требуется устанавливать количество buckets при создании таблицы. Поэтому SHOW CREATE TABLE по умолчанию отображает количество buckets.

Синтаксис

```
SHOW CREATE TABLE [db_name.]table_name
```

Параметры

Параметр	Обязательный	Описание
db_name	Нет	Имя базы данных. Если этот параметр не указан, по умолчанию возвращается команда CREATE TABLE для указанной таблицы в вашей текущей базе данных.
table_name	Да	Имя таблицы.

Вывод

```
+-----------+----------------+
| Table     | Create Table   |
+-----------+----------------+
```

В следующей таблице описаны параметры, возвращаемые этой командой.

Параметр	Описание
Table	Имя таблицы.
Create Table	Команда CREATE TABLE для таблицы.

Примеры

Количество buckets не указано

Создайте таблицу с именем `example_table` без указания количества buckets в DISTRIBUTED BY.

```
CREATE TABLE example_table
(
    k1 TINYINT,
    k2 DECIMAL(10, 2) DEFAULT "10.5",
    v1 CHAR(10) REPLACE,
    v2 INT SUM
)
ENGINE = olap
AGGREGATE KEY(k1, k2)
COMMENT "my first selena table"
DISTRIBUTED BY HASH(k1);
```

Выполните SHOW CREATE TABLE для отображения команды CREATE TABLE для `example_table`. Количество buckets не отображается в DISTRIBUTED BY. Обратите внимание, что если вы не указали PROPERTIES при создании таблицы, в выводе SHOW CREATE TABLE отображаются свойства по умолчанию.

```
SHOW CREATE TABLE example_table\G
*************************** 1. row ***************************
       Table: example_table
Create Table: CREATE TABLE `example_table` (
  `k1` tinyint(4) NULL COMMENT "",
  `k2` decimal64(10, 2) NULL DEFAULT "10.5" COMMENT "",
  `v1` char(10) REPLACE NULL COMMENT "",
  `v2` int(11) SUM NULL COMMENT ""
) ENGINE=OLAP
AGGREGATE KEY(`k1`, `k2`)
COMMENT "my first selena table"
DISTRIBUTED BY HASH(`k1`)
PROPERTIES (
"replication_num" = "3",
"in_memory" = "false",
"enable_persistent_index" = "true",
"replicated_storage" = "true",
"compression" = "LZ4"
);
```

Количество buckets указано

Создайте таблицу с именем `example_table1` с количеством buckets, установленным в 10 в DISTRIBUTED BY.

```
CREATE TABLE example_table1
(
    k1 TINYINT,
    k2 DECIMAL(10, 2) DEFAULT "10.5",
    v1 CHAR(10) REPLACE,
    v2 INT SUM
)
ENGINE = olap
AGGREGATE KEY(k1, k2)
COMMENT "my first selena table"
DISTRIBUTED BY HASH(k1) BUCKETS 10;
```

Выполните SHOW CREATE TABLE для отображения команды CREATE TABLE для `example_table`. Количество buckets (`BUCKETS 10`) отображается в DISTRIBUTED BY. Обратите внимание, что если вы не указали PROPERTIES при создании таблицы, в выводе SHOW CREATE TABLE отображаются свойства по умолчанию.

```
SHOW CREATE TABLE example_table1\G
*************************** 1. row ***************************
       Table: example_table1
Create Table: CREATE TABLE `example_table1` (
  `k1` tinyint(4) NULL COMMENT "",
  `k2` decimal64(10, 2) NULL DEFAULT "10.5" COMMENT "",
  `v1` char(10) REPLACE NULL COMMENT "",
  `v2` int(11) SUM NULL COMMENT ""
) ENGINE=OLAP
AGGREGATE KEY(`k1`, `k2`)
COMMENT "my first selena table"
DISTRIBUTED BY HASH(`k1`) BUCKETS 10
PROPERTIES (
"replication_num" = "3",
"in_memory" = "false",
"enable_persistent_index" = "true",
"replicated_storage" = "true",
"compression" = "LZ4"
);
```

Ссылки

- CREATE TABLE
- SHOW TABLES

- ALTER TABLE
- DROP TABLE

SHOW DELETE

SHOW DELETE запрашивает исторические операции DELETE, которые были успешно выполнены на таблицах Duplicate Key, Unique Key и Aggregate в указанной базе данных. Дополнительную информацию об удалении данных см. в разделе DELETE.

Обратите внимание, что эта команда не может использоваться для запроса операций DELETE, выполненных на таблицах Primary Key.

Синтаксис

```
SHOW DELETE [FROM <db_name>]
```

`db_name`: имя базы данных, необязательный параметр. Если этот параметр не указан, по умолчанию используется текущая база данных.

Возвращаемые поля:

- `TableName`: таблица, из которой удаляются данные.
- `PartitionName`: partition, из которого удаляются данные. Если таблица не является partitioned таблицей, отображается `*`.
- `CreateTime`: время создания задачи DELETE.
- `DeleteCondition`: указанное условие DELETE.
- `State`: статус задачи DELETE.

Примеры

Показать все исторические операции DELETE для `database`.

```
SHOW DELETE FROM database;

+------------+---------------+---------------------+-----------------+-----
-----+
| TableName  | PartitionName | CreateTime          | DeleteCondition |
State    |
+------------+---------------+---------------------+-----------------+-----
-----+
| mail_merge | *             | 2023-03-14 10:39:03 | name EQ "Peter" |
FINISHED |
+------------+---------------+---------------------+-----------------+-----
-----+
```

SHOW DYNAMIC PARTITION TABLES

SHOW DYNAMIC PARTITION TABLES используется для отображения статуса всех partitioned таблиц, для которых настроены свойства динамического партиционирования в базе данных.

Синтаксис

```
SHOW DYNAMIC PARTITION TABLES FROM <db_name>
```

Эта команда возвращает следующие поля:

- TableName: имя таблицы.
- Enable: включено ли динамическое партиционирование.
- TimeUnit: временная гранулярность для partitions.
- Start: начальное смещение динамического партиционирования.
- End: конечное смещение динамического партиционирования.
- Prefix: префикс имени partition.
- Buckets: количество buckets на partition.
- ReplicationNum: количество реплик для таблицы.
- StartOf: первый день каждой недели/месяца в зависимости от указанного TimeUnit.
- LastUpdateTime: время последнего обновления таблицы.
- LastSchedulerTime: время последнего планирования данных в таблице.
- State: статус таблицы.
- LastCreatePartitionMsg: сообщение о последней операции создания partition.
- LastDropPartitionMsg: сообщение о последней операции удаления partition.

Примеры

Отобразить статус всех partitioned таблиц, для которых настроены свойства динамического партиционирования в `db_test`.

```
SHOW DYNAMIC PARTITION TABLES FROM db_test;
```

SHOW FULL COLUMNS

SHOW FULL COLUMNS используется для отображения содержимого столбцов из указанных таблиц.

> **ПОДСКАЗКА**
>
> Эта операция не требует привилегий.

Синтаксис

```
SHOW FULL COLUMNS FROM <tbl_name>
```

Примеры

1. Просмотр содержимого столбцов из указанных таблиц.

   ```
   SHOW FULL COLUMNS FROM tbl;
   ```

SHOW INDEX

SHOW INDEX используется для отображения информации об индексах в таблице. В настоящее время поддерживается только bitmap index.

> **ПОДСКАЗКА**
>
> Эта операция не требует привилегий.

Синтаксис

```
SHOW INDEX[ES] FROM [db_name.]table_name [FROM database]
Or
SHOW KEY[S] FROM [db_name.]table_name [FROM database]
```

Примеры

1. Отобразить все индексы для указанной таблицы table_name:

```
SHOW INDEX FROM example_db.table_name;
```

SHOW PARTITIONS

SHOW PARTITIONS отображает информацию о partition, включая обычные partition и временные partition.

Синтаксис

```
SHOW [TEMPORARY] PARTITIONS FROM [db_name.]table_name [WHERE] [ORDER BY]
[LIMIT]
```

> NOTE
>
> Этот синтаксис поддерживает только таблицы Selena (`"ENGINE" = "OLAP"`). Начиная с версии v1.5.2, эта операция требует привилегию SELECT для указанной таблицы. Для версий v1.5.2 и более ранних эта операция требует привилегию SELECT_PRIV для указанной таблицы.

Описание возвращаемых полей

```
-- In shared-nothing cluster
+-------------+---------------+----------------+---------------------+-----
---------------+--------+--------------+-------+--------------------+------
---+----------------+---------------+---------------------+----------------
----------+----------+------------+----------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  |
VisibleVersionHash | State  | PartitionKey | Range | DistributionKey    |
Buckets | ReplicationNum | StorageMedium | CooldownTime        |
LastConsistencyCheckTime | DataSize | IsInMemory | RowCount |
+-------------+---------------+----------------+---------------------+-----
---------------+--------+--------------+-------+--------------------+------
---+----------------+---------------+---------------------+----------------
----------+----------+------------+----------+
-- In shared-data cluster
+-------------+---------------+----------------+----------------+----------
---+--------+--------------+-----------------------------------------------
-------------------------------------------------------+-----------------+-
--------+----------+-------------+----------+-----------------+------------
+-------+-------+-------+-------------+--------------------+---------------
-+
| PartitionId | PartitionName | CompactVersion | VisibleVersion |
NextVersion | State  | PartitionKey | Range | DistributionKey | Buckets |
DataSize | StorageSize | RowCount | EnableDataCache | AsyncWrite | AvgCS |
P50CS | MaxCS | DataVersion | VersionEpoch       | VersionTxnType |
+-------------+---------------+----------------+----------------+----------
---+--------+--------------+-----------------------------------------------
-------------------------------------------------------+-----------------+-
--------+----------+-------------+----------+-----------------+------------
+-------+-------+-------+-------------+--------------------+---------------
-+
```

Поле	Описание
PartitionId	ID partition.
PartitionName	Имя partition.
VisibleVersion	Номер версии последней успешной транзакции загрузки. Номер версии увеличивается на 1 с каждой успешной транзакцией загрузки.
CompactVersion	Номер версии последней успешной операции compaction.
VisibleVersionTime	Временная метка последней успешной транзакции загрузки.

Поле	Описание
VisibleVersionHash	Хеш-значение номера версии последней успешной транзакции загрузки.
State	Статус partition. Фиксированное значение: `Normal`.
PartitionKey	Ключ partition, состоящий из одного или нескольких столбцов partition.
Range	Диапазон partition, представляющий собой полуоткрытый интервал справа.
DistributionKey	Ключ bucket для hash bucketing.
Buckets	Количество bucket для partition.
ReplicationNum	Количество replica для каждого tablet в partition.
StorageMedium	Носитель хранения для данных в partition. Значение `HHD` обозначает жёсткие диски, а значение `SSD` обозначает твердотельные накопители.
CooldownTime	Время охлаждения для данных в partition. Если начальный носитель хранения — SSD, носитель переключается с SSD на HDD по истечении времени, указанного этим параметром. Формат: "yyyy-MM-dd HH:mm:ss".
LastConsistencyCheckTime	Время последней проверки согласованности. `NULL` означает, что проверка согласованности не выполнялась.
DataSize	Размер данных в partition.
IsInMemory	Хранятся ли все данные в partition в памяти.
StorageSize	Размер хранилища, фактически занимаемый partition. Только для shared-data cluster.
RowCount	Количество строк данных в partition. Только для shared-data cluster.
EnableDataCache	Включён ли Data Cache. Только для shared-data cluster.
AsyncWrite	Включена ли асинхронная запись. Только для shared-data cluster.
AvgCS	Средний Compaction Score partition. Только для shared-data cluster.
P50CS	P50 Compaction Score partition. Только для shared-data cluster.
MaxCS	Максимальный Compaction Score partition. Только для shared-data cluster.
DataVersion	Номер версии транзакций загрузки. Операции compaction не включены.
VersionEpoch	Эпоха partition. Система назначает эпоху версии при создании partition и изменяет её при обмене partition.
VersionTxnType	Тип транзакции, генерирующей текущую версию данных. Допустимые значения: `NORMAL` (обычная транзакция) и `REPLICATION` (репликация данных).

Примеры

1. Отобразить информацию обо всех обычных partition из указанной таблицы `site_access` в указанной базе данных `test`.

```
MySQL > show partitions from test.site_access\G
*************************** 1. row ***************************
            PartitionId: 20990
        PartitionName: p2019
        VisibleVersion: 1
    VisibleVersionTime: 2023-08-08 15:45:13
    VisibleVersionHash: 0
                State: NORMAL
            PartitionKey: datekey
                Range: [types: [DATE]; keys: [2019-01-01]; ..types:
[DATE]; keys: [2020-01-01]; )
        DistributionKey: site_id
                Buckets: 6
        ReplicationNum: 3
        StorageMedium: HDD
            CooldownTime: 9999-12-31 23:59:59
LastConsistencyCheckTime: NULL
                DataSize:  4KB
            IsInMemory: false
                RowCount: 3
1 row in set (0.00 sec)
```

2. Отобразить информацию обо всех временных partition из указанной таблицы `site_access` в указанной базе данных `test`.

```
SHOW TEMPORARY PARTITIONS FROM test.site_access;
```

3. Отобразить информацию об указанной partition `p1` указанной таблицы `site_access` в указанной базе данных `test`.

```
-- Обычная partition
SHOW PARTITIONS FROM test.site_access WHERE PartitionName = "p1";
-- Временная partition
SHOW TEMPORARY PARTITIONS FROM test.site_access WHERE PartitionName =
"p1";
```

4. Отобразить информацию о последней partition указанной таблицы `site_access` в указанной базе данных `test`.

```
-- Обычная partition
SHOW PARTITIONS FROM test.site_access ORDER BY PartitionId DESC LIMIT
1;
-- Временная partition
SHOW TEMPORARY PARTITIONS FROM test.site_access ORDER BY PartitionId
DESC LIMIT 1;
```

SHOW TABLES

SHOW TABLES отображает все таблицы в базе данных Selena или в базе данных внешнего источника данных, например, Hive, Iceberg, Hudi или Delta Lake.

> **ПРИМЕЧАНИЕ**
>
> Для просмотра таблиц во внешнем источнике данных необходимо иметь привилегию USAGE на внешний catalog, соответствующий этому источнику данных.

Синтаксис

```
SHOW TABLES [FROM <catalog_name>.<db_name>]
```

Параметры

Параметр	Обязательный	Описание
catalog_name	Нет	Имя внутреннего catalog или внешнего catalog. • Если вы не указываете этот параметр или устанавливаете значение `default_catalog`, возвращаются таблицы в базах данных Selena. • Если вы устанавливаете этот параметр в имя внешнего catalog, возвращаются таблицы в базах данных внешнего источника данных. Вы можете выполнить SHOW CATALOGS для просмотра внутренних и внешних catalogs.
db_name	Нет	Имя базы данных. Если не указано, по умолчанию используется текущая база данных.

Примеры

Пример 1: Просмотр таблиц в базе данных `example_db` из `default_catalog` после подключения к cluster Selena. Следующие два выражения эквивалентны.

```
show tables from example_db;
+----------------------------+
| Tables_in_example_db       |
+----------------------------+
| depts                      |
| depts_par                  |
| emps                       |
| emps2                      |
+----------------------------+

show tables from default_catalog.example_db;
+----------------------------+
| Tables_in_example_db       |
+----------------------------+
| depts                      |
| depts_par                  |
| emps                       |
| emps2                      |
+----------------------------+
```

Пример 2: Просмотр таблиц в текущей базе данных `example_db` после подключения к этой базе данных.

```
show tables;
+----------------------------+
| Tables_in_example_db       |
+----------------------------+
| depts                      |
| depts_par                  |
| emps                       |
| emps2                      |
+----------------------------+
```

Пример 3: Просмотр таблиц в базе данных `hudi_db` внешнего catalog `hudi_catalog`.

```
show tables from hudi_catalog.hudi_db;
+----------------------------+
| Tables_in_hudi_db          |
+----------------------------+
| hudi_sync_mor              |
| hudi_table1                |
+----------------------------+
```

В качестве альтернативы вы можете выполнить SET CATALOG для переключения на внешний catalog `hudi_catalog`, а затем выполнить `SHOW TABLES FROM hudi_db;`.

Ссылки

- SHOW CATALOGS: Просматривает все catalogs в cluster Selena.
- SHOW DATABASES: Просматривает все базы данных во внутреннем catalog или внешнем catalog.
- SET CATALOG: Переключается между catalogs.

SHOW TABLET

SHOW TABLET отображает информацию, связанную с tablet.

> **ПРИМЕЧАНИЕ**
>
> Для версии 3.0 и более поздних эта операция требует привилегию OPERATE на уровне SYSTEM и привилегию SELECT на уровне TABLE. Для версии 2.5 и более ранних эта операция требует привилегию ADMIN_PRIV.

Синтаксис

Запрос информации о tablets в таблице или partition

```
SHOW TABLET
FROM [<db_name>.]<table_name>
[PARTITION(<partition_name>, ...)]
[
WHERE [version = <version_number>]
    [[AND] backendid = <backend_id>]
    [[AND] STATE = "NORMAL"|"ALTER"|"CLONE"|"DECOMMISSION"]
]
[ORDER BY <field_name> [ASC | DESC]]
[LIMIT [<offset>,]<limit>]
```

Параметр	Обязательный	Описание
db_name	Нет	Имя базы данных. Если вы не указываете этот параметр, по умолчанию используется текущая база данных.
table_name	Да	Имя таблицы, из которой вы хотите запросить информацию о tablet. Вы должны указать этот параметр. В противном случае возвращается ошибка.
partition_name	Нет	Имя partition, из которого вы хотите запросить информацию о tablet.
version_number	Нет	Номер версии данных.
backend_id	Нет	ID BE, где находится replica tablet.
STATE	Нет	Статус replicas tablet. • `NORMAL`: Replica в нормальном состоянии. • `ALTER`: На replica выполняется Rollup или изменение схемы. • `CLONE`: Replica клонируется. (Replicas в этом состоянии недоступны для использования). • `DECOMMISSION`: Replica выводится из эксплуатации.
field_name	Нет	Поле, по которому сортируются результаты. Все поля, возвращаемые `SHOW TABLET FROM <table_name>`, можно сортировать. • Если вы хотите отобразить результаты в порядке возрастания, используйте `ORDER BY field_name ASC`. • Если вы хотите отобразить результаты в порядке убывания, используйте `ORDER BY field_name DESC`.

Параметр	Обязательный	Описание
offset	Нет	Количество tablets для пропуска из результатов. Например, `OFFSET 5` означает пропустить первые пять tablets. Значение по умолчанию: 0.
limit	Нет	Количество tablets для возврата. Например, `LIMIT 10` означает вернуть только 10 tablets. Если этот параметр не указан, возвращаются все tablets, соответствующие условиям фильтра.

Запрос информации об одном tablet

После получения всех ID tablets с помощью `SHOW TABLET FROM <table_name>`, вы можете запросить информацию об одном tablet.

```
SHOW TABLET <tablet_id>
```

Параметр	Обязательный	Описание
tablet_id	Да	ID Tablet

Описание возвращаемых полей

Запрос информации о tablets в таблице или partition

```
-- В shared-nothing cluster
+----------+-----------+-----------+------------+---------+-------------+--
-----------------+----------------------+------------------+--------------
--------+---------------+----------+----------+--------+------------------
------+--------------+------------------+--------------+----------+--------
--+-------------------+---------------+
| TabletId | ReplicaId | BackendId | SchemaHash | Version | VersionHash |
LstSuccessVersion | LstSuccessVersionHash | LstFailedVersion |
LstFailedVersionHash | LstFailedTime | DataSize | RowCount | State  |
LstConsistencyCheckTime | CheckVersion | CheckVersionHash | VersionCount |
PathHash | MetaUrl  | CompactionStatus  | DiskRootPath  |
+----------+-----------+-----------+------------+---------+-------------+--
-----------------+----------------------+------------------+--------------
--------+---------------+----------+----------+--------+------------------
------+--------------+------------------+--------------+----------+--------
--+-------------------+---------------+
-- В shared-data cluster
+----------+-----------+----------+----------+------------+
| TabletId | BackendId | DataSize | RowCount | MinVersion |
+----------+-----------+----------+----------+------------+
```

Поле	Описание
TabletId	ID таблицы.
ReplicaId	ID Replica.
BackendId	ID BE, где находится replica.
SchemaHash	Hash схемы (генерируется случайно).
Version	Номер версии данных.
VersionHash	Hash номера версии данных.
LstSuccessVersion	Последняя успешно загруженная версия.
LstSuccessVersionHash	Hash последней успешно загруженной версии.
LstFailedVersion	Версия последней неудачной загрузки. `-1` означает, что ни одна версия не была загружена с ошибкой.
LstFailedVersionHash	Hash последней неудачной версии.
LstFailedTime	Время последней неудачной загрузки. `NULL` означает, что ошибок загрузки не было.
DataSize	Размер данных tablet.

Поле	Описание
RowCount	Количество строк данных tablet.
State	Статус replica tablet.
LstConsistencyCheckTime	Время последней проверки согласованности. `NULL` означает, что проверка согласованности не выполнялась.
CheckVersion	Версия данных, на которой была выполнена проверка согласованности. `-1` означает, что ни одна версия не была проверена.
CheckVersionHash	Hash версии, на которой была выполнена проверка согласованности.
VersionCount	Общее количество версий данных.
PathHash	Hash директории, в которой хранится tablet.
MetaUrl	URL, используемый для запроса дополнительной мета-информации.
CompactionStatus	URL, используемый для запроса статуса уплотнения версий данных.
DiskRootPath	Диск, на котором находится replica.
MinVersion	Минимальная версия данных, сохраненная для tablet. Только для shared-data clusters.

Запрос информации о конкретном tablet

```
+--------+-----------+---------------+-----------+------+---------+--------
-----+---------+--------+-----------+
| DbName | TableName | PartitionName | IndexName | DbId | TableId |
PartitionId | IndexId | IsSync | DetailCmd |
+--------+-----------+---------------+-----------+------+---------+--------
-----+---------+--------+-----------+
```

Поле	Описание
DbName	Имя базы данных, к которой принадлежит tablet.
TableName	Имя таблицы, к которой принадлежит tablet.
PartitionName	Имя partition, к которому принадлежит tablet.
IndexName	Имя индекса.
DbId	ID базы данных.
TableId	ID таблицы.
PartitionId	ID partition.
IndexId	ID индекса.

Поле	Описание
IsSync	Согласованы ли данные на tablet с метаданными таблицы. `true` означает, что данные согласованы, и tablet находится в нормальном состоянии. `false` означает, что данные на tablet отсутствуют.
DetailCmd	URL, используемый для запроса дополнительной информации.

Примеры

Создайте таблицу `test_show_tablet` в базе данных `example_db`.

```
CREATE TABLE `test_show_tablet` (
  `k1` date NULL COMMENT "",
  `k2` datetime NULL COMMENT "",
  `k3` char(20) NULL COMMENT "",
  `k4` varchar(20) NULL COMMENT "",
  `k5` boolean NULL COMMENT "",
  `k6` tinyint(4) NULL COMMENT "",
  `k7` smallint(6) NULL COMMENT "",
  `k8` int(11) NULL COMMENT "",
  `k9` bigint(20) NULL COMMENT "",
  `k10` largeint(40) NULL COMMENT "",
  `k11` float NULL COMMENT "",
  `k12` double NULL COMMENT "",
  `k13` decimal128(27, 9) NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`k1`, `k2`, `k3`, `k4`, `k5`)
COMMENT "OLAP"
PARTITION BY RANGE(`k1`)
(PARTITION p20210101 VALUES [("2021-01-01"), ("2021-01-02")),
PARTITION p20210102 VALUES [("2021-01-02"), ("2021-01-03")),
PARTITION p20210103 VALUES [("2021-01-03"), ("2021-01-04")),
PARTITION p20210104 VALUES [("2021-01-04"), ("2021-01-05")),
PARTITION p20210105 VALUES [("2021-01-05"), ("2021-01-06")),
PARTITION p20210106 VALUES [("2021-01-06"), ("2021-01-07")),
PARTITION p20210107 VALUES [("2021-01-07"), ("2021-01-08")),
PARTITION p20210108 VALUES [("2021-01-08"), ("2021-01-09")),
PARTITION p20210109 VALUES [("2021-01-09"), ("2021-01-10")))
DISTRIBUTED BY HASH(`k1`, `k2`, `k3`);
```

- Пример 1: Запрос информации обо всех tablets в указанной таблице. Следующий пример извлекает информацию только об одном tablet из возвращаемой информации.

```
    mysql> show tablet from example_db.test_show_tablet\G
    *************************** 1. row ***************************
            TabletId: 9588955
            ReplicaId: 9588956
            BackendId: 10004
            SchemaHash: 0
                Version: 1
            VersionHash: 0
      LstSuccessVersion: 1
  LstSuccessVersionHash: 0
       LstFailedVersion: -1
   LstFailedVersionHash: 0
        LstFailedTime: NULL
            DataSize: 0B
            RowCount: 0
                State: NORMAL
LstConsistencyCheckTime: NULL
        CheckVersion: -1
    CheckVersionHash: 0
        VersionCount: 1
            PathHash: 0
             MetaUrl: http://172.26.92.141:8038/api/meta/header/9588955
    CompactionStatus: http://172.26.92.141:8038/api/compaction/show?
tablet_id=9588955
        DiskRootPath: /storage/disk
```

- Пример 2: Запрос информации о tablet 9588955.

```
    mysql> show tablet 9588955\G
    *************************** 1. row ***************************
    DbName: example_db
    TableName: test_show_tablet
    PartitionName: p20210103
    IndexName: test_show_tablet
        DbId: 11145
    TableId: 9588953
PartitionId: 9588946
    IndexId: 9588954
    IsSync: true
    DetailCmd: SHOW PROC
'/dbs/11145/9588953/partitions/9588946/9588954/9588955';
```

- Пример 3: Запрос информации о tablets в partition `p20210103`.

```
SHOW TABLET FROM test_show_tablet partition(p20210103);
```

- Пример 4: Вернуть информацию о 10 tablets.

```
    SHOW TABLET FROM test_show_tablet limit 10;
```

- Пример 5: Вернуть информацию о 10 tablets со смещением 5.

```
SHOW TABLET FROM test_show_tablet limit 5,10;
```

- Пример 6: Фильтровать tablets по `backendid`, `version` и `state`.

```
    SHOW TABLET FROM test_show_tablet
    WHERE backendid = 10004 and version = 1 and state = "NORMAL";
```

- Пример 7: Сортировать tablets по `version`.

```
    SHOW TABLET FROM table_name where backendid = 10004 order by
version;
```

- Пример 8: Вернуть информацию о tablets, имя индекса которых `test_show_tablet`.

```
SHOW TABLET FROM test_show_tablet where indexname = "test_show_tablet";
```

TRUNCATE TABLE

TRUNCATE TABLE используется для очистки указанных данных таблицы и partition.

Синтаксис:

```
TRUNCATE TABLE [db.]tbl[ PARTITION(PartitionName1, PartitionName2, ...)]
```

Примечание:

1. Этот оператор используется для очистки данных при сохранении таблиц или partitions.
2. В отличие от DELETE, этот оператор может только очищать указанные таблицы или partitions целиком, и условия фильтрации не могут быть добавлены.
3. В отличие от DELETE, использование этого метода для очистки данных не повлияет на производительность запросов.
4. Этот оператор напрямую удаляет данные. Удаленные данные не могут быть восстановлены.
5. Таблица, на которой вы выполняете эту операцию, должна находиться в состоянии NORMAL. Например, вы не можете выполнить TRUNCATE TABLE на таблице с выполняющимся SCHEMA CHANGE.

Примеры

1. Очистить таблицу `tbl` в `example_db`.

```
TRUNCATE TABLE example_db.tbl;
```

2. Очистить partitions `PartitionName1` и `PartitionName2` в таблице `tbl`.

```
TRUNCATE TABLE tbl PARTITION(PartitionName1, PartitionName2);
```

UPDATE

Обновляет строки в таблице с Primary Key.

Selena поддерживает оператор UPDATE начиная с версии 2.3, который поддерживает только UPDATE для одной таблицы и не поддерживает общие табличные выражения (CTE). Начиная с версии 3.0, Selena расширяет синтаксис для поддержки соединений нескольких таблиц и CTE. Если вам нужно соединить обновляемую таблицу с другими таблицами в базе данных, вы можете ссылаться на эти другие таблицы в предложении FROM или CTE. Начиная с версии 3.1, оператор UPDATE поддерживает частичные обновления в режиме столбцов, что подходит для сценариев с небольшим количеством столбцов, но большим количеством строк, что приводит к более высокой скорости обновления.

Эта команда требует привилегию UPDATE для таблицы, которую вы хотите обновить.

Примечания по использованию

При выполнении оператора UPDATE с несколькими таблицами Selena преобразует табличное выражение в предложении FROM оператора UPDATE в эквивалентный запрос JOIN. Поэтому убедитесь, что табличное выражение, которое вы указываете в предложении FROM оператора UPDATE, поддерживает это преобразование. Например, оператор UPDATE: 'UPDATE t0 SET v1=t1.v1 FROM t1 WHERE t0.pk = t1.pk;'. Табличное выражение в предложении FROM может быть преобразовано в 't0 JOIN t1 ON t0.pk=t1.pk;'. Selena сопоставляет строки данных для обновления на основе результирующего набора запроса JOIN. Возможно, что несколько строк в результирующем наборе совпадают с определенной строкой в обновляемой таблице. В этом сценарии эта строка обновляется на основе значения случайной строки из этих нескольких строк.

Вы можете использовать системную переменную `insert_timeout` для установки времени ожидания операций UPDATE.

Синтаксис

UPDATE для одной таблицы

Если строки данных обновляемой таблицы соответствуют условию WHERE, указанные столбцы этих строк данных получают новые значения.

```
[ WITH <with_query> [, ...] ]
UPDATE <table_name>
    SET <column_name> = <expression> [, ...]
    WHERE <where_condition>
```

UPDATE для нескольких таблиц

Результирующий набор от соединения нескольких таблиц сопоставляется с обновляемой таблицей. Если строки данных обновляемой таблицы совпадают с результирующим набором и соответствуют условию WHERE, указанные столбцы этих строк данных получают новые значения.

```
[ WITH <with_query> [, ...] ]
UPDATE <table_name>
    SET <column_name> = <expression> [, ...]
    [ FROM <from_item> [, ...] ]
    WHERE <where_condition>
```

Параметры

`with_query`

Одно или несколько CTE, на которые можно ссылаться по имени в операторе UPDATE. CTE - это временные результирующие наборы, которые могут улучшить читаемость сложных операторов.

`table_name`

Имя обновляемой таблицы.

`column_name`

Имя обновляемого столбца. Не может включать имя таблицы. Например, 'UPDATE t1 SET col = 1' недопустимо.

`expression`

Выражение, которое присваивает новые значения столбцу.

`from_item`

Одна или несколько других таблиц в базе данных. Эти таблицы могут быть соединены с обновляемой таблицей на основе условия, указанного в предложении WHERE. Значения строк в результирующем наборе используются для обновления значений указанных столбцов в совпадающих строках обновляемой таблицы. Например, если предложение FROM - `FROM t1 WHERE t0.pk = t1.pk`, Selena преобразует табличное выражение в предложении FROM в `t0 JOIN t1 ON t0.pk=t1.pk` при выполнении оператора UPDATE.

`where_condition`

Условие, на основе которого вы хотите обновить строки. Обновлены могут быть только строки, соответствующие условию WHERE. Этот параметр обязателен, так как он

помогает предотвратить случайное обновление всей таблицы. Если вы хотите обновить всю таблицу, вы можете использовать 'WHERE true'. Однако этот параметр не требуется для частичных обновлений в режиме столбцов.

Частичные обновления в режиме столбцов (начиная с v1.5.2)

Частичные обновления в режиме столбцов подходят для сценариев, где нужно обновить только небольшое количество столбцов, но большое количество строк. В таких сценариях включение режима столбцов обеспечивает более высокую скорость обновления. Например, в таблице со 100 столбцами, если обновляются только 10 столбцов (10% от общего числа) для всех строк, скорость обновления в режиме столбцов в 10 раз выше.

Системная переменная `partial_update_mode` управляет режимом частичных обновлений и поддерживает следующие значения:

- `auto` (по умолчанию): Система автоматически определяет режим частичных обновлений, анализируя оператор UPDATE и задействованные столбцы. Если выполняются следующие критерии, система автоматически использует режим столбцов:

  - Процент обновляемых столбцов по сравнению с общим количеством столбцов составляет менее 30%, и количество обновляемых столбцов меньше 4.
  - Оператор обновления не использует условие WHERE.

  В противном случае система не использует режим столбцов.

- `column`: Используется режим столбцов для частичных обновлений, который особенно подходит для частичных обновлений, затрагивающих небольшое количество столбцов и большое количество строк.

Вы можете использовать `EXPLAIN UPDATE xxx` для просмотра режима частичных обновлений.

Примеры

UPDATE для одной таблицы

Создайте таблицу `Employees` для записи информации о сотрудниках и вставьте пять строк данных в таблицу.

```
CREATE TABLE Employees (
    EmployeeID INT,
    Name VARCHAR(50),
    Salary DECIMAL(10, 2)
)
PRIMARY KEY (EmployeeID)
DISTRIBUTED BY HASH (EmployeeID)
PROPERTIES ("replication_num" = "3");

INSERT INTO Employees VALUES
    (1, 'John Doe', 5000),
    (2, 'Jane Smith', 6000),
    (3, 'Robert Johnson', 5500),
    (4, 'Emily Williams', 4500),
    (5, 'Michael Brown', 7000);
```

Если вам нужно повысить зарплату всем сотрудникам на 10%, вы можете выполнить следующий оператор:

```
UPDATE Employees
SET Salary = Salary * 1.1  -- Увеличить зарплату на 10%.
WHERE true;
```

Если вам нужно повысить зарплату на 10% сотрудникам с зарплатой ниже средней, вы можете выполнить следующий оператор:

```
UPDATE Employees
SET Salary = Salary * 1.1   -- Увеличить зарплату на 10%.
WHERE Salary < (SELECT AVG(Salary) FROM Employees);
```

Вы также можете использовать CTE для переписывания приведенного выше оператора для улучшения читаемости.

```
WITH AvgSalary AS (
    SELECT AVG(Salary) AS AverageSalary
    FROM Employees
)
UPDATE Employees
SET Salary = Salary * 1.1   -- Увеличить зарплату на 10%.
FROM AvgSalary
WHERE Employees.Salary < AvgSalary.AverageSalary;
```

UPDATE для нескольких таблиц

Создайте таблицу `Accounts` для записи информации об аккаунтах и вставьте три строки данных в таблицу.

```
CREATE TABLE Accounts (
    Accounts_id BIGINT NOT NULL,
    Name VARCHAR(26) NOT NULL,
    Sales_person VARCHAR(50) NOT NULL
)
PRIMARY KEY (Accounts_id)
DISTRIBUTED BY HASH (Accounts_id)
PROPERTIES ("replication_num" = "3");

INSERT INTO Accounts VALUES
    (1,'Acme Corporation','John Doe'),
    (2,'Acme Corporation','Robert Johnson'),
    (3,'Acme Corporation','Lily Swift');
```

Если вам нужно повысить зарплату на 10% сотрудникам в таблице `Employees`, которые управляют аккаунтами для Acme Corporation, вы можете выполнить следующий оператор:

```
UPDATE Employees
SET Salary = Salary * 1.1  -- Увеличить зарплату на 10%.
FROM Accounts
WHERE Accounts.name = 'Acme Corporation'
    AND Employees.Name = Accounts.Sales_person;
```

Вы также можете использовать CTE для переписывания приведенного выше оператора для улучшения читаемости.

```
WITH Acme_Accounts as (
    SELECT * from Accounts
    WHERE Accounts.name = 'Acme Corporation'
)
UPDATE Employees SET Salary = Salary * 1.1 -- Увеличить зарплату на 10%.
FROM Acme_Accounts
WHERE Employees.Name = Acme_Accounts.Sales_person;
```

AUTO_INCREMENT

Начиная с версии 3.0, Selena поддерживает атрибут столбца `AUTO_INCREMENT`, который может упростить управление данными. В этой теме рассматриваются сценарии применения, использование и функции атрибута столбца `AUTO_INCREMENT`.

Введение

Когда новая строка данных загружается в таблицу и значения для столбца `AUTO_INCREMENT` не указаны, Selena автоматически присваивает целочисленное значение для столбца `AUTO_INCREMENT` этой строки в качестве её уникального ID в таблице. Последующие значения для столбца `AUTO_INCREMENT` автоматически увеличиваются с определенным шагом, начиная с ID этой строки. Столбец `AUTO_INCREMENT` может использоваться для упрощения управления данными и ускорения некоторых запросов. Вот некоторые сценарии применения столбца `AUTO_INCREMENT`:

- Использование в качестве первичных ключей: Столбец `AUTO_INCREMENT` может использоваться в качестве первичного ключа для обеспечения уникального ID каждой строки и упрощения запросов и управления данными.
- Соединение таблиц: Когда несколько таблиц соединяются, столбец `AUTO_INCREMENT` может использоваться в качестве Join Key, что может ускорить запросы по сравнению с использованием столбца типа данных STRING, например, UUID.
- Подсчет количества различных значений в столбце с высокой кардинальностью: Столбец `AUTO_INCREMENT` может использоваться для представления столбца уникальных значений в словаре. По сравнению с прямым подсчетом различных значений STRING, подсчет различных целочисленных значений столбца `AUTO_INCREMENT` иногда может улучшить скорость запроса в несколько раз или даже в десятки раз.

Вам нужно указать столбец `AUTO_INCREMENT` в операторе CREATE TABLE. Типами данных столбца `AUTO_INCREMENT` должен быть BIGINT. Значение для столбца AUTO_INCREMENT

может быть присвоено неявно или указано явно. Оно начинается с 1 и увеличивается на 1 для каждой новой строки.

Основные операции

Указание столбца `AUTO_INCREMENT` при создании таблицы

Создайте таблицу с именем `test_tbl1` с двумя столбцами, `id` и `number`. Укажите столбец `number` как столбец `AUTO_INCREMENT`.

```
CREATE TABLE test_tbl1
(
    id BIGINT NOT NULL,
    number BIGINT NOT NULL AUTO_INCREMENT
)
PRIMARY KEY (id)
DISTRIBUTED BY HASH(id)
PROPERTIES("replicated_storage" = "true");
```

Присвоение значений для столбца `AUTO_INCREMENT`

Неявное присвоение значений

Когда вы загружаете данные в таблицу Selena, вам не нужно указывать значения для столбца `AUTO_INCREMENT`. Selena автоматически присваивает уникальные целочисленные значения для этого столбца и вставляет их в таблицу.

```
INSERT INTO test_tbl1 (id) VALUES (1);
INSERT INTO test_tbl1 (id) VALUES (2);
INSERT INTO test_tbl1 (id) VALUES (3),(4),(5);
```

Просмотр данных в таблице.

```
mysql > SELECT * FROM test_tbl1 ORDER BY id;
+------+--------+
| id   | number |
+------+--------+
|    1 |      1 |
|    2 |      2 |
|    3 |      3 |
|    4 |      4 |
|    5 |      5 |
+------+--------+
5 rows in set (0.02 sec)
```

Когда вы загружаете данные в таблицу Selena, вы также можете указать значения как `DEFAULT` для столбца `AUTO_INCREMENT`. Selena автоматически присваивает уникальные целочисленные значения для этого столбца и вставляет их в таблицу.

```
INSERT INTO test_tbl1 (id, number) VALUES (6, DEFAULT);
```

Просмотр данных в таблице.

```
mysql > SELECT * FROM test_tbl1 ORDER BY id;
+------+--------+
| id   | number |
+------+--------+
|    1 |      1 |
|    2 |      2 |
|    3 |      3 |
|    4 |      4 |
|    5 |      5 |
|    6 |      6 |
+------+--------+
6 rows in set (0.02 sec)
```

В реальном использовании следующий результат может быть возвращен при просмотре данных в таблице. Это происходит потому, что Selena не может гарантировать, что значения для столбца `AUTO_INCREMENT` строго монотонны. Но Selena может гарантировать, что значения примерно увеличиваются в хронологическом порядке. Для получения дополнительной информации см. Монотонность.

```
mysql > SELECT * FROM test_tbl1 ORDER BY id;
+------+--------+
| id   | number |
+------+--------+
|    1 |      1 |
|    2 | 100001 |
|    3 | 200001 |
|    4 | 200002 |
|    5 | 200003 |
|    6 | 200004 |
+------+--------+
6 rows in set (0.01 sec)
```

Явное указание значений

Вы также можете явно указать значения для столбца `AUTO_INCREMENT` и вставить их в таблицу.

```
INSERT INTO test_tbl1 (id, number) VALUES (7, 100);

-- просмотр данных в таблице.

mysql > SELECT * FROM test_tbl1 ORDER BY id;
+------+--------+
| id   | number |
+------+--------+
|    1 |      1 |
|    2 | 100001 |
|    3 | 200001 |
|    4 | 200002 |
|    5 | 200003 |
|    6 | 200004 |
|    7 |    100 |
+------+--------+
7 rows in set (0.01 sec)
```

Более того, явное указание значений не влияет на последующие значения, генерируемые Selena для вновь вставленных строк данных.

```
INSERT INTO test_tbl1 (id) VALUES (8);

-- просмотр данных в таблице.

mysql > SELECT * FROM test_tbl1 ORDER BY id;
+------+--------+
| id   | number |
+------+--------+
|    1 |      1 |
|    2 | 100001 |
|    3 | 200001 |
|    4 | 200002 |
|    5 | 200003 |
|    6 | 200004 |
|    7 |    100 |
|    8 |      2 |
+------+--------+
8 rows in set (0.01 sec)
```

ПРИМЕЧАНИЕ

Мы рекомендуем не использовать неявно присвоенные значения и явно указанные значения для столбца `AUTO_INCREMENT` одновременно. Потому что указанные значения могут быть такими же, как значения, сгенерированные Selena, что нарушает глобальную уникальность автоинкрементных ID.

Основные функции

Уникальность

В общем случае Selena гарантирует, что значения для столбца `AUTO_INCREMENT` глобально уникальны в таблице. Мы рекомендуем не присваивать неявно и явно указывать значения для столбца `AUTO_INCREMENT` одновременно. Если вы это сделаете, это может нарушить глобальную уникальность автоинкрементных ID. Вот простой пример: Создайте таблицу с именем `test_tbl2` с двумя столбцами, `id` и `number`. Укажите столбец `number` как столбец `AUTO_INCREMENT`.

```
CREATE TABLE test_tbl2
(
    id BIGINT NOT NULL,
    number BIGINT NOT NULL AUTO_INCREMENT
 )
PRIMARY KEY (id)
DISTRIBUTED BY HASH(id)
PROPERTIES("replicated_storage" = "true");
```

Неявно присвойте и явно укажите значения для столбца `AUTO_INCREMENT` `number` в таблице `test_tbl2`.

```
INSERT INTO test_tbl2 (id, number) VALUES (1, DEFAULT);
INSERT INTO test_tbl2 (id, number) VALUES (2, 2);
INSERT INTO test_tbl2 (id) VALUES (3);
```

Запросите таблицу `test_tbl2`.

```
mysql > SELECT * FROM test_tbl2 ORDER BY id;
+------+--------+
| id   | number |
+------+--------+
|    1 |      1 |
|    2 |      2 |
|    3 | 100001 |
+------+--------+
3 rows in set (0.08 sec)
```

Монотонность

Для улучшения производительности выделения автоинкрементных ID, BE кэшируют некоторые автоинкрементные ID локально. В этой ситуации Selena не может гарантировать, что значения для столбца `AUTO_INCREMENT` строго монотонны. Можно гарантировать только то, что значения примерно увеличиваются в хронологическом порядке.

> **ПРИМЕЧАНИЕ**

> Количество автоинкрементных ID, кэшируемых BE, определяется динамическим параметром FE `auto_increment_cache_size`, который по умолчанию равен `100,000`. Вы можете изменить значение, используя `ADMIN SET FRONTEND CONFIG ("auto_increment_cache_size" = "xxx");`

Например, кластер Selena имеет один узел FE и два узла BE. Создайте таблицу с именем `test_tbl3` и вставьте пять строк данных следующим образом:

```
CREATE TABLE test_tbl3
(
    id BIGINT NOT NULL,
    number BIGINT NOT NULL AUTO_INCREMENT
)
PRIMARY KEY (id)
DISTRIBUTED BY HASH(id)
PROPERTIES("replicated_storage" = "true");

INSERT INTO test_tbl3 VALUES (1, DEFAULT);
INSERT INTO test_tbl3 VALUES (2, DEFAULT);
INSERT INTO test_tbl3 VALUES (3, DEFAULT);
INSERT INTO test_tbl3 VALUES (4, DEFAULT);
INSERT INTO test_tbl3 VALUES (5, DEFAULT);
```

Автоинкрементные ID в таблице `test_tbl3` не увеличиваются монотонно, потому что два узла BE кэшируют автоинкрементные ID, [1, 100000] и [100001, 200000] соответственно. Когда данные загружаются с использованием нескольких операторов INSERT, данные отправляются на разные узлы BE, которые выделяют автоинкрементные ID независимо. Поэтому нельзя гарантировать, что автоинкрементные ID строго монотонны.

```
mysql > SELECT * FROM test_tbl3 ORDER BY id;
+------+--------+
| id   | number |
+------+--------+
|    1 |      1 |
|    2 | 100001 |
|    3 | 200001 |
|    4 |      2 |
|    5 | 100002 |
+------+--------+
5 rows in set (0.07 sec)
```

Частичные обновления и столбец AUTO_INCREMENT

В этом разделе объясняется, как обновить только несколько указанных столбцов в таблице, содержащей столбец `AUTO_INCREMENT`.

> **ПРИМЕЧАНИЕ**
>
> В настоящее время только таблицы с Primary Key поддерживают частичные обновления.

Столбец `AUTO_INCREMENT` является первичным ключом

Вам нужно указать первичный ключ во время частичных обновлений. Поэтому, если столбец `AUTO_INCREMENT` является первичным ключом или частью первичного ключа, поведение пользователя для частичных обновлений точно такое же, как когда столбец `AUTO_INCREMENT` не определен.

1. Создайте таблицу `test_tbl4` в базе данных `example_db` и вставьте одну строку данных.

```
-- Создать таблицу.
CREATE TABLE test_tbl4
(
    id BIGINT AUTO_INCREMENT,
    name BIGINT NOT NULL,
    job1 BIGINT NOT NULL,
    job2 BIGINT NOT NULL
)
PRIMARY KEY (id, name)
DISTRIBUTED BY HASH(id)
PROPERTIES("replicated_storage" = "true");

-- Подготовленные данные.
mysql > INSERT INTO test_tbl4 (id, name, job1, job2) VALUES (0, 0, 1,
1);
Query OK, 1 row affected (0.04 sec)
{'label':'insert_6af28e77-7d2b-11ed-af6e-02424283676b',
'status':'VISIBLE', 'txnId':'152'}

-- Запросить таблицу.
mysql > SELECT * FROM test_tbl4 ORDER BY id;
+------+------+------+------+
| id   | name | job1 | job2 |
+------+------+------+------+
|    0 |    0 |    1 |    1 |
+------+------+------+------+
1 row in set (0.01 sec)
```

2. Подготовьте CSV-файл **my_data4.csv** для обновления таблицы `test_tbl4`. CSV-файл включает значения для столбца `AUTO_INCREMENT` и не включает значения для столбца `job1`. Первичный ключ первой строки уже существует в таблице `test_tbl4`, в то время как первичный ключ второй строки не существует в таблице.

```
0,0,99
1,1,99
```

3. Запустите задачу Stream Load и используйте CSV-файл для обновления таблицы `test_tbl4`.

```
curl --location-trusted -u <username>:<password> -H "label:1" \
    -H "column_separator:," \
    -H "partial_update:true" \
    -H "columns:id,name,job2" \
    -T my_data4.csv -XPUT \
    http://<fe_host>:
<fe_http_port>/api/example_db/test_tbl4/_stream_load
```

4. Запросите обновленную таблицу. Первая строка данных уже существует в таблице `test_tbl4`, и значение для столбца `job1` остается неизменным. Вторая строка данных вновь вставлена, и поскольку значение по умолчанию для столбца `job1` не указано, фреймворк частичного обновления непосредственно устанавливает значение для этого столбца в `0`.

```
mysql > SELECT * FROM test_tbl4 ORDER BY id;
+------+------+------+------+
| id   | name | job1 | job2 |
+------+------+------+------+
|    0 |    0 |    1 |   99 |
|    1 |    1 |    0 |   99 |
+------+------+------+------+
2 rows in set (0.01 sec)
```

Столбец `AUTO_INCREMENT` не является первичным ключом

Если столбец `AUTO_INCREMENT` не является первичным ключом или частью первичного ключа, и автоинкрементные ID не предоставлены в задаче Stream Load, происходят следующие ситуации:

- Если строка уже существует в таблице, Selena не обновляет автоинкрементный ID.
- Если строка вновь загружается в таблицу, Selena генерирует новый автоинкрементный ID.

Эта функция может использоваться для построения словарной таблицы для быстрого вычисления различных значений STRING.

1. В базе данных `example_db` создайте таблицу `test_tbl5` и укажите столбец `job1` как столбец `AUTO_INCREMENT` и вставьте строку данных в таблицу `test_tbl5`.

```
-- Создать таблицу.
CREATE TABLE test_tbl5
(
    id BIGINT NOT NULL,
    name BIGINT NOT NULL,
    job1 BIGINT NOT NULL AUTO_INCREMENT,
    job2 BIGINT NOT NULL
)
PRIMARY KEY (id, name)
DISTRIBUTED BY HASH(id)
PROPERTIES("replicated_storage" = "true");

-- Подготовить данные.
mysql > INSERT INTO test_tbl5 VALUES (0, 0, -1, -1);
Query OK, 1 row affected (0.04 sec)
{'label':'insert_458d9487-80f6-11ed-ae56-aa528ccd0ebf',
'status':'VISIBLE', 'txnId':'94'}

-- Запросить таблицу.
mysql > SELECT * FROM test_tbl5 ORDER BY id;
+------+------+------+------+
| id   | name | job1 | job2 |
+------+------+------+------+
|    0 |    0 |   -1 |   -1 |
+------+------+------+------+
1 row in set (0.01 sec)
```

2. Подготовьте CSV-файл **my_data5.csv** для обновления таблицы `test_tbl5`. CSV-файл не содержит значений для столбца `AUTO_INCREMENT` `job1`. Первичный ключ первой строки уже существует в таблице, в то время как первичные ключи второй и третьей строк не существуют.

```
0,0,99
1,1,99
2,2,99
```

3. Запустите задачу Stream Load для загрузки данных из CSV-файла в таблицу `test_tbl5`.

```
curl --location-trusted -u <username>:<password> -H "label:2" \
    -H "column_separator:," \
    -H "partial_update:true" \
    -H "columns: id,name,job2" \
    -T my_data5.csv -XPUT \
    http://<fe_host>:
<fe_http_port>/api/example_db/test_tbl5/_stream_load
```

4. Запросите обновленную таблицу. Первая строка данных уже существует в таблице `test_tbl5`, поэтому столбец `AUTO_INCREMENT` `job1` сохраняет свое исходное значение. Вторая и третья строки данных вновь вставлены, поэтому Selena генерирует новые значения для столбца `AUTO_INCREMENT` `job1`.

```
mysql > SELECT * FROM test_tbl5 ORDER BY id;
+------+------+--------+------+
| id   | name | job1   | job2 |
+------+------+--------+------+
|    0 |    0 |     -1 |   99 |
|    1 |    1 |      1 |   99 |
|    2 |    2 | 100001 |   99 |
+------+------+--------+------+
3 rows in set (0.01 sec)
```

Ограничения

- При создании таблицы со столбцом `AUTO_INCREMENT` необходимо установить `'replicated_storage' = 'true'`, чтобы гарантировать, что все реплики имеют одинаковые автоинкрементные ID.
- Каждая таблица может иметь только один столбец `AUTO_INCREMENT`.
- Тип данных столбца `AUTO_INCREMENT` должен быть BIGINT.
- Столбец `AUTO_INCREMENT` должен быть `NOT NULL` и не иметь значения по умолчанию.
- Вы можете удалять данные из таблицы с Primary Key со столбцом `AUTO_INCREMENT`. Однако, если столбец `AUTO_INCREMENT` не является первичным ключом или частью первичного ключа, вам нужно учитывать следующие ограничения при удалении данных в следующих сценариях:

- Во время операции DELETE также существует задача загрузки для частичных обновлений, которая содержит только операции UPSERT. Если операции UPSERT и DELETE затрагивают одну и ту же строку данных, и операция UPSERT выполняется после операции DELETE, операция UPSERT может не вступить в силу.
- Существует задача загрузки для частичных обновлений, которая включает несколько операций UPSERT и DELETE для одной и той же строки данных. Если определенная операция UPSERT выполняется после операции DELETE, операция UPSERT может не вступить в силу.

- Добавление атрибута `AUTO_INCREMENT` с помощью ALTER TABLE не поддерживается.

- Начиная с версии 3.1, режим shared-data Selena поддерживает атрибут `AUTO_INCREMENT`.

- Selena не поддерживает указание начального значения и размера шага для столбца `AUTO_INCREMENT`.

Ключевые слова

AUTO_INCREMENT, AUTO INCREMENT

3.8 View

ALTER VIEW

ALTER VIEW изменяет определение представления.

Синтаксис

```
ALTER VIEW
[db_name.]view_name
(column1[ COMMENT "col comment"][, column2, ...])
AS query_stmt
```

Примечание:

1. Представление является логическим, данные не хранятся на физическом носителе. Представление будет использоваться как подзапрос в выражении при выполнении

запроса. Таким образом, изменение определения представления эквивалентно изменению query_stmt.
2. query_stmt - это произвольный поддерживаемый SQL.

Примеры

Изменить `example_view` в `example_db`.

```
ALTER VIEW example_db.example_view
(
c1 COMMENT "column 1",
c2 COMMENT "column 2",
c3 COMMENT "column 3"
)
AS SELECT k1, k2, SUM(v1)
FROM example_table
GROUP BY k1, k2
```

CREATE VIEW

CREATE VIEW создает логическое представление.

Представление, или логическое представление, - это виртуальная таблица, данные которой получаются из запроса к другим существующим физическим таблицам. Таким образом, представление не использует физическое хранилище, и все запросы к представлению эквивалентны подзапросам выражения запроса, использованного для создания представления.

Информацию о материализованных представлениях, поддерживаемых Selena, см. в разделах Синхронные материализованные представления и Асинхронные материализованные представления.

Начиная с версии v1.5.2, Selena поддерживает безопасные представления. Вы можете ограничить доступ пользователей без привилегии SELECT на базовые таблицы, на которые ссылается представление, от выполнения запросов к представлению.

ПОДСКАЗКА

Только пользователи с привилегией CREATE VIEW для конкретной базы данных могут выполнять эту операцию.

Синтаксис

```
CREATE [OR REPLACE] VIEW [IF NOT EXISTS]
[<database>.]<view_name>
(
    <column_name>[ COMMENT 'column comment']
    [, <column_name>[ COMMENT 'column comment'], ...]
)
[COMMENT 'view comment']
[SECURITY {NONE | INVOKER}]
AS <query_statement>
```

Параметры

Параметр	Описание
OR REPLACE	Заменить существующее представление.
database	Имя базы данных, в которой находится представление.
view_name	Имя представления. Для ознакомления с соглашениями об именовании см. Системные ограничения.
column_name	Имя столбца(ов) в представлении. Обратите внимание, что столбцы в представлении и столбцы, запрашиваемые в `query_statement`, должны совпадать по количеству.
COMMENT	Комментарий к столбцу в представлении или к самому представлению.
SECURITY	Указывает контекст безопасности, который будет использоваться при проверке прав доступа во время вызова представления. Допустимые значения: • `NONE` (По умолчанию): Только пользователи с привилегией SELECT на представление могут запрашивать представление. • `INVOKER`: Только пользователи с привилегией SELECT как на представление, так и на базовые таблицы, на которые оно ссылается, могут запрашивать представление.
query_statement	Выражение запроса, используемое для создания представления. Может быть любым выражением запроса, поддерживаемым Selena.

Примечания по использованию

- Запрос к представлению требует наличия привилегии SELECT на представление и на соответствующие базовые таблицы.
- Если выражение запроса, использованное для создания представления, не может быть выполнено из-за изменения схемы в базовых таблицах, Selena вернет ошибку при запросе к представлению.

Примеры

Пример 1: Создать представление с именем `example_view` в `example_db` с агрегирующим запросом к `example_table`.

```
CREATE VIEW example_db.example_view (k1, k2, k3, v1)
AS
SELECT c1 as k1, k2, k3, SUM(v1) FROM example_table
WHERE k1 = 20160112 GROUP BY k1,k2,k3;
```

Пример 2: Создать представление с именем `example_view` в базе данных `example_db` с агрегирующим запросом к таблице `example_table` и указать комментарии для представления и каждого столбца в нем.

```
CREATE VIEW example_db.example_view
(
    k1 COMMENT 'first key',
    k2 COMMENT 'second key',
    k3 COMMENT 'third key',
    v1 COMMENT 'first value'
)
COMMENT 'my first view'
AS
SELECT c1 as k1, k2, k3, SUM(v1) FROM example_table
WHERE k1 = 20160112 GROUP BY k1,k2,k3;
```

Пример 3: Создать безопасное представление с именем `example_view` в `example_db` с запросом к `example_table`. Представление разрешает запросы только от пользователей с привилегией SELECT на базовую таблицу `example_table`.

```
CREATE VIEW example_db.example_view (k1, k2, k3, v1)
COMMENT 'my secure view'
SECURITY INVOKER
AS
SELECT c1 as k1, k2, k3, SUM(v1) FROM example_table
WHERE k1 = 20160112 GROUP BY k1,k2,k3;
```

Связанные SQL

- SHOW CREATE VIEW
- ALTER VIEW
- DROP VIEW

DROP VIEW

DROP VIEW используется для удаления логического представления VIEW

Синтаксис

```
DROP VIEW [IF EXISTS]
[db_name.]view_name
```

Примеры

1. Если представление существует, удалить представление example_view в example_db.

   ```
   DROP VIEW IF EXISTS example_db.example_view;
   ```

SHOW CREATE VIEW

Возвращает выражение CREATE, которое было использовано для создания указанного представления. Выражение CREATE VIEW помогает понять, как определено представление, и предоставляет справочную информацию для изменения или повторного создания представления. Обратите внимание, что выражение SHOW CREATE VIEW требует наличия привилегии `SELECT` на представление и таблицу, на основе которой создано представление.

Начиная с версии v1.5.2, вы можете использовать SHOW CREATE VIEW для запроса выражения, которое используется для создания **материализованного представления**.

Синтаксис

```
SHOW CREATE VIEW [db_name.]view_name
```

Параметры

Параметр	Обязательный	Описание
db_name	Нет	Имя базы данных. Если этот параметр не указан, по умолчанию возвращается выражение CREATE VIEW для указанного представления в вашей текущей базе данных.
view_name	Да	Имя представления.

Вывод

```
+---------+--------------+----------------------+----------------------+
| View    | Create View  | character_set_client | collation_connection |
+---------+--------------+----------------------+----------------------+
```

Следующая таблица описывает параметры, возвращаемые этим выражением.

Параметр	Описание
View	Имя представления.
Create View	Выражение CREATE VIEW для представления.
character_set_client	Набор символов, который клиент использует для отправки выражений в Selena.
collation_connection	Правила сравнения символов в наборе символов.

Примеры

Создать таблицу с именем `example_table`.

```
CREATE TABLE example_table
(
    k1 TINYINT,
    k2 DECIMAL(10, 2) DEFAULT "10.5",
    v1 CHAR(10) REPLACE,
    v2 INT SUM
)
ENGINE = olap
AGGREGATE KEY(k1, k2)
DISTRIBUTED BY HASH(k1);
```

Создать представление с именем `example_view` на основе `example_table`.

```
CREATE VIEW example_view (k1, k2, k3, v1)
AS SELECT k1, k2, k3, v1 FROM example_table;
```

Отобразить выражение CREATE VIEW для `example_view`.

```
SHOW CREATE VIEW example_db.example_view;

+--------------+-------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-------------------------------------+----------------------+--------------
--------+
| View         | Create View
| character_set_client | collation_connection |
+--------------+-------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-------------------------------------+----------------------+--------------
--------+
| example_view | CREATE VIEW `example_view` (k1, k2, k3, v1) COMMENT "VIEW"
AS SELECT `default_cluster:db1`.`example_table`.`k1` AS `k1`,
`default_cluster:db1`.`example_table`.`k2` AS `k2`,
`default_cluster:db1`.`example_table`.`k3` AS `k3`,
`default_cluster:db1`.`example_table`.`v1` AS `v1` FROM
`default_cluster:db1`.`example_table`; | utf8                 |
utf8_general_ci      |
+--------------+-------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-------------------------------------+----------------------+--------------
--------+
```

3.9 Materialized View

ALTER MATERIALIZED VIEW

ALTER MATERIALIZED VIEW может:

- Изменить имя асинхронного materialized view.
- Изменить стратегию обновления асинхронного materialized view.
- Изменить статус асинхронного materialized view на активный или неактивный.
- Выполнить атомарный обмен между двумя асинхронными materialized views.

- Изменить свойства асинхронного materialized view.

  Вы можете использовать этот SQL-оператор для изменения следующих свойств:

  - `bloom_filter_columns`
  - `partition_ttl_number`
  - `partition_refresh_number`
  - `partition_refresh_strategy`
  - `resource_group`
  - `auto_refresh_partitions_limit`
  - `excluded_trigger_tables`
  - `mv_rewrite_staleness_second`
  - `unique_constraints`
  - `foreign_key_constraints`
  - `colocate_with`
  - `excluded_refresh_tables`
  - Все свойства, связанные с переменными сессии. Для получения информации о переменных сессии см. System variables.

ПОДСКАЗКА

- Эта операция требует привилегию ALTER для целевого materialized view. Вы можете следовать инструкциям в GRANT для предоставления этой привилегии.
- ALTER MATERIALIZED VIEW не поддерживает прямое изменение SQL-запроса, используемого для построения materialized view. Вы можете создать новый materialized view и обменять его с оригинальным, используя ALTER MATERIALIZED VIEW SWAP WITH.

Синтаксис

```
ALTER MATERIALIZED VIEW [db_name.]<mv_name>
    { RENAME [db_name.]<new_mv_name>
    | REFRESH <new_refresh_scheme_desc>
    | ACTIVE | INACTIVE
    | SWAP WITH [db_name.]<mv2_name>
    | SET ( "<key>" = "<value>"[,...]) }
```

Параметры

Parameter	Required	Description
mv_name	yes	Имя materialized view для изменения.
new_refresh_scheme_desc	no	Новая стратегия обновления, см. SQL Reference - CREATE MATERIALIZED VIEW - Parameters для подробностей.
new_mv_name	no	Новое имя для materialized view.
ACTIVE	no	Установить статус materialized view на активный. Selena автоматически устанавливает materialized view в неактивное состояние, если какая-либо из его базовых таблиц изменяется, например, удаляется и пересоздаётся, чтобы предотвратить ситуацию, когда исходные метаданные не соответствуют изменённой базовой таблице. Неактивные materialized views не могут использоваться для ускорения запросов или переписывания запросов. Вы можете использовать этот SQL для активации materialized view после изменения базовых таблиц.
INACTIVE	no	Установить статус materialized view на неактивный. Неактивный асинхронный materialized view не может быть обновлён. Но вы всё равно можете запрашивать его как таблицу.
SWAP WITH	no	Выполнить атомарный обмен с другим асинхронным materialized view после необходимых проверок согласованности.
key	no	Имя свойства для изменения, см. SQL Reference - CREATE MATERIALIZED VIEW - Parameters для подробностей. **ПРИМЕЧАНИЕ** Если вы хотите изменить свойство materialized view, связанное с переменной сессии, вы должны добавить префикс `session.` к свойству, например, `session.insert_timeout`. Вам не нужно указывать префикс для свойств, не связанных с сессией, например, `mv_rewrite_staleness_second`.
value	no	Значение свойства для изменения.

Примеры

Пример 1: Изменить имя materialized view.

```
ALTER MATERIALIZED VIEW lo_mv1 RENAME lo_mv1_new_name;
```

Пример 2: Изменить интервал обновления materialized view.

```
ALTER MATERIALIZED VIEW lo_mv2 REFRESH ASYNC EVERY(INTERVAL 1 DAY);
```

Пример 3: Изменить продолжительность таймаута для задач обновления materialized view на 1 час (по умолчанию).

```
ALTER MATERIALIZED VIEW mv1 SET ("session.insert_timeout" = "3600");
```

Пример 4: Изменить статус materialized view на активный.

```
ALTER MATERIALIZED VIEW order_mv ACTIVE;
```

Пример 5: Выполнить атомарный обмен между materialized views `order_mv` и `order_mv1`.

```
ALTER MATERIALIZED VIEW order_mv SWAP WITH order_mv1;
```

Пример 6: Включить профилирование для процесса обновления materialized view. Эта функция включена по умолчанию.

```
ALTER MATERIALIZED VIEW mv1 SET ("session.enable_profile" = "true");
```

Пример 7: Включить выгрузку промежуточных результатов для процесса обновления materialized view и установить режим выгрузки в `force`. Выгрузка промежуточных результатов включена по умолчанию начиная с v1.5.2.

```
-- Включить выгрузку во время обновления materialized view.
ALTER MATERIALIZED VIEW mv1 SET ("session.enable_spill" = "true");
-- Установить spill_mode в force (значение по умолчанию - auto).
ALTER MATERIALIZED VIEW mv1 SET ("session.spill_mode" = "force");
```

Пример 8: Изменить продолжительность таймаута оптимизатора для materialized view на 30 секунд (по умолчанию начиная с v1.5.2), если его SQL-запрос содержит внешние таблицы или несколько JOIN.

```
ALTER MATERIALIZED VIEW mv1 SET ("session.new_planner_optimize_timeout" =
"30000");
```

Пример 9: Изменить время устаревания для переписывания запросов для materialized view на 600 секунд.

```
ALTER MATERIALIZED VIEW mv1 SET ("mv_rewrite_staleness_second" = "600");
```

Пример 10: Изменить индексы bloom filter materialized view.

```
ALTER MATERIALIZED VIEW mv1 SET ("bloom_filter_columns" = "col1, col2");
```

CANCEL REFRESH MATERIALIZED VIEW

CANCEL REFRESH MATERIALIZED VIEW отменяет задачу обновления асинхронного материализованного представления.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия REFRESH на целевом материализованном представлении.

Синтаксис

```
CANCEL REFRESH MATERIALIZED VIEW [<database_name>.]<materialized_view_name>
[FORCE]
```

Параметры

Параметр	Обязательный	Описание
database_name	Нет	Имя базы данных, в которой находится материализованное представление. Если этот параметр не указан, используется текущая база данных.
materialized_view_name	Да	Имя материализованного представления.
FORCE	НЕТ	Принудительно отменяет выполняющуюся задачу обновления материализованного представления.

Примеры

Пример 1: Отменить задачу обновления для материализованного представления с ASYNC обновлением `lo_mv1`.

```
CANCEL REFRESH MATERIALIZED VIEW lo_mv1;
CANCEL REFRESH MATERIALIZED VIEW lo_mv1 FORCE;
```

CREATE MATERIALIZED VIEW

CREATE MATERIALIZED VIEW создает материализованное представление. Информацию об использовании материализованных представлений см. в разделах Синхронное материализованное представление и Асинхронное материализованное представление.

> **ПРЕДУПРЕЖДЕНИЕ**
>
> - Только пользователи с привилегией CREATE MATERIALIZED VIEW в базе данных, где находится базовая таблица, могут создавать материализованное представление.
> - Начиная с версии v1.5.2, Selena поддерживает создание синхронных материализованных представлений в shared-data кластерах.

Создание материализованного представления — это асинхронная операция. Успешное выполнение этой команды означает, что задача создания материализованного представления успешно отправлена. Вы можете просмотреть статус сборки синхронного материализованного представления в базе данных с помощью команды SHOW ALTER MATERIALIZED VIEW, а асинхронного материализованного представления — путем запроса метаданных представлений `tasks` и `task_runs` в Information Schema.

Selena поддерживает асинхронные материализованные представления начиная с версии v1.5.2. Основные различия между асинхронными и синхронными материализованными представлениями в предыдущих версиях следующие:

	Агрегация по одной таблице	Многотабличный join	Переписывание запросов	Стратегия обновления	Базовая таблица
ASYNC MV	Да	Да	Да	• Асинхронное обновление	Множество таблиц из: • Каталога по умолчанию

	Агрегация по одной таблице	Многотабличный join	Переписывание запросов	Стратегия обновления	Базовая таблица
				• Ручное обновление	• Внешних каталогов (v1.5.2) • Существующих материализованных представлений (v1.5.2) • Существующих представлений (v1.5.2)
SYNC MV (Rollup)	Ограниченный выбор агрегатных функций	Нет	Да	Синхронное обновление при загрузке данных	Одна таблица в каталоге по умолчанию

Синхронное материализованное представление

Синтаксис

```
CREATE MATERIALIZED VIEW [IF NOT EXISTS] [database.]<mv_name>
[COMMENT ""]
[PROPERTIES ("key"="value", ...)]
AS
<query_statement>
```

Параметры в квадратных скобках [] являются необязательными.

Параметры

mv_name (обязательный)

Имя материализованного представления. Требования к именованию следующие:

- Имя должно состоять из букв (a-z или A-Z), цифр (0-9) или символов подчеркивания (_) и может начинаться только с буквы.
- Длина имени не может превышать 64 символа.
- Имя чувствительно к регистру.

COMMENT (необязательный)

Комментарий к материализованному представлению. Обратите внимание, что `COMMENT` должен быть размещен после `mv_name`. В противном случае материализованное

представление не может быть создано.

query_statement (обязательный)

Запрос для создания материализованного представления. Его результат — это данные в материализованном представлении. Синтаксис следующий:

```
SELECT select_expr[, select_expr ...]
[WHERE where_expr]
[GROUP BY column_name[, column_name ...]]
[ORDER BY column_name[, column_name ...]]
```

- select_expr (обязательный)

  Все столбцы в запросе, то есть все столбцы в схеме материализованного представления. Этот параметр поддерживает следующие значения:

  - Простые столбцы или агрегированные столбцы, такие как `SELECT a, abs(b), min(c) FROM table_a`, где `a`, `b` и `c` — имена столбцов в базовой таблице. Если вы не укажете имена столбцов для материализованного представления, Selena автоматически назначит имена столбцам.
  - Выражения, такие как `SELECT a+1 AS x, b+2 AS y, c*c AS z FROM table_a`, где `a+1`, `b+2` и `c*c` — выражения, которые ссылаются на столбцы в базовых таблицах, а `x`, `y` и `z` — псевдонимы, назначенные столбцам в материализованном представлении.

> **ПРИМЕЧАНИЕ**
>
> - Вы должны указать хотя бы один столбец в `select_expr`.
> - При создании синхронного материализованного представления с агрегатной функцией необходимо указать предложение GROUP BY и указать хотя бы один столбец GROUP BY в `select_expr`.
> - Синхронные материализованные представления не поддерживают предложения, такие как JOIN и предложение HAVING в GROUP BY.
> - Начиная с версии v1.5.2, каждое синхронное материализованное представление может поддерживать более одной агрегатной функции для каждого столбца базовой таблицы, например, запросы такие как `select b, sum(a), min(a) from table group by b`.

> - Начиная с версии v1.5.2, синхронные материализованные представления поддерживают сложные выражения для SELECT и агрегатных функций, например, запросы такие как `select b, sum(a + 1) as sum_a1, min(cast (a as bigint)) as min_a from table group by b` или `select abs(b) as col1, a + 1 as col2, cast(a as bigint) as col3 from table`. Следующие ограничения накладываются на сложные выражения, используемые для синхронных материализованных представлений:
> - Каждое сложное выражение должно иметь псевдоним, и разные псевдонимы должны быть назначены разным сложным выражениям среди всех синхронных материализованных представлений базовой таблицы. Например, запросы `select b, sum(a + 1) as sum_a from table group by b` и `select b, sum(a) as sum_a from table group by b` не могут использоваться для создания синхронных материализованных представлений для одной и той же базовой таблицы. Вы можете установить разные псевдонимы для сложного выражения.
> - Вы можете проверить, переписаны ли ваши запросы синхронными материализованными представлениями, созданными со сложными выражениями, выполнив `EXPLAIN <sql_statement>`. Для получения дополнительной информации см. Анализ запросов.

- WHERE (необязательный)

  Начиная с версии v1.5.2, синхронные материализованные представления поддерживают предложение WHERE, которое может фильтровать строки, используемые для материализованного представления.

- GROUP BY (необязательный)

  Столбец GROUP BY запроса. Если этот параметр не указан, данные не будут группироваться по умолчанию.

- ORDER BY (необязательный)

  Столбец ORDER BY запроса.

  - Столбцы в предложении ORDER BY должны быть объявлены в том же порядке, что и столбцы в `select_expr`.

- Если запрос содержит предложение GROUP BY, столбцы ORDER BY должны быть идентичны столбцам GROUP BY.
- Если этот параметр не указан, система автоматически дополнит столбец ORDER BY в соответствии со следующими правилами:
  - Если материализованное представление имеет тип AGGREGATE, все столбцы GROUP BY автоматически используются в качестве ключей сортировки.
  - Если материализованное представление не имеет типа AGGREGATE, Selena автоматически выбирает ключи сортировки на основе префиксных столбцов.

Запрос синхронного материализованного представления

Поскольку синхронное материализованное представление по сути является индексом базовой таблицы, а не физической таблицей, вы можете запросить синхронное материализованное представление только с помощью подсказки `[_SYNC_MV_]`:

```
-- Не опускайте квадратные скобки [] в подсказке.
SELECT * FROM <mv_name> [_SYNC_MV_];
```

> **ПРЕДУПРЕЖДЕНИЕ**
>
> В настоящее время Selena автоматически генерирует имена для столбцов в синхронном материализованном представлении, даже если вы указали для них псевдонимы.

Автоматическое переписывание запросов с синхронным материализованным представлением

Когда выполняется запрос, который соответствует шаблону синхронного материализованного представления, исходный запрос автоматически переписывается, и используются промежуточные результаты, хранящиеся в материализованном представлении.

В следующей таблице показано соответствие между агрегатной функцией в исходном запросе и агрегатной функцией, используемой для построения материализованного представления. Вы можете выбрать соответствующую агрегатную функцию для построения материализованного представления в соответствии с вашим бизнес-сценарием.

агрегатная функция в исходном запросе	агрегатная функция материализованного представления
sum	sum
min	min
max	max
count	count
bitmap_union, bitmap_union_count, count(distinct)	bitmap_union
hll_raw_agg, hll_union_agg, ndv, approx_count_distinct	hll_union
percentile_approx, percentile_union	percentile_union

Помимо вышеуказанных функций, начиная с Selena v1.5.2, синхронные материализованные представления также поддерживают общие агрегатные функции. Для получения дополнительной информации об общих агрегатных функциях см. Состояния общих агрегатных функций.

```
-- Создать синхронное материализованное представление test_mv1 для хранения
агрегатных состояний.
CREATE MATERIALIZED VIEW test_mv1
AS
SELECT
    dt,
    -- Исходные агрегатные функции.
    min(id) AS min_id,
    max(id) AS max_id,
    sum(id) AS sum_id,
    bitmap_union(to_bitmap(id)) AS bitmap_union_id,
    hll_union(hll_hash(id)) AS hll_union_id,
    percentile_union(percentile_hash(id)) AS percentile_union_id,
    -- Функции состояния общих агрегатных функций.
    ds_hll_count_distinct_union(ds_hll_count_distinct_state(id)) AS hll_id,
    avg_union(avg_state(id)) AS avg_id,
    array_agg_union(array_agg_state(id)) AS array_agg_id,
    min_by_union(min_by_state(province, id)) AS min_by_province_id
FROM t1
GROUP BY dt;
```

Асинхронное материализованное представление

Синтаксис

```
CREATE MATERIALIZED VIEW [IF NOT EXISTS] [database.]<mv_name>
[COMMENT ""]
-- Вы должны указать либо `distribution_desc`, либо `refresh_scheme`, либо
оба.
-- distribution_desc
[DISTRIBUTED BY HASH(<bucket_key>[,<bucket_key2> ...]) [BUCKETS
<bucket_number>]]
-- refresh_desc
[REFRESH
-- refresh_moment
    [IMMEDIATE | DEFERRED]
-- refresh_scheme
    [ASYNC | ASYNC [START (<start_time>)] EVERY (INTERVAL
<refresh_interval>) | MANUAL]
]
-- partition_expression
[PARTITION BY
  [ <partition_column> [,...] ] | [ <date_function_expr> ]
]
-- order_by_expression
[ORDER BY (<sort_key>)]
[PROPERTIES ("key"="value", ...)]
AS
<query_statement>
```

Параметры в квадратных скобках [] являются необязательными.

Параметры

mv_name (обязательный)

Имя материализованного представления. Требования к именованию следующие:

- Имя должно состоять из букв (a-z или A-Z), цифр (0-9) или символов подчеркивания (_) и может начинаться только с буквы.
- Длина имени не может превышать 64 символа.
- Имя чувствительно к регистру.

> **ПРЕДУПРЕЖДЕНИЕ**

> Для одной базовой таблицы можно создать несколько материализованных представлений, но имена материализованных представлений в одной базе данных не могут дублироваться.

COMMENT (необязательный)

Комментарий к материализованному представлению. Обратите внимание, что `COMMENT` должен быть размещен после `mv_name`. В противном случае материализованное представление не может быть создано.

distribution_desc (необязательный)

Стратегия bucketing асинхронного материализованного представления. Selena поддерживает hash bucketing и random bucketing (начиная с версии v1.5.2). Если вы не указываете этот параметр, Selena использует стратегию random bucketing и автоматически устанавливает количество buckets.

> **ПРИМЕЧАНИЕ**
>
> При создании асинхронного материализованного представления необходимо указать либо `distribution_desc`, либо `refresh_scheme`, либо оба.

- **Hash bucketing**:

  Синтаксис

  ```
  DISTRIBUTED BY HASH (<bucket_key1>[,<bucket_key2> ...]) [BUCKETS
  <bucket_number>]
  ```

  Для получения дополнительной информации см. Распределение данных.

  > **ПРИМЕЧАНИЕ**
  >
  > Начиная с версии v1.5.2, Selena может автоматически устанавливать количество buckets (BUCKETS) при создании таблицы или добавлении раздела. Вам больше не нужно вручную устанавливать количество buckets. Подробную информацию см. в разделе установка количества buckets.

- **Random bucketing**:

Если вы выбираете стратегию random bucketing и разрешаете Selena автоматически устанавливать количество buckets, вам не нужно указывать `distribution_desc`. Однако, если вы хотите вручную установить количество buckets, вы можете воспользоваться следующим синтаксисом:

```
DISTRIBUTED BY RANDOM BUCKETS <bucket_number>
```

> **ПРЕДУПРЕЖДЕНИЕ**
>
> Асинхронные материализованные представления со стратегией random bucketing не могут быть назначены группе colocation.

Для получения дополнительной информации см. Random bucketing

refresh_moment (необязательный)

Момент обновления материализованного представления. Значение по умолчанию: `IMMEDIATE`. Допустимые значения:

- `IMMEDIATE`: Обновить асинхронное материализованное представление немедленно после его создания.
- `DEFERRED`: Асинхронное материализованное представление не обновляется после создания. Вы можете вручную обновить материализованное представление или запланировать регулярные задачи обновления.

refresh_scheme (необязательный)

> **ПРИМЕЧАНИЕ**
>
> - При создании асинхронного материализованного представления необходимо указать либо `distribution_desc`, либо `refresh_scheme`, либо оба.
> - Материализованные представления внешних таблиц не поддерживают автоматическое обновление **по триггеру изменения данных базовой таблицы**. Они поддерживают только асинхронное обновление с **фиксированным интервалом** и ручное обновление.

Стратегия обновления асинхронного материализованного представления. Допустимые значения:

- `ASYNC`: Режим автоматического обновления. Каждый раз, когда изменяются данные базовой таблицы, материализованное представление автоматически обновляется.
- `ASYNC [START (<start_time>)] EVERY(INTERVAL <interval>)`: Режим регулярного обновления. Материализованное представление обновляется регулярно с указанным интервалом. Вы можете указать интервал как `EVERY (interval n day/hour/minute/second)` с использованием следующих единиц: `DAY`, `HOUR`, `MINUTE` и `SECOND`. Значение по умолчанию — `10 MINUTE`. Вы можете дополнительно указать время начала обновления как `START('yyyy-MM-dd hh:mm:ss')`. Если время начала не указано, используется текущее время. Пример: `ASYNC START ('2023-09-12 16:30:25') EVERY (INTERVAL 5 MINUTE)`.
- `MANUAL`: Режим ручного обновления. Материализованное представление не будет обновляться, если вы не запустите задачу обновления вручную.

Если этот параметр не указан, используется значение по умолчанию `MANUAL`.

partition_expression (необязательный)

Стратегия разделения асинхронного материализованного представления. Если этот параметр не указан, стратегия разделения по умолчанию не применяется.

Допустимые значения:

- `partition_column`: Столбец(ы), используемые для разделения. Выражение `PARTITION BY dt` означает разделение материализованного представления в соответствии со столбцом `dt`.
- `date_function_expr`: Сложное выражение с функциями дат, используемое для разделения.
  - Функция `date_trunc`: Функция, используемая для усечения единицы времени. `PARTITION BY date_trunc("MONTH", dt)` означает, что столбец `dt` усекается до месяца в качестве единицы для разделения. Функция `date_trunc` поддерживает усечение времени до единиц, включая `YEAR`, `MONTH`, `DAY`, `HOUR` и `MINUTE`.
  - Функция `str2date`: Функция, используемая для преобразования разделов типа string базовой таблицы в типы дат. `PARTITION BY str2date(dt, "%Y%m%d")` означает, что столбец `dt` является типом даты STRING, формат даты которого `"%Y%m%d"`. Функция `str2date` поддерживает множество форматов дат, вы можете обратиться к str2date для получения дополнительной информации. Поддерживается с версии v1.5.2.

- Функция `time_slice`: Начиная с версии v1.5.2, вы можете дополнительно использовать эти функции для преобразования указанного времени в начало или конец временного интервала на основе указанной временной детализации, например, `PARTITION BY date_trunc("MONTH", time_slice(dt, INTERVAL 7 DAY))`, где time_slice должен иметь более тонкую детализацию, чем date_trunc. Вы можете использовать их для указания столбца GROUP BY с более тонкой детализацией, чем у ключа разделения, например, `GROUP BY time_slice(dt, INTERVAL 1 MINUTE) PARTITION BY date_trunc('DAY', ts)`.

Начиная с версии v1.5.2, асинхронные материализованные представления поддерживают многостолбцовые выражения разделов. Вы можете указать несколько столбцов разделов для материализованного представления для сопоставления всех или части столбцов разделов базовых таблиц.

Примечания для многостолбцовых выражений разделов:

- В настоящее время многостолбцовые разделы в материализованных представлениях могут быть сопоставлены только напрямую со столбцами разделов базовой таблицы. Сопоставление с использованием функций или выражений на столбцах разделов базовой таблицы не поддерживается.

- Поскольку выражения разделов Iceberg поддерживают функцию `transform`, требуется дополнительная обработка при сопоставлении выражений разделов Iceberg с выражениями разделов материализованных представлений Selena. Соотношение сопоставления следующее:

Iceberg Transform	Выражение раздела Iceberg	Выражение раздела материализованного представления
Identity	`<col>`	`<col>`
hour	`hour(<col>)`	`date_trunc('hour', <col>)`
day	`day(<col>)`	`date_trunc('day', <col>)`
month	`month(<col>)`	`date_trunc('month', <col>)`
year	`year(<col>)`	`date_trunc('year', <col>)`
bucket	`bucket(<col>, <n>)`	Не поддерживается
truncate	`truncate(<col>)`	Не поддерживается

- Для столбцов разделов, не относящихся к Iceberg, где вычисление выражений разделов не требуется, дополнительная обработка выражений разделов не

требуется. Вы можете сопоставить их напрямую.

См. Пример -5 для подробных инструкций по многостолбцовым выражениям разделов.

> **ПРЕДУПРЕЖДЕНИЕ**
>
> Начиная с версии v1.5.2, Selena поддерживает создание асинхронных материализованных представлений со стратегией разделения List Partitioning.
>
> - Вы можете создавать материализованные представления с разделением списка на основе таблиц, созданных со стратегией разделения List Partitioning или Expression partitioning.
> - В настоящее время при создании материализованных представлений со стратегией разделения List Partitioning можно указать только один ключ раздела. Вы должны выбрать один ключ раздела, если базовая таблица имеет более одного ключа раздела.
> - Поведение обновления и логика переписывания запросов материализованных представлений со стратегией разделения List Partitioning согласуются с теми, которые имеют стратегию разделения Range Partitioning.

order_by_expression (необязательный)

Ключ сортировки асинхронного материализованного представления. Если вы не указываете ключ сортировки, Selena выбирает некоторые из префиксных столбцов из столбцов SELECT в качестве ключей сортировки. Например, в `select a, b, c, d` ключами сортировки могут быть `a` и `b`. Этот параметр поддерживается начиная с Selena v1.5.2.

> **ПРИМЕЧАНИЕ** Существует два разных использования `ORDER BY` в материализованных представлениях:
>
> - `ORDER BY` в операторе CREATE MATERIALIZED VIEW определяет ключ сортировки материализованного представления, что помогает ускорить запросы на основе ключа сортировки. Это не влияет на способность материализованного представления к прозрачному ускорению на основе SPJG, но не гарантирует глобальную сортировку результатов запроса материализованного представления.
> - `ORDER BY` в определении запроса материализованного представления гарантирует глобальную сортировку результатов запроса, но предотвращает

> использование материализованного представления для прозрачного переписывания запросов на основе SPJG. Поэтому `ORDER BY` не следует использовать в определении запроса материализованного представления, если MV используется для переписывания запросов.

INDEX (необязательный)

Асинхронные материализованные представления поддерживают индексы Bitmap и BloomFilter для ускорения производительности запросов, и их использование такое же, как и в обычных таблицах. Подробности об использовании и информацию об индексах Bitmap и BloomFilter см. в разделах Индекс Bitmap и Индекс Bloom filter.

Использование индексов Bitmap:

```
-- Создать индекс
CREATE INDEX <index_name> ON <mv_name>(<column_name>) USING BITMAP COMMENT
'<comment>';

-- Проверить прогресс создания индекса
SHOW ALTER TABLE COLUMN;

-- Просмотреть индексы
SHOW INDEXES FROM <mv_name>;

-- Удалить индекс
DROP INDEX <index_name> ON <mv_name>;
```

Использование индексов BloomFilter:

```
-- Создать индекс
ALTER MATERIALIZED VIEW <mv_name> SET ("bloom_filter_columns" = "
<col1,col2,col3,...>");

-- Просмотреть индексы
SHOW CREATE MATERIALIZED VIEW <mv_name>;

-- Удалить индекс
ALTER MATERIALIZED VIEW <mv_name> SET ("bloom_filter_columns" = "");
```

PROPERTIES (необязательный)

Свойства асинхронного материализованного представления. Вы можете изменить свойства существующего материализованного представления с помощью ALTER MATERIALIZED VIEW.

- `session.`: Если вы хотите изменить свойство, связанное с переменной сеанса, материализованного представления, вы должны добавить префикс `session.` к свойству, например, `session.insert_timeout`. Вам не нужно указывать префикс для свойств, не связанных с сеансом, например, `mv_rewrite_staleness_second`.

- `replication_num`: Количество реплик материализованного представления для создания.

- `storage_medium`: Тип носителя хранения. Допустимые значения: `HDD` и `SSD`.

- `storage_cooldown_time`: Время охлаждения хранилища для раздела. Если используются носители хранения HDD и SSD, данные в хранилище SSD перемещаются в хранилище HDD после времени, указанного этим свойством. Формат: "yyyy-MM-dd HH:mm:ss". Указанное время должно быть позже текущего времени. Если это свойство не указано явно, охлаждение хранилища не выполняется по умолчанию.

- `bloom_filter_columns`: Массив имен столбцов, которые включают индексацию Bloom filter. Подробности об индексах Bloom filter см. в разделе Индекс Bloom filter.

- `partition_ttl`: Время жизни (TTL) для разделов. Сохраняются разделы, данные которых находятся в пределах указанного временного диапазона. Просроченные разделы удаляются автоматически. Единица: `YEAR`, `MONTH`, `DAY`, `HOUR` и `MINUTE`. Например, вы можете указать это свойство как `2 MONTH`. Это свойство рекомендуется вместо `partition_ttl_number`. Оно поддерживается с версии v1.5.2.

- `partition_ttl_number`: Количество самых последних разделов материализованного представления для сохранения. Для разделов со временем начала, ранним, чем текущее время, после того, как количество этих разделов превысит это значение, менее недавние разделы будут удалены. Selena периодически проверяет разделы материализованного представления в соответствии с временным интервалом, указанным в элементе конфигурации FE `dynamic_partition_check_interval_seconds`, и автоматически удаляет просроченные разделы. Если вы включили стратегию динамического разделения,

разделы, созданные заранее, не учитываются. Когда значение равно `-1`, все разделы материализованного представления будут сохранены. По умолчанию: `-1`.

- `partition_refresh_number`: В одном обновлении максимальное количество разделов для обновления. Если количество разделов для обновления превышает это значение, Selena разделит задачу обновления и завершит ее партиями. Только когда предыдущая партия разделов успешно обновлена, Selena продолжит обновлять следующую партию разделов, пока все разделы не будут обновлены. Если какой-либо из разделов не удастся обновить, последующие задачи обновления не будут созданы. Когда значение равно `-1`, задача обновления не будет разделена. Значение по умолчанию изменено с `-1` на `1` с версии v1.5.2, что означает, что Selena обновляет разделы один за другим.

- `partition_refresh_strategy` : Стратегия обновления для материализованного представления во время одной операции обновления. Когда установлено значение `adaptive`, количество разделов для обновления будет автоматически определяться на основе объема данных в разделах базовой таблицы, что значительно повышает эффективность обновления. Если это свойство не указано, стратегия по умолчанию — `strict`, что означает, что количество разделов, обновленных за одну операцию, строго контролируется `partition_refresh_number`.

- `excluded_trigger_tables`: Если базовая таблица материализованного представления указана здесь, задача автоматического обновления не будет запущена при изменении данных в базовой таблице. Этот параметр применяется только к стратегии обновления, вызванной загрузкой, и обычно используется вместе со свойством `auto_refresh_partitions_limit`. Формат: `[db_name.]table_name`. Когда значение — пустая строка, любое изменение данных во всех базовых таблицах вызывает обновление соответствующего материализованного представления. Значение по умолчанию — пустая строка.

- `excluded_refresh_tables`: Базовые таблицы, перечисленные в этом свойстве, не будут обновлены в материализованное представление при изменении их данных. Формат: `[db_name.]table_name`. Значение по умолчанию — пустая строка. Когда значение — пустая строка, любое изменение данных базовой таблицы вызовет обновление соответствующего материализованного представления.

ПОДСКАЗКА

Разница между `excluded_trigger_tables` и `excluded_refresh_tables`:

- `excluded_trigger_tables` контролирует, запускать ли обновление, а не участвовать ли в обновлении. Например, разделенное материализованное представление получается путем соединения двух разделенных таблиц A и B, и разделы двух таблиц A и B соответствуют один к одному. `excluded_trigger_table` содержит таблицу A. В течение периода времени таблица A обновила разделы `[1,2,3]`, но поскольку это `excluded_trigger_table`, обновление материализованного представления не запускается. В это время таблица B обновляет раздел `[3]`, и материализованное представление запускает обновление, которое обновит три раздела `[1, 2, 3]`. Здесь вы можете видеть, что `excluded_trigger_table` только контролирует, запускать ли обновление. Хотя обновление таблицы A не может запустить обновление материализованного представления, когда обновление таблицы B запускает обновление материализованного представления, раздел, обновленный таблицей A, также будет добавлен в задачу обновления.
- `excluded_refresh_tables` контролирует, участвовать ли в обновлении. В приведенном выше примере, если таблица A существует как в `excluded_trigger_table`, так и в `excluded_refresh_tables`, когда обновление таблицы B запускает обновление материализованного представления, будет обновлен только раздел `[3]`.

- `auto_refresh_partitions_limit`: Количество самых последних разделов материализованного представления, которые должны быть обновлены при запуске обновления материализованного представления. Вы можете использовать это свойство для ограничения диапазона обновления и снижения затрат на обновление. Однако, поскольку не все разделы обновляются, данные в материализованном представлении могут не соответствовать базовой таблице. По умолчанию: `-1`. Когда значение равно `-1`, все разделы будут обновлены. Когда значение является положительным целым числом N, Selena сортирует существующие разделы в хронологическом порядке и обновляет текущий раздел и N-1 самых последних разделов. Если количество разделов меньше N, Selena обновляет все существующие разделы. Если в вашем материализованном представлении есть динамические

разделы, созданные заранее, Selena обновляет все предварительно созданные разделы.

- `mv_rewrite_staleness_second`: Если последнее обновление материализованного представления находится в пределах временного интервала, указанного в этом свойстве, это материализованное представление может использоваться непосредственно для переписывания запросов, независимо от того, изменяются ли данные в базовых таблицах. Если последнее обновление было до этого временного интервала, Selena проверяет, были ли обновлены базовые таблицы, чтобы определить, может ли материализованное представление использоваться для переписывания запросов. Единица: Секунда. Это свойство поддерживается с версии v1.5.2.

- `colocate_with`: Группа colocation асинхронного материализованного представления. См. Colocate Join для получения дополнительной информации. Это свойство поддерживается с версии v1.5.2.

- `unique_constraints` и `foreign_key_constraints`: Ограничения Unique Key и ограничения Foreign Key при создании асинхронного материализованного представления для переписывания запросов в сценарии View Delta Join. См. Асинхронное материализованное представление - Переписывание запросов в сценарии View Delta Join для получения дополнительной информации. Это свойство поддерживается с версии v1.5.2.

- `excluded_refresh_tables` : Базовые таблицы, перечисленные в этом свойстве, не запускают обновление данных в материализованное представление при изменении их данных. Это свойство обычно используется вместе со свойством `excluded_trigger_tables`. Формат: `[db_name.]table_name`. Значение по умолчанию — пустая строка. Когда значение — пустая строка, любое изменение данных во всех базовых таблицах вызовет обновление соответствующего материализованного представления.

> **ПРЕДУПРЕЖДЕНИЕ**
>
> Ограничения Unique Key и Foreign Key используются только для переписывания запросов. Проверки ограничений Foreign Key не гарантируются при загрузке данных в таблицу. Вы должны убедиться, что данные, загружаемые в таблицу, соответствуют ограничениям.

- `resource_group`: Группа ресурсов, к которой принадлежат задачи обновления материализованного представления. Значение по умолчанию этого свойства — `default_mv_wg`, которая является системной группой ресурсов, специально используемой для обновления материализованных представлений. `cpu_core_limit` группы `default_mv_wg` равен `1`, `mem_limit` равен `0.8`. Для получения дополнительной информации о группах ресурсов см. Группа ресурсов.

- `query_rewrite_consistency`: Правило переписывания запросов для асинхронных материализованных представлений. Это свойство поддерживается с версии v1.5.2. Допустимые значения:

  - `disable`: Отключить автоматическое переписывание запросов асинхронного материализованного представления.

  - `checked` (Значение по умолчанию): Включить автоматическое переписывание запросов только тогда, когда материализованное представление соответствует требованию своевременности, что означает:

    - Если `mv_rewrite_staleness_second` не указан, материализованное представление может использоваться для переписывания запросов только тогда, когда его данные согласованы с данными во всех базовых таблицах.
    - Если `mv_rewrite_staleness_second` указан, материализованное представление может использоваться для переписывания запросов, когда его последнее обновление находится в пределах временного интервала устаревания.
  - `loose`: Включить автоматическое переписывание запросов напрямую, и проверка согласованности не требуется.

  - `force_mv`: Начиная с версии v1.5.2, материализованные представления Selena поддерживают TTL выражения общего раздела. Семантика `force_mv` специально разработана для этого сценария. Когда эта семантика включена:

    - Если материализованное представление не имеет свойства `partition_retention_condition`, оно всегда будет принудительно использовать материализованное представление для переписывания запросов, независимо от того, была ли обновлена базовая таблица.
    - Если материализованное представление имеет свойство `partition_retention_condition`:

    - Для разделов в пределах диапазона TTL переписывание запросов на основе материализованного представления всегда доступно, независимо от того, была ли обновлена базовая таблица.
    - Для разделов вне диапазона TTL требуется компенсация Union между материализованным представлением и базовой таблицей, независимо от того, была ли обновлена базовая таблица.

  Например, если материализованное представление имеет определенное свойство `partition_retention_condition`, и раздел для `20241131` истек, но данные базовой таблицы для `20241203` были обновлены, в то время как данные материализованного представления для `20241203` не были обновлены, применяется следующее, когда свойство `query_rewrite_consistency` установлено в `force_mv`:

    - Материализованное представление гарантирует, что запросы к разделам в пределах диапазона TTL (например, с `20241201` по `20241203`), определенного в `partition_retention_condition`, всегда могут быть прозрачно переписаны.
    - Для запросов к разделам вне диапазона `partition_retention_condition` компенсация будет происходить автоматически на основе Union материализованного представления и базовой таблицы.

  См. Пример 6 для подробных инструкций по семантике `force_mv` и `partition_retention_condition`.

- `storage_volume`: Имя тома хранения, используемого для хранения асинхронного материализованного представления, которое вы хотите создать, если вы используете shared-data кластер. Это свойство поддерживается с версии v1.5.2. Если это свойство не указано, используется том хранения по умолчанию. Пример: `"storage_volume" = "def_volume"`.

- `force_external_table_query_rewrite`: Включить ли переписывание запросов для материализованных представлений на основе внешнего каталога. Это свойство поддерживается с версии v1.5.2. Допустимые значения:

  - `true`(Значение по умолчанию с версии v1.5.2): Включить переписывание запросов для материализованных представлений на основе внешнего каталога.
  - `false`: Отключить переписывание запросов для материализованных представлений на основе внешнего каталога.

Поскольку строгая согласованность данных не гарантируется между базовыми таблицами и материализованными представлениями на основе внешнего каталога, эта функция по умолчанию установлена в `false`. Когда эта функция включена, материализованное представление используется для переписывания запросов в соответствии с правилом, указанным в `query_rewrite_consistency`.

- `enable_query_rewrite`: Использовать ли материализованное представление для переписывания запросов. Когда существует много материализованных представлений, переписывание запросов на основе материализованных представлений может повлиять на время, затрачиваемое оптимизатором. С помощью этого свойства вы можете контролировать, может ли материализованное представление использоваться для переписывания запросов. Эта функция поддерживается с версии v1.5.2. Допустимые значения:
    - `default` (По умолчанию): Система не будет выполнять семантические проверки материализованного представления, но только материализованные представления типа SPJG могут использоваться для переписывания запросов. Обратите внимание, что если включено переписывание запросов на основе текста, материализованные представления не типа SPJG также могут использоваться для переписывания запросов.
    - `true`: Система будет выполнять семантические проверки при создании или изменении материализованного представления. Если материализованное представление не подходит для переписывания запросов (то есть определение материализованного представления не является запросом типа SPJG), будет возвращен сбой.
    - `false`: Материализованное представление не будет использоваться для переписывания запросов.
- [Preview] `transparent_mv_rewrite_mode`: Указывает режим прозрачного переписывания для **запросов непосредственно к материализованному представлению**. Эта функция поддерживается с версии v1.5.2. Допустимые значения:
    - `false` (По умолчанию, совместимо с поведением в более ранних версиях): Запросы непосредственно к материализованному представлению не будут переписаны и возвращаются только с существующими данными в материализованном представлении. Их результаты запроса могут отличаться от результатов запросов, основанных на определении материализованного

представления, в соответствии со статусом обновления (согласованность данных) материализованного представления.
  - `true`: Запросы непосредственно к материализованному представлению будут переписаны и возвращены с самыми обновленными данными, которые согласуются с результатом запроса определения материализованного представления. Обратите внимание, что когда материализованное представление неактивно или не поддерживает прозрачное переписывание запросов, эти запросы будут выполняться как запрос определения материализованного представления.
  - `transparent_or_error`: Запросы непосредственно к материализованному представлению будут переписаны, когда они подходят. Если материализованное представление неактивно или не поддерживает прозрачное переписывание запросов, эти запросы будут возвращены с ошибкой.
  - `transparent_or_default` Запросы непосредственно к материализованному представлению будут переписаны, когда они подходят. Если материализованное представление неактивно или не поддерживает прозрачное переписывание запросов, эти запросы будут возвращены с существующими данными в материализованном представлении.
- `partition_retention_condition`: Начиная с версии v1.5.2, материализованные представления Selena поддерживают TTL выражения общего раздела. Это свойство — выражение, которое объявляет разделы, которые должны быть сохранены динамически. Разделы, которые не соответствуют условию в выражении, будут регулярно удаляться. Пример: `'partition_retention_condition' = 'dt >= CURRENT_DATE() - INTERVAL 3 MONTH'`.

  - Выражение может содержать только столбцы разделов и константы. Столбцы, не являющиеся разделами, не поддерживаются.
  - Выражение общего раздела применяется к разделам List и Range по-разному:
    - Для материализованных представлений с разделами List, Selena поддерживает удаление разделов, отфильтрованных выражением общего раздела.
    - Для материализованных представлений с разделами Range, Selena может фильтровать и удалять разделы только с использованием возможности обрезки разделов FE. Разделы, соответствующие предикатам, которые не поддерживаются обрезкой разделов, не могут быть отфильтрованы и удалены.

См. Пример 6 для подробных инструкций по семантике `force_mv` и `partition_retention_condition`.

query_statement (обязательный)

Запрос для создания асинхронного материализованного представления. Начиная с версии v1.5.2, Selena поддерживает создание асинхронных материализованных представлений с Common Table Expression (CTE).

Запрос асинхронного материализованного представления

Асинхронное материализованное представление — это физическая таблица. Вы можете работать с ним как с любой обычной таблицей, **за исключением того, что вы не можете напрямую загружать данные в асинхронное материализованное представление**.

Автоматическое переписывание запросов с асинхронным материализованным представлением

Selena v1.5.2 поддерживает автоматическое и прозрачное переписывание запросов на основе асинхронных материализованных представлений типа SPJG. Материализованные представления типа SPJG относятся к материализованным представлениям, план которых включает только операторы типов Scan, Filter, Project и Aggregate. Переписывание запросов материализованных представлений типа SPJG включает в себя переписывание запросов к одной таблице, переписывание запросов Join, переписывание запросов агрегации, переписывание запросов Union и переписывание запросов на основе вложенных материализованных представлений.

См. Асинхронное материализованное представление - Переписывание запросов с асинхронным материализованным представлением для получения дополнительной информации.

Поддерживаемые типы данных

- Асинхронные материализованные представления, созданные на основе каталога Selena по умолчанию, поддерживают следующие типы данных:
    - **Date**: DATE, DATETIME
    - **String**: CHAR, VARCHAR
    - **Numeric**: BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, DECIMAL, PERCENTILE

- **Semi-structured**: ARRAY, JSON, MAP (с версии v1.5.2), STRUCT (с версии v1.5.2)
- **Other**: BITMAP, HLL

> **ПРИМЕЧАНИЕ**
>
> BITMAP, HLL и PERCENTILE поддерживаются с версии v1.5.2.

- Асинхронные материализованные представления, созданные на основе внешних каталогов Selena, поддерживают следующие типы данных:
  - Hive Catalog
    - **Numeric**: INT/INTEGER, BIGINT, DOUBLE, FLOAT, DECIMAL
    - **Date**: TIMESTAMP
    - **String**: STRING, VARCHAR, CHAR
    - **Semi-structured**: ARRAY
  - Hudi Catalog
    - **Numeric**: BOOLEAN, INT, LONG, FLOAT, DOUBLE, DECIMAL
    - **Date**: DATE, TimeMillis/TimeMicros, TimestampMillis/TimestampMicros
    - **String**: STRING
    - **Semi-structured**: ARRAY
  - Iceberg Catalog
    - **Numeric**: BOOLEAN, INT, LONG, FLOAT, DOUBLE, DECIMAL(P, S)
    - **Date**: DATE, TIME, TIMESTAMP
    - **String**: STRING, UUID, FIXED(L), BINARY
    - **Semi-structured**: LIST

Примечания по использованию

- Текущая версия Selena не поддерживает одновременное создание нескольких материализованных представлений. Новое материализованное представление может быть создано только после завершения предыдущего.

- О синхронных материализованных представлениях:
  - Синхронные материализованные представления поддерживают только агрегатные функции на одном столбце. Запросы в форме `sum(a+b)` не поддерживаются.

- Синхронные материализованные представления поддерживают только одну агрегатную функцию для каждого столбца базовой таблицы. Запросы, такие как `select sum(a), min(a) from table`, не поддерживаются.
- При создании синхронного материализованного представления с агрегатной функцией необходимо указать предложение GROUP BY и указать хотя бы один столбец GROUP BY в SELECT.
- Синхронные материализованные представления не поддерживают предложения, такие как JOIN, и предложение HAVING в GROUP BY.
- При использовании ALTER TABLE DROP COLUMN для удаления определенного столбца в базовой таблице необходимо убедиться, что все синхронные материализованные представления базовой таблицы не содержат удаляемого столбца, в противном случае операция удаления не удастся. Прежде чем удалить столбец, вы должны сначала удалить все синхронные материализованные представления, которые содержат этот столбец.
- Создание слишком большого количества синхронных материализованных представлений для таблицы повлияет на эффективность загрузки данных. При загрузке данных в базовую таблицу данные в синхронном материализованном представлении и базовой таблице будут обновляться синхронно. Если базовая таблица содержит `n` синхронных материализованных представлений, эффективность загрузки данных в базовую таблицу примерно такая же, как эффективность загрузки данных в `n` таблиц.

- О вложенных асинхронных материализованных представлениях:

  - Стратегия обновления для каждого материализованного представления применяется только к соответствующему материализованному представлению.
  - В настоящее время Selena не ограничивает количество уровней вложенности. В производственной среде мы рекомендуем, чтобы количество уровней вложенности не превышало TPEX.
- О асинхронных материализованных представлениях внешнего каталога:

  - Материализованное представление внешнего каталога поддерживает только асинхронное обновление с фиксированным интервалом и ручное обновление.
  - Строгая согласованность не гарантируется между материализованным представлением и базовыми таблицами во внешнем каталоге.
  - В настоящее время создание материализованных представлений на основе внешних ресурсов не поддерживается.

- В настоящее время Selena не может определить, изменились ли данные базовой таблицы во внешнем каталоге, поэтому все разделы будут обновлены по умолчанию при каждом обновлении базовой таблицы. Вы можете вручную обновить только некоторые разделы с помощью REFRESH MATERIALIZED VIEW.

Примеры

Примеры синхронных материализованных представлений

Схема базовой таблицы следующая:

```
mysql> desc duplicate_table;
+-------+--------+------+------+---------+-------+
| Field | Type   | Null | Key  | Default | Extra |
+-------+--------+------+------+---------+-------+
| k1    | INT    | Yes  | true | N/A     |       |
| k2    | INT    | Yes  | true | N/A     |       |
| k3    | BIGINT | Yes  | true | N/A     |       |
| k4    | BIGINT | Yes  | true | N/A     |       |
+-------+--------+------+------+---------+-------+
```

Пример 1: Создать синхронное материализованное представление, которое содержит только столбцы исходной таблицы (k1, k2).

```
create materialized view k1_k2 as
select k1, k2 from duplicate_table;
```

Материализованное представление содержит только два столбца k1 и k2 без какой-либо агрегации.

```
+-----------------+-------+--------+------+------+---------+-------+
| IndexName       | Field | Type   | Null | Key  | Default | Extra |
+-----------------+-------+--------+------+------+---------+-------+
| k1_k2           | k1    | INT    | Yes  | true | N/A     |       |
|                 | k2    | INT    | Yes  | true | N/A     |       |
+-----------------+-------+--------+------+------+---------+-------+
```

Пример 2: Создать синхронное материализованное представление, отсортированное по k2.

```
create materialized view k2_order as
select k2, k1 from duplicate_table order by k2;
```

Схема материализованного представления показана ниже. Материализованное представление содержит только два столбца k2 и k1, где столбец k2 является столбцом сортировки без какой-либо агрегации.

```
+-----------------+-------+--------+------+-------+---------+-------+
| IndexName       | Field | Type   | Null | Key   | Default | Extra |
+-----------------+-------+--------+------+-------+---------+-------+
| k2_order        | k2    | INT    | Yes  | true  | N/A     |       |
|                 | k1    | INT    | Yes  | false | N/A     | NONE  |
+-----------------+-------+--------+------+-------+---------+-------+
```

Пример 3: Создать синхронное материализованное представление, сгруппированное по k1 и k2, и с агрегацией SUM на k3.

```
create materialized view k1_k2_sumk3 as
select k1, k2, sum(k3) from duplicate_table group by k1, k2;
```

Схема материализованного представления показана ниже. Материализованное представление содержит три столбца k1, k2 и sum (k3), где k1, k2 — сгруппированные столбцы, а sum (k3) — сумма столбцов k3, сгруппированных в соответствии с k1 и k2.

```
+-----------------+-------+--------+------+-------+---------+-------+
| IndexName       | Field | Type   | Null | Key   | Default | Extra |
+-----------------+-------+--------+------+-------+---------+-------+
| k1_k2_sumk3     | k1    | INT    | Yes  | true  | N/A     |       |
|                 | k2    | INT    | Yes  | true  | N/A     |       |
|                 | k3    | BIGINT | Yes  | false | N/A     | SUM   |
+-----------------+-------+--------+------+-------+---------+-------+
```

Поскольку материализованное представление не объявляет столбец сортировки и использует агрегатную функцию, Selena дополняет сгруппированные столбцы k1 и k2 по умолчанию.

Пример 4: Создать синхронное материализованное представление для удаления дублирующихся строк.

```
create materialized view deduplicate as
select k1, k2, k3, k4 from duplicate_table group by k1, k2, k3, k4;
```

Схема материализованного представления показана ниже. Материализованное представление содержит столбцы k1, k2, k3 и k4, и нет дублирующихся строк.

```
+-----------------+-------+--------+------+-------+---------+-------+
| IndexName       | Field | Type   | Null | Key   | Default | Extra |
+-----------------+-------+--------+------+-------+---------+-------+
| deduplicate     | k1    | INT    | Yes  | true  | N/A     |       |
|                 | k2    | INT    | Yes  | true  | N/A     |       |
|                 | k3    | BIGINT | Yes  | true  | N/A     |       |
|                 | k4    | BIGINT | Yes  | true  | N/A     |       |
+-----------------+-------+--------+------+-------+---------+-------+
```

Пример 5: Создать неагрегированное синхронное материализованное представление, которое не объявляет столбец сортировки.

Схема базовой таблицы показана ниже:

```
+-------+--------------+------+-------+---------+-------+
| Field | Type         | Null | Key   | Default | Extra |
+-------+--------------+------+-------+---------+-------+
| k1    | TINYINT      | Yes  | true  | N/A     |       |
| k2    | SMALLINT     | Yes  | true  | N/A     |       |
| k3    | INT          | Yes  | true  | N/A     |       |
| k4    | BIGINT       | Yes  | true  | N/A     |       |
| k5    | DECIMAL(9,0) | Yes  | true  | N/A     |       |
| k6    | DOUBLE       | Yes  | false | N/A     | NONE  |
| k7    | VARCHAR(20)  | Yes  | false | N/A     | NONE  |
+-------+--------------+------+-------+---------+-------+
```

Материализованное представление содержит столбцы k3, k4, k5, k6 и k7, и столбец сортировки не объявлен. Создайте материализованное представление со следующим оператором:

```
create materialized view mv_1 as
select k3, k4, k5, k6, k7 from all_type_table;
```

Selena автоматически использует k3, k4 и k5 в качестве столбцов сортировки по умолчанию. Сумма байтов, занимаемых этими тремя типами столбцов, составляет 4 (INT) + 8 (BIGINT) + 16 (DECIMAL) = 28 < 36. Поэтому эти три столбца добавляются в качестве столбцов сортировки.

Схема материализованного представления следующая.

```
+----------------+-------+--------------+------+-------+---------+-------+
| IndexName      | Field | Type         | Null | Key   | Default | Extra |
+----------------+-------+--------------+------+-------+---------+-------+
| mv_1           | k3    | INT          | Yes  | true  | N/A     |       |
|                | k4    | BIGINT       | Yes  | true  | N/A     |       |
|                | k5    | DECIMAL(9,0) | Yes  | true  | N/A     |       |
|                | k6    | DOUBLE       | Yes  | false | N/A     | NONE  |
|                | k7    | VARCHAR(20)  | Yes  | false | N/A     | NONE  |
+----------------+-------+--------------+------+-------+---------+-------+
```

Можно заметить, что поле `key` столбцов k3, k4 и k5 — `true`, что указывает на то, что они являются ключами сортировки. Поле key столбцов k6 и k7 — `false`, что указывает на то, что они не являются ключами сортировки.

Пример 6: Создать синхронное материализованное представление, которое содержит предложение WHERE и сложные выражения.

```sql
-- Создать базовую таблицу: user_event
CREATE TABLE user_event (
      ds date   NOT NULL,
      id  varchar(256)    NOT NULL,
      user_id int DEFAULT NULL,
      user_id1    varchar(256)    DEFAULT NULL,
      user_id2    varchar(256)    DEFAULT NULL,
      column_01   int DEFAULT NULL,
      column_02   int DEFAULT NULL,
      column_03   int DEFAULT NULL,
      column_04   int DEFAULT NULL,
      column_05   int DEFAULT NULL,
      column_06   DECIMAL(12,2)   DEFAULT NULL,
      column_07   DECIMAL(12,3)   DEFAULT NULL,
      column_08   JSON   DEFAULT NULL,
      column_09   DATETIME    DEFAULT NULL,
      column_10   DATETIME    DEFAULT NULL,
      column_11   DATE    DEFAULT NULL,
      column_12   varchar(256)    DEFAULT NULL,
      column_13   varchar(256)    DEFAULT NULL,
      column_14   varchar(256)    DEFAULT NULL,
      column_15   varchar(256)    DEFAULT NULL,
      column_16   varchar(256)    DEFAULT NULL,
      column_17   varchar(256)    DEFAULT NULL,
      column_18   varchar(256)    DEFAULT NULL,
      column_19   varchar(256)    DEFAULT NULL,
      column_20   varchar(256)    DEFAULT NULL,
      column_21   varchar(256)    DEFAULT NULL,
      column_22   varchar(256)    DEFAULT NULL,
      column_23   varchar(256)    DEFAULT NULL,
      column_24   varchar(256)    DEFAULT NULL,
      column_25   varchar(256)    DEFAULT NULL,
      column_26   varchar(256)    DEFAULT NULL,
      column_27   varchar(256)    DEFAULT NULL,
      column_28   varchar(256)    DEFAULT NULL,
      column_29   varchar(256)    DEFAULT NULL,
      column_30   varchar(256)    DEFAULT NULL,
      column_31   varchar(256)    DEFAULT NULL,
      column_32   varchar(256)    DEFAULT NULL,
      column_33   varchar(256)    DEFAULT NULL,
      column_34   varchar(256)    DEFAULT NULL,
      column_35   varchar(256)    DEFAULT NULL,
      column_36   varchar(256)    DEFAULT NULL,
      column_37   varchar(256)    DEFAULT NULL
  )
```

```
  PARTITION BY date_trunc("day", ds)
  DISTRIBUTED BY hash(id);

  -- Создать материализованное представление с предложением WHERE и
сложными выражениями.
  CREATE MATERIALIZED VIEW test_mv1
  AS
  SELECT
  ds,
  column_19,
  column_36,
  sum(column_01) as column_01_sum,
  bitmap_union(to_bitmap( user_id)) as user_id_dist_cnt,
  bitmap_union(to_bitmap(case when column_01 > 1 and column_34 IN
('1','34')   then user_id2 else null end)) as filter_dist_cnt_1,
  bitmap_union(to_bitmap( case when column_02 > 60 and column_35 IN
('11','13') then  user_id2 else null end)) as filter_dist_cnt_2,
  bitmap_union(to_bitmap(case when column_03 > 70 and column_36 IN
('21','23') then  user_id2 else null end)) as filter_dist_cnt_3,
  bitmap_union(to_bitmap(case when column_04 > 20 and column_27 IN
('31','27') then  user_id2 else null end)) as filter_dist_cnt_4,
  bitmap_union(to_bitmap( case when column_05 > 90 and column_28 IN
('41','43') then  user_id2 else null end)) as filter_dist_cnt_5
  FROM user_event
  WHERE ds >= '2023-11-02'
  GROUP BY
  ds,
  column_19,
  column_36;
```

Примеры асинхронных материализованных представлений

Следующие примеры основаны на базовых таблицах ниже:

```
CREATE TABLE `lineorder` (
  `lo_orderkey` int(11) NOT NULL COMMENT "",
  `lo_linenumber` int(11) NOT NULL COMMENT "",
  `lo_custkey` int(11) NOT NULL COMMENT "",
  `lo_partkey` int(11) NOT NULL COMMENT "",
  `lo_suppkey` int(11) NOT NULL COMMENT "",
  `lo_orderdate` int(11) NOT NULL COMMENT "",
  `lo_orderpriority` varchar(16) NOT NULL COMMENT "",
  `lo_shippriority` int(11) NOT NULL COMMENT "",
  `lo_quantity` int(11) NOT NULL COMMENT "",
  `lo_extendedprice` int(11) NOT NULL COMMENT "",
  `lo_ordtotalprice` int(11) NOT NULL COMMENT "",
  `lo_discount` int(11) NOT NULL COMMENT "",
  `lo_revenue` int(11) NOT NULL COMMENT "",
  `lo_supplycost` int(11) NOT NULL COMMENT "",
  `lo_tax` int(11) NOT NULL COMMENT "",
  `lo_commitdate` int(11) NOT NULL COMMENT "",
  `lo_shipmode` varchar(11) NOT NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`lo_orderkey`)
COMMENT "OLAP"
PARTITION BY RANGE(`lo_orderdate`)
(PARTITION p1 VALUES [("-2147483648"), ("19930101")),
PARTITION p2 VALUES [("19930101"), ("19940101")),
PARTITION p3 VALUES [("19940101"), ("19950101")),
PARTITION p4 VALUES [("19950101"), ("19960101")),
PARTITION p5 VALUES [("19960101"), ("19970101")),
PARTITION p6 VALUES [("19970101"), ("19980101")),
PARTITION p7 VALUES [("19980101"), ("19990101")))
DISTRIBUTED BY HASH(`lo_orderkey`);

CREATE TABLE IF NOT EXISTS `customer` (
  `c_custkey` int(11) NOT NULL COMMENT "",
  `c_name` varchar(26) NOT NULL COMMENT "",
  `c_address` varchar(41) NOT NULL COMMENT "",
  `c_city` varchar(11) NOT NULL COMMENT "",
  `c_nation` varchar(16) NOT NULL COMMENT "",
  `c_region` varchar(13) NOT NULL COMMENT "",
  `c_phone` varchar(16) NOT NULL COMMENT "",
  `c_mktsegment` varchar(11) NOT NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`c_custkey`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`c_custkey`);
```

```
CREATE TABLE IF NOT EXISTS `dates` (
  `d_datekey` int(11) NOT NULL COMMENT "",
  `d_date` varchar(20) NOT NULL COMMENT "",
  `d_dayofweek` varchar(10) NOT NULL COMMENT "",
  `d_month` varchar(11) NOT NULL COMMENT "",
  `d_year` int(11) NOT NULL COMMENT "",
  `d_yearmonthnum` int(11) NOT NULL COMMENT "",
  `d_yearmonth` varchar(9) NOT NULL COMMENT "",
  `d_daynuminweek` int(11) NOT NULL COMMENT "",
  `d_daynuminmonth` int(11) NOT NULL COMMENT "",
  `d_daynuminyear` int(11) NOT NULL COMMENT "",
  `d_monthnuminyear` int(11) NOT NULL COMMENT "",
  `d_weeknuminyear` int(11) NOT NULL COMMENT "",
  `d_sellingseason` varchar(14) NOT NULL COMMENT "",
  `d_lastdayinweekfl` int(11) NOT NULL COMMENT "",
  `d_lastdayinmonthfl` int(11) NOT NULL COMMENT "",
  `d_holidayfl` int(11) NOT NULL COMMENT "",
  `d_weekdayfl` int(11) NOT NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`d_datekey`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`d_datekey`);

CREATE TABLE IF NOT EXISTS `supplier` (
  `s_suppkey` int(11) NOT NULL COMMENT "",
  `s_name` varchar(26) NOT NULL COMMENT "",
  `s_address` varchar(26) NOT NULL COMMENT "",
  `s_city` varchar(11) NOT NULL COMMENT "",
  `s_nation` varchar(16) NOT NULL COMMENT "",
  `s_region` varchar(13) NOT NULL COMMENT "",
  `s_phone` varchar(16) NOT NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`s_suppkey`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`s_suppkey`);

CREATE TABLE IF NOT EXISTS `part` (
  `p_partkey` int(11) NOT NULL COMMENT "",
  `p_name` varchar(23) NOT NULL COMMENT "",
  `p_mfgr` varchar(7) NOT NULL COMMENT "",
  `p_category` varchar(8) NOT NULL COMMENT "",
  `p_brand` varchar(10) NOT NULL COMMENT "",
  `p_color` varchar(12) NOT NULL COMMENT "",
  `p_type` varchar(26) NOT NULL COMMENT "",
  `p_size` int(11) NOT NULL COMMENT "",
  `p_container` varchar(11) NOT NULL COMMENT ""
```

```
) ENGINE=OLAP
DUPLICATE KEY(`p_partkey`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`p_partkey`);

create table orders (
    dt date NOT NULL,
    order_id bigint NOT NULL,
    user_id int NOT NULL,
    merchant_id int NOT NULL,
    good_id int NOT NULL,
    good_name string NOT NULL,
    price int NOT NULL,
    cnt int NOT NULL,
    revenue int NOT NULL,
    state tinyint NOT NULL
)
PRIMARY KEY (dt, order_id)
PARTITION BY RANGE(`dt`)
( PARTITION p20210820 VALUES [('2021-08-20'), ('2021-08-21')),
PARTITION p20210821 VALUES [('2021-08-21'), ('2021-08-22')) )
DISTRIBUTED BY HASH(order_id)
PROPERTIES (
    "replication_num" = "3",
    "enable_persistent_index" = "true"
);
```

Пример 1: Создать неразделенное материализованное представление.

```
-- создать неразделенное материализованное представление, отсортированное
по lo_custkey
CREATE MATERIALIZED VIEW lo_mv1
DISTRIBUTED BY HASH(`lo_orderkey`)
ORDER BY `lo_custkey`
REFRESH ASYNC
AS
select
    lo_orderkey,
    lo_custkey,
    sum(lo_quantity) as total_quantity,
    sum(lo_revenue) as total_revenue,
    count(lo_shipmode) as shipmode_count
from lineorder
group by lo_orderkey, lo_custkey;
```

Пример 2: Создать разделенное материализованное представление.

```
-- создать разделенное материализованное представление, разделенное по `lo_orderdate` и отсортированное по `lo_custkey`.
CREATE MATERIALIZED VIEW lo_mv2
PARTITION BY `lo_orderdate`
DISTRIBUTED BY HASH(`lo_orderkey`)
ORDER BY `lo_custkey`
REFRESH ASYNC START('2023-07-01 10:00:00') EVERY (interval 1 day)
AS
select
    lo_orderkey,
    lo_orderdate,
    lo_custkey,
    sum(lo_quantity) as total_quantity,
    sum(lo_revenue) as total_revenue,
    count(lo_shipmode) as shipmode_count
from lineorder
group by lo_orderkey, lo_orderdate, lo_custkey;

-- Использовать функцию date_trunc() для разделения материализованного представления по месяцам.
CREATE MATERIALIZED VIEW order_mv1
PARTITION BY date_trunc('month', `dt`)
DISTRIBUTED BY HASH(`order_id`)
REFRESH ASYNC START('2023-07-01 10:00:00') EVERY (interval 1 day)
AS
select
    dt,
    order_id,
    user_id,
    sum(cnt) as total_cnt,
    sum(revenue) as total_revenue,
    count(state) as state_count
from orders
group by dt, order_id, user_id;
```

Пример 3: Создать асинхронное материализованное представление.

```sql
CREATE MATERIALIZED VIEW flat_lineorder
DISTRIBUTED BY HASH(`lo_orderkey`)
REFRESH MANUAL
AS
SELECT
    l.LO_ORDERKEY AS LO_ORDERKEY,
    l.LO_LINENUMBER AS LO_LINENUMBER,
    l.LO_CUSTKEY AS LO_CUSTKEY,
    l.LO_PARTKEY AS LO_PARTKEY,
    l.LO_SUPPKEY AS LO_SUPPKEY,
    l.LO_ORDERDATE AS LO_ORDERDATE,
    l.LO_ORDERPRIORITY AS LO_ORDERPRIORITY,
    l.LO_SHIPPRIORITY AS LO_SHIPPRIORITY,
    l.LO_QUANTITY AS LO_QUANTITY,
    l.LO_EXTENDEDPRICE AS LO_EXTENDEDPRICE,
    l.LO_ORDTOTALPRICE AS LO_ORDTOTALPRICE,
    l.LO_DISCOUNT AS LO_DISCOUNT,
    l.LO_REVENUE AS LO_REVENUE,
    l.LO_SUPPLYCOST AS LO_SUPPLYCOST,
    l.LO_TAX AS LO_TAX,
    l.LO_COMMITDATE AS LO_COMMITDATE,
    l.LO_SHIPMODE AS LO_SHIPMODE,
    c.C_NAME AS C_NAME,
    c.C_ADDRESS AS C_ADDRESS,
    c.C_CITY AS C_CITY,
    c.C_NATION AS C_NATION,
    c.C_REGION AS C_REGION,
    c.C_PHONE AS C_PHONE,
    c.C_MKTSEGMENT AS C_MKTSEGMENT,
    s.S_NAME AS S_NAME,
    s.S_ADDRESS AS S_ADDRESS,
    s.S_CITY AS S_CITY,
    s.S_NATION AS S_NATION,
    s.S_REGION AS S_REGION,
    s.S_PHONE AS S_PHONE,
    p.P_NAME AS P_NAME,
    p.P_MFGR AS P_MFGR,
    p.P_CATEGORY AS P_CATEGORY,
    p.P_BRAND AS P_BRAND,
    p.P_COLOR AS P_COLOR,
    p.P_TYPE AS P_TYPE,
    p.P_SIZE AS P_SIZE,
    p.P_CONTAINER AS P_CONTAINER FROM lineorder AS l
INNER JOIN customer AS c ON c.C_CUSTKEY = l.LO_CUSTKEY
```

```
INNER JOIN supplier AS s ON s.S_SUPPKEY = l.LO_SUPPKEY
INNER JOIN part AS p ON p.P_PARTKEY = l.LO_PARTKEY;
```

Пример 4: Создать разделенное материализованное представление и использовать `str2date` для преобразования ключа раздела типа STRING базовой таблицы в тип даты для материализованного представления.

```sql
-- Таблица Hive со столбцом раздела string.
CREATE TABLE `part_dates` (
  `d_date` varchar(20) DEFAULT NULL,
  `d_dayofweek` varchar(10) DEFAULT NULL,
  `d_month` varchar(11) DEFAULT NULL,
  `d_year` int(11) DEFAULT NULL,
  `d_yearmonthnum` int(11) DEFAULT NULL,
  `d_yearmonth` varchar(9) DEFAULT NULL,
  `d_daynuminweek` int(11) DEFAULT NULL,
  `d_daynuminmonth` int(11) DEFAULT NULL,
  `d_daynuminyear` int(11) DEFAULT NULL,
  `d_monthnuminyear` int(11) DEFAULT NULL,
  `d_weeknuminyear` int(11) DEFAULT NULL,
  `d_sellingseason` varchar(14) DEFAULT NULL,
  `d_lastdayinweekfl` int(11) DEFAULT NULL,
  `d_lastdayinmonthfl` int(11) DEFAULT NULL,
  `d_holidayfl` int(11) DEFAULT NULL,
  `d_weekdayfl` int(11) DEFAULT NULL,
  `d_datekey` varchar(11) DEFAULT NULL
) partition by (d_datekey);


-- Создать материализованное представление с `str2date`.
CREATE MATERIALIZED VIEW IF NOT EXISTS `test_mv`
PARTITION BY str2date(`d_datekey`,'%Y%m%d')
DISTRIBUTED BY HASH(`d_date`, `d_month`, `d_month`)
REFRESH MANUAL
AS
SELECT
`d_date` ,
  `d_dayofweek`,
  `d_month` ,
  `d_yearmonthnum` ,
  `d_yearmonth` ,
  `d_daynuminweek`,
  `d_daynuminmonth`,
  `d_daynuminyear` ,
  `d_monthnuminyear` ,
  `d_weeknuminyear` ,
  `d_sellingseason`,
  `d_lastdayinweekfl`,
  `d_lastdayinmonthfl`,
  `d_holidayfl` ,
  `d_weekdayfl`,
```

```
    `d_datekey`
FROM
 `hive_catalog`.`ssb_1g_orc`.`part_dates` ;
```

Пример 5: Создать разделенное материализованное представление с многостолбцовым выражением раздела на основе базовой таблицы из Iceberg Catalog (Spark).

Определение базовой таблицы в Spark:

```
-- Выражение раздела базовой таблицы содержит несколько столбцов и
преобразование `days`.
CREATE TABLE lineitem_days (
      l_orderkey    BIGINT,
      l_partkey     INT,
      l_suppkey     INT,
      l_linenumber  INT,
      l_quantity    DECIMAL(15, 2),
      l_extendedprice  DECIMAL(15, 2),
      l_discount    DECIMAL(15, 2),
      l_tax         DECIMAL(15, 2),
      l_returnflag  VARCHAR(1),
      l_linestatus  VARCHAR(1),
      l_shipdate    TIMESTAMP,
      l_commitdate  TIMESTAMP,
      l_receiptdate TIMESTAMP,
      l_shipinstruct VARCHAR(25),
      l_shipmode     VARCHAR(10),
      l_comment      VARCHAR(44)
) USING ICEBERG
PARTITIONED BY (l_returnflag, l_linestatus, days(l_shipdate));
```

Создать материализованное представление со столбцами разделов, сопоставленными один к одному с теми из базовой таблицы:

```
CREATE MATERIALIZED VIEW test_days
PARTITION BY (l_returnflag, l_linestatus, date_trunc('day', l_shipdate))
REFRESH DEFERRED MANUAL
AS
SELECT * FROM iceberg_catalog.test_db.lineitem_days;
```

Пример 6: Создать разделенное материализованное представление, определить TTL выражения общего раздела для него и включить семантику `force_mv` для переписывания запросов.

```
CREATE MATERIALIZED VIEW test_mv1
PARTITION BY (dt, province)
REFRESH MANUAL
PROPERTIES (
    "partition_retention_condition" = "dt >= CURRENT_DATE() - INTERVAL 3
MONTH",
    "query_rewrite_consistency" = "force_mv"
)
AS SELECT * from t1;
```

Пример 7: Создать разделенное материализованное представление с определенным ключом сортировки:

```
CREATE MATERIALIZED VIEW lo_mv2
PARTITION BY `lo_orderdate`
DISTRIBUTED BY HASH(`lo_orderkey`)
ORDER BY `lo_custkey`
REFRESH ASYNC START('2023-07-01 10:00:00') EVERY (interval 1 day)
AS
select
    lo_orderkey,
    lo_orderdate,
    lo_custkey,
    sum(lo_quantity) as total_quantity,
    sum(lo_revenue) as total_revenue,
    count(lo_shipmode) as shipmode_count
from lineorder
group by lo_orderkey, lo_orderdate, lo_custkey;
```

DROP MATERIALIZED VIEW

DROP MATERIALIZED VIEW удаляет материализованное представление.

Вы не можете удалить синхронное материализованное представление, которое находится в процессе создания, с помощью этой команды. Чтобы удалить синхронное материализованное представление, которое находится в процессе создания, см.

Синхронное материализованное представление - Удаление незавершенного материализованного представления для получения дополнительных инструкций.

> **ПОДСКАЗКА**
>
> Для выполнения этой операции требуется привилегия DROP на целевом материализованном представлении.

Синтаксис

```
DROP MATERIALIZED VIEW [IF EXISTS] [database.]mv_name
```

Параметры в квадратных скобках [] являются необязательными.

Параметры

Параметр	Обязательный	Описание
IF EXISTS	нет	Если этот параметр указан, Selena не выдаст исключение при удалении несуществующего материализованного представления. Если этот параметр не указан, система выдаст исключение при удалении несуществующего материализованного представления.
mv_name	да	Имя материализованного представления для удаления.

Примеры

Пример 1: Удалить существующее материализованное представление

1. Просмотреть все существующие материализованные представления в базе данных.

```
MySQL > SHOW MATERIALIZED VIEWS\G
*************************** 1. row ***************************
            id: 470740
          name: order_mv1
database_name: default_cluster:sr_hub
      text: SELECT `sr_hub`.`orders`.`dt` AS `dt`,
`sr_hub`.`orders`.`order_id` AS `order_id`, `sr_hub`.`orders`.`user_id` AS
`user_id`, sum(`sr_hub`.`orders`.`cnt`) AS `total_cnt`,
sum(`sr_hub`.`orders`.`revenue`) AS `total_revenue`,
count(`sr_hub`.`orders`.`state`) AS `state_count` FROM `sr_hub`.`orders`
GROUP BY `sr_hub`.`orders`.`dt`, `sr_hub`.`orders`.`order_id`,
`sr_hub`.`orders`.`user_id`
        rows: 0
1 rows in set (0.00 sec)
```

2. Удалить материализованное представление `order_mv1`.

```
DROP MATERIALIZED VIEW order_mv1;
```

3. Проверить, существует ли удаленное материализованное представление.

```
MySQL > SHOW MATERIALIZED VIEWS;
Empty set (0.01 sec)
```

Пример 2: Удалить несуществующее материализованное представление

- Если указан параметр `IF EXISTS`, Selena не выдаст исключение при удалении несуществующего материализованного представления.

```
MySQL > DROP MATERIALIZED VIEW IF EXISTS k1_k2;
Query OK, 0 rows affected (0.00 sec)
```

- Если параметр `IF EXISTS` не указан, система выдаст исключение при удалении несуществующего материализованного представления.

```
MySQL > DROP MATERIALIZED VIEW k1_k2;
ERROR 1064 (HY000): Materialized view k1_k2 is not find
```

REFRESH MATERIALIZED VIEW

REFRESH MATERIALIZED VIEW вручную обновляет конкретное асинхронное материализованное представление или его партиции.

> **ВНИМАНИЕ**
>
> Вы можете вручную обновлять только материализованные представления со стратегией обновления ASYNC или MANUAL. Проверить стратегию обновления асинхронного материализованного представления можно с помощью SHOW MATERIALIZED VIEWS. Эта операция требует привилегии REFRESH на целевом материализованном представлении.

Синтаксис

```
REFRESH MATERIALIZED VIEW [database.]mv_name
[PARTITION START ("<partition_start_date>") END ("<partition_end_date>")]
[FORCE]
[WITH { SYNC | ASYNC } MODE]
```

Параметры

Параметр	Обязательный	Описание
mv_name	да	Имя материализованного представления для ручного обновления.
PARTITION START () END ()	нет	Ручное обновление партиций в определённом временном интервале.
partition_start_date	нет	Начальная дата партиций для ручного обновления.

Параметр	Обязательный	Описание
partition_end_date	нет	Конечная дата партиций для ручного обновления.
FORCE	нет	Если вы указываете этот параметр, Selena принудительно обновляет соответствующее материализованное представление или партиции. Если вы не указываете этот параметр, Selena автоматически определяет, обновлены ли данные, и обновляет партицию только при необходимости.
WITH ... MODE	нет	Синхронный или асинхронный вызов задачи обновления. `SYNC` указывает на синхронный вызов задачи обновления, и Selena возвращает результат задачи только когда задача завершится успешно или с ошибкой. `ASYNC` указывает на асинхронный вызов задачи обновления, и Selena возвращает успех сразу после отправки задачи, оставляя задачу выполняться асинхронно в фоновом режиме. Вы можете проверить статус задачи обновления асинхронного материализованного представления, запросив представления метаданных `tasks` и `task_runs` в Information Schema Selena. Для получения дополнительной информации см. Проверка статуса выполнения асинхронного материализованного представления. По умолчанию: `ASYNC`. Поддерживается начиная с версий v1.5.2.

> **ВНИМАНИЕ**
>
> При обновлении материализованных представлений, созданных на основе внешних каталогов, Selena обновляет все партиции в материализованных представлениях.

Мониторинг

После выполнения `REFRESH MATERIALIZED VIEW` вы можете отслеживать статус и результат операции обновления следующими способами:

- Выполните `SHOW MATERIALIZED VIEWS` для просмотра статуса материализованного представления, включая время последнего обновления, состояние обновления и любые сообщения об ошибках.
- Для получения более подробной информации о выполнении, особенно для асинхронных задач обновления, запросите системное представление `information_schema.task_runs`. Это представление предоставляет записи о выполнении задач и их состояниях (PENDING, RUNNING, SUCCESS, FAILED). Для обновлений материализованных представлений поле `EXTRA_MESSAGE` включает

подробную информацию о партициях и деталях выполнения для каждой подзадачи обновления. Подробности см. в Понимание выполнения задач материализованного представления.

Примеры

Пример 1: Ручное обновление конкретного материализованного представления через асинхронный вызов.

```
REFRESH MATERIALIZED VIEW lo_mv1;

REFRESH MATERIALIZED VIEW lo_mv1 WITH ASYNC MODE;
```

Пример 2: Ручное обновление определённых партиций конкретного материализованного представления.

```
REFRESH MATERIALIZED VIEW lo_mv1
PARTITION START ("2020-02-01") END ("2020-03-01");
```

Пример 3: Принудительное обновление определённых партиций конкретного материализованного представления.

```
REFRESH MATERIALIZED VIEW lo_mv1
PARTITION START ("2020-02-01") END ("2020-03-01") FORCE;
```

Пример 4: Ручное обновление материализованного представления через синхронный вызов.

```
REFRESH MATERIALIZED VIEW lo_mv1 WITH SYNC MODE;
```

SHOW ALTER MATERIALIZED VIEW

SHOW ALTER MATERIALIZED VIEW показывает статус построения синхронных материализованных представлений.

ПОДСКАЗКА

Эта операция не требует привилегий.

Синтаксис

```
SHOW ALTER MATERIALIZED VIEW [ { FROM | IN } db_name]
```

Параметры в квадратных скобках [] являются опциональными.

Параметры

Параметр	Обязательный	Описание
db_name	нет	Имя базы данных, в которой находится материализованное представление. Если этот параметр не указан, по умолчанию используется текущая база данных.

Возвращаемые значения

Возвращаемое значение	Описание
JobId	ID задачи обновления.
TableName	Имя таблицы.
CreateTime	Время создания задачи обновления.
FinishedTime	Время завершения задачи обновления.
BaseIndexName	Имя базовой таблицы.
RollupIndexName	Имя материализованного представления.
RollupId	ID rollup материализованного представления.
TransactionId	ID транзакции, ожидающей выполнения.
State	Состояние задачи.
Msg	Сообщение об ошибке.
Progress	Прогресс задачи обновления.
Timeout	Таймаут для задачи обновления.

Примеры

Пример 1: статус построения синхронных материализованных представлений

```
MySQL > SHOW ALTER MATERIALIZED VIEW\G
*************************** 1. row ***************************
          JobId: 475991
      TableName: lineorder
     CreateTime: 2022-08-24 19:46:53
   FinishedTime: 2022-08-24 19:47:15
  BaseIndexName: lineorder
RollupIndexName: lo_mv_sync_1
       RollupId: 475992
  TransactionId: 33067
          State: FINISHED
            Msg:
       Progress: NULL
        Timeout: 86400
*************************** 2. row ***************************
          JobId: 477337
      TableName: lineorder
     CreateTime: 2022-08-24 19:47:25
   FinishedTime: 2022-08-24 19:47:45
  BaseIndexName: lineorder
RollupIndexName: lo_mv_sync_2
       RollupId: 477338
  TransactionId: 33068
          State: FINISHED
            Msg:
       Progress: NULL
        Timeout: 86400
2 rows in set (0.00 sec)
```

SHOW CREATE MATERIALIZED VIEW

SHOW CREATE MATERIALIZED VIEW показывает определение конкретного асинхронного материализованного представления.

> **ПОДСКАЗКА**
>
> Эта операция не требует привилегий.

Синтаксис

```
SHOW CREATE MATERIALIZED VIEW [database.]<mv_name>
```

Параметры в квадратных скобках [] являются опциональными.

Параметры

Параметр	Обязательный	Описание
mv_name	да	Имя материализованного представления для отображения.

Возвращаемые значения

Возвращаемое значение	Описание
Materialized View	Имя материализованного представления.
Create Materialized View	Определение материализованного представления.

Примеры

Пример 1: Отобразить определение конкретного материализованного представления

```
MySQL > SHOW CREATE MATERIALIZED VIEW lo_mv1\G
*************************** 1. row ***************************
       Materialized View: lo_mv1
Create Materialized View: CREATE MATERIALIZED VIEW `lo_mv1`
COMMENT "MATERIALIZED_VIEW"
DISTRIBUTED BY HASH(`lo_orderkey`)
REFRESH ASYNC
PROPERTIES (
"replication_num" = "3",
"storage_medium" = "HDD"
)
AS SELECT `wlc_test`.`lineorder`.`lo_orderkey` AS `lo_orderkey`,
`wlc_test`.`lineorder`.`lo_custkey` AS `lo_custkey`,
sum(`wlc_test`.`lineorder`.`lo_quantity`) AS `total_quantity`,
sum(`wlc_test`.`lineorder`.`lo_revenue`) AS `total_revenue`,
count(`wlc_test`.`lineorder`.`lo_shipmode`) AS `shipmode_count` FROM
`wlc_test`.`lineorder` GROUP BY `wlc_test`.`lineorder`.`lo_orderkey`,
`wlc_test`.`lineorder`.`lo_custkey` ORDER BY
`wlc_test`.`lineorder`.`lo_orderkey` ASC ;
1 row in set (0.01 sec)
```

SHOW MATERIALIZED VIEWS

SHOW MATERIALIZED VIEWS показывает все или одно конкретное асинхронное материализованное представление.

Начиная с версии v1.5.2, название этой команды изменено с SHOW MATERIALIZED VIEW на SHOW MATERIALIZED VIEWS.

ПОДСКАЗКА

Эта операция не требует привилегий.

Синтаксис

```
SHOW MATERIALIZED VIEWS
[FROM db_name]
[
WHERE NAME { = "mv_name" | LIKE "mv_name_matcher"}
]
```

ПРИМЕЧАНИЕ

Начиная с версии v1.5.2, команда `SHOW MATERIALIZED VIEWS` отслеживает состояние всех task_runs, если задача обновления состоит из нескольких разделов/task_runs для обновления. Только когда все task_runs успешны, `last_refresh_state` вернет `SUCCESS`.

Параметры

Параметр	Обязательный	Описание
db_name	нет	Имя базы данных, в которой находится материализованное представление. Если этот параметр не указан, по умолчанию используется текущая база данных.
mv_name	нет	Имя материализованного представления для отображения.
mv_name_matcher	нет	Шаблон для фильтрации материализованных представлений.

Возвращаемые значения

Возвращаемое значение	Описание
id	ID материализованного представления.
database_name	Имя базы данных, в которой находится материализованное представление.
name	Имя материализованного представления.

Возвращаемое значение	Описание
refresh_type	Тип обновления материализованного представления, включая ROLLUP, MANUAL, ASYNC и INCREMENTAL.
is_active	Активно ли состояние материализованного представления. Допустимые значения: `true` и `false`.
partition_type	Тип разделения материализованного представления, включая RANGE и UNPARTITIONED.
task_id	ID задачи обновления материализованного представления.
task_name	Имя задачи обновления материализованного представления.
last_refresh_start_time	Время начала последнего обновления материализованного представления.
last_refresh_finished_time	Время окончания последнего обновления материализованного представления.
last_refresh_duration	Время, затраченное на последнее обновление. Единица: секунды.
last_refresh_state	Статус последнего обновления, включая PENDING, RUNNING, FAILED и SUCCESS.
last_refresh_force_refresh	Было ли последнее обновление принудительным (FORCE refresh).
last_refresh_start_partition	Начальный раздел последнего обновления в материализованном представлении.
last_refresh_end_partition	Конечный раздел последнего обновления в материализованном представлении.
last_refresh_base_refresh_partitions	Разделы базовой таблицы, которые были обновлены при последнем обновлении.
last_refresh_mv_refresh_partitions	Разделы материализованного представления, которые были обновлены при последнем обновлении.
last_refresh_error_code	Код ошибки последнего неудачного обновления материализованного представления (если состояние материализованного представления неактивно).
last_refresh_error_message	Причина, по которой последнее обновление не удалось (если состояние материализованного представления неактивно).
rows	Количество строк данных в материализованном представлении.
text	Оператор, использованный для создания материализованного представления.

Примеры

Следующие примеры основаны на этом бизнес-сценарии:

```
-- Создание таблицы: customer
CREATE TABLE customer ( C_CUSTKEY     INTEGER NOT NULL,
                        C_NAME        VARCHAR(25) NOT NULL,
                        C_ADDRESS     VARCHAR(40) NOT NULL,
                        C_NATIONKEY   INTEGER NOT NULL,
                        C_PHONE       CHAR(15) NOT NULL,
                        C_ACCTBAL     double   NOT NULL,
                        C_MKTSEGMENT  CHAR(10) NOT NULL,
                        C_COMMENT     VARCHAR(117) NOT NULL,
                        PAD char(1) NOT NULL)
    ENGINE=OLAP
DUPLICATE KEY(`c_custkey`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`c_custkey`)
PROPERTIES (
"replication_num" = "3",
"storage_format" = "DEFAULT"
);

-- Создание MV: customer_mv
CREATE MATERIALIZED VIEW customer_mv
DISTRIBUTED BY HASH(c_custkey)
REFRESH MANUAL
PROPERTIES (
    "replication_num" = "3"
)
AS SELECT
              c_custkey, c_phone, c_acctbal, count(1) as c_count,
sum(c_acctbal) as c_sum
   FROM
              customer
   GROUP BY c_custkey, c_phone, c_acctbal;

-- Обновление MV
REFRESH MATERIALIZED VIEW customer_mv;
```

Пример 1: Показать конкретное материализованное представление.

```
mysql> SHOW MATERIALIZED VIEWS WHERE NAME='customer_mv'\G
*************************** 1. row ***************************
                        id: 10142
                      name: customer_mv
             database_name: test
              refresh_type: MANUAL
                 is_active: true
   last_refresh_start_time: 2023-02-17 10:27:33
last_refresh_finished_time: 2023-02-17 10:27:33
     last_refresh_duration: 0
        last_refresh_state: SUCCESS
             inactive_code: 0
           inactive_reason:
                      text: CREATE MATERIALIZED VIEW `customer_mv`
COMMENT "MATERIALIZED_VIEW"
DISTRIBUTED BY HASH(`c_custkey`)
REFRESH MANUAL
PROPERTIES (
"replication_num" = "3",
"storage_medium" = "HDD"
)
AS SELECT `customer`.`c_custkey`, `customer`.`c_phone`,
`customer`.`c_acctbal`, count(1) AS `c_count`, sum(`customer`.`c_acctbal`)
AS `c_sum`
FROM `test`.`customer`
GROUP BY `customer`.`c_custkey`, `customer`.`c_phone`,
`customer`.`c_acctbal`;
                      rows: 0
1 row in set (0.11 sec)
```

Пример 2: Показать материализованные представления по совпадению имени.

```
mysql> SHOW MATERIALIZED VIEWS WHERE NAME LIKE 'customer_mv'\G
*************************** 1. row ***************************
                        id: 10142
                      name: customer_mv
             database_name: test
              refresh_type: MANUAL
                 is_active: true
   last_refresh_start_time: 2023-02-17 10:27:33
last_refresh_finished_time: 2023-02-17 10:27:33
     last_refresh_duration: 0
        last_refresh_state: SUCCESS
             inactive_code: 0
           inactive_reason:
                      text: CREATE MATERIALIZED VIEW `customer_mv`
COMMENT "MATERIALIZED_VIEW"
DISTRIBUTED BY HASH(`c_custkey`)
REFRESH MANUAL
PROPERTIES (
"replication_num" = "3",
"storage_medium" = "HDD"
)
AS SELECT `customer`.`c_custkey`, `customer`.`c_phone`,
`customer`.`c_acctbal`, count(1) AS `c_count`, sum(`customer`.`c_acctbal`)
AS `c_sum`
FROM `test`.`customer`
GROUP BY `customer`.`c_custkey`, `customer`.`c_phone`,
`customer`.`c_acctbal`;
                      rows: 0
1 row in set (0.12 sec)
```

3.10 Function

CREATE FUNCTION

CREATE FUNCTION создает пользовательскую функцию (UDF). В настоящее время вы можете создавать только Java UDF, включая Scalar-функции, пользовательские агрегатные функции (UDAF), пользовательские оконные функции (UDWF) и пользовательские табличные функции (UDTF).

Подробную информацию о том, как компилировать, создавать и использовать Java UDF, см. в разделе Java UDF.

> **ПРИМЕЧАНИЕ**

> Для создания глобальной UDF у вас должна быть привилегия CREATE GLOBAL FUNCTION на уровне SYSTEM. Для создания UDF на уровне базы данных у вас должна быть привилегия CREATE FUNCTION на уровне DATABASE.

Синтаксис

```
CREATE [OR REPLACE] [GLOBAL] [AGGREGATE | TABLE] FUNCTION function_name
(arg_type [, ...])
RETURNS return_type
PROPERTIES ("key" = "value" [, ...])
```

Параметры

Параметр	Обязательный	Описание
OR REPLACE	Нет	Если существует функция с той же сигнатурой, она будет заменена. Поддерживается начиная с версии v1.5.2.
GLOBAL	Нет	Определяет, нужно ли создать глобальную UDF, поддерживается начиная с версии v1.5.2.
AGGREGATE	Нет	Определяет, нужно ли создать UDAF или UDWF.
TABLE	Нет	Определяет, нужно ли создать UDTF. Если ни `AGGREGATE`, ни `TABLE` не указаны, создается Scalar-функция.
function_name	Да	Имя функции, которую вы хотите создать. Вы можете включить имя базы данных в этот параметр, например, `db1.my_func`. Если `function_name` включает имя базы данных, UDF создается в этой базе данных. В противном случае UDF создается в текущей базе данных. Имя новой функции и её параметры не могут совпадать с существующим именем в целевой базе данных. В противном случае функция не может быть создана. Создание успешно, если имя функции совпадает, но параметры различаются.
arg_type	Да	Тип аргумента функции. Добавленный аргумент может быть представлен как `, ...`. Поддерживаемые типы данных см. в разделе Java UDF.
return_type	Да	Возвращаемый тип функции. Поддерживаемые типы данных см. в разделе Java UDF.

Параметр	Обязательный	Описание
PROPERTIES	Да	Свойства функции, которые различаются в зависимости от типа создаваемой UDF. Подробности см. в разделе Java UDF.

DROP FUNCTION

Описание

DROP FUNCTION удаляет пользовательскую функцию. Функция может быть удалена только в том случае, если её имя и тип параметра совпадают.

Только владелец пользовательской функции имеет права на её удаление.

Синтаксис

```
DROP FUNCTION function_name(arg_type [, ...])
```

Параметры

`function_name`: имя функции, которую нужно удалить.

`arg_type`: тип аргумента функции, которую нужно удалить.

Примеры

1. Удалить функцию.

```
DROP FUNCTION my_add(INT, INT)
```

SHOW FUNCTIONS

SHOW FUNCTIONS запрашивает все пользовательские (или встроенные) функции в базе данных. Если база данных не указана, по умолчанию используется текущая база данных.

Синтаксис

```
SHOW [FULL] [BUILTIN] FUNCTIONS [IN|FROM db] [LIKE 'function_pattern']
```

Параметры

```
full: указывает на отображение всех функций.

builtin: указывает на отображение функций, предоставляемых системой.

db: имя базы данных для запроса.

function_pattern: шаблон, используемый для фильтрации имен функций.
```

© ООО «Платформа Данных Селена», 2026

Примеры

```
mysql> show full functions in testDb\G
*************************** 1. row ***************************
        Signature: my_add(INT,INT)
      Return Type: INT
    Function Type: Scalar
Intermediate Type: NULL
       Properties: {"symbol":"_ZN9selena_udf6AddUdfEPNS_15FunctionContextERKN
*************************** 2. row ***************************
        Signature: my_count(BIGINT)
      Return Type: BIGINT
    Function Type: Aggregate
Intermediate Type: NULL
       Properties:
{"object_file":"http://host:port/libudasample.so","finalize_fn":"_ZN9selena_u

2 rows in set (0.00 sec)
mysql> show builtin functions in testDb like 'year%';
+---------------+
| Function Name |
+---------------+
| year          |
| years_add     |
| years_diff    |
| years_sub     |
+---------------+
2 rows in set (0.00 sec)
```

3.11 CBO statistics

ANALYZE TABLE

ANALYZE TABLE создаёт задачу ручного сбора статистики для CBO. По умолчанию ручной сбор является синхронной операцией. Вы также можете настроить его как асинхронную операцию. В асинхронном режиме после выполнения ANALYZE TABLE система сразу возвращает результат успешности выполнения запроса. Однако задача сбора будет работать в фоновом режиме, и вам не нужно ждать результата. Вы можете проверить статус задачи, выполнив SHOW ANALYZE STATUS. Асинхронный сбор подходит для таблиц с большим объёмом данных, тогда как синхронный сбор подходит для таблиц с малым объёмом данных.

Задачи ручного сбора выполняются только один раз после создания. Вам не нужно удалять задачи ручного сбора.

Этот запрос поддерживается начиная с v1.5.2.

Ручной сбор базовой статистики

Подробнее о базовой статистике см. Сбор статистики для CBO.

Синтаксис

```
ANALYZE [FULL|SAMPLE] TABLE tbl_name (col_name [,col_name])
[WITH SYNC | ASYNC MODE]
PROPERTIES (property [,property])
```

Описание параметров

- Тип сбора
  - FULL: указывает на полный сбор.
  - SAMPLE: указывает на выборочный сбор.
  - Если тип сбора не указан, по умолчанию используется полный сбор.
- `col_name`: столбцы, из которых собирается статистика. Несколько столбцов разделяются запятыми (`,`). Если этот параметр не указан, сбор выполняется для всей таблицы.
- [WITH SYNC | ASYNC MODE]: запускать ли задачу ручного сбора в синхронном или асинхронном режиме. Если этот параметр не указан, по умолчанию используется синхронный сбор.
- `PROPERTIES`: пользовательские параметры. Если `PROPERTIES` не указан, используются настройки по умолчанию из файла `fe.conf`. Фактически используемые свойства можно посмотреть в столбце `Properties` в выводе SHOW ANALYZE STATUS.

PROPERTIES	Тип	Значение по умолчанию	Описание
statistic_sample_collect_rows	INT	200000	Минимальное количество строк для сбора при выборочном сборе. Если значение параметра превышает фактическое количество строк в вашей таблице, выполняется полный сбор.

Примеры

Пример 1: Ручной полный сбор

```
-- Ручной сбор полной статистики таблицы с использованием настроек по умолчанию.
ANALYZE TABLE tbl_name;

-- Ручной сбор полной статистики таблицы с использованием настроек по умолчанию.
ANALYZE FULL TABLE tbl_name;

-- Ручной сбор статистики указанных столбцов в таблице с использованием настроек по умолчанию.
ANALYZE TABLE tbl_name(c1, c2, c3);
```

Пример 2: Ручной выборочный сбор

```
-- Ручной сбор частичной статистики таблицы с использованием настроек по умолчанию.
ANALYZE SAMPLE TABLE tbl_name;

-- Ручной сбор статистики указанных столбцов в таблице с указанием количества строк для сбора.
ANALYZE SAMPLE TABLE tbl_name (v1, v2, v3) PROPERTIES(
    "statistic_sample_collect_rows" = "1000000"
);
```

Ручной сбор гистограмм

Подробнее о гистограммах см. Сбор статистики для CBO.

Синтаксис

```
ANALYZE TABLE tbl_name UPDATE HISTOGRAM ON col_name [, col_name]
[WITH SYNC | ASYNC MODE]
[WITH N BUCKETS]
PROPERTIES (property [,property]);
```

Описание параметров

- `col_name`: столбцы, из которых собирается статистика. Несколько столбцов разделяются запятыми (`,`). Если этот параметр не указан, сбор выполняется для всей таблицы. Этот параметр обязателен для гистограмм.

- [WITH SYNC | ASYNC MODE]: запускать ли задачу ручного сбора в синхронном или асинхронном режиме. Если этот параметр не указан, по умолчанию используется синхронный сбор.

- `WITH N BUCKETS`: `N` — это количество бакетов для сбора гистограммы. Если не указано, используется значение по умолчанию из `fe.conf`.

- PROPERTIES: пользовательские параметры. Если `PROPERTIES` не указан, используются настройки по умолчанию из `fe.conf`. Фактически используемые свойства можно посмотреть в столбце `Properties` в выводе SHOW ANALYZE STATUS.

PROPERTIES	Тип	Значение по умолчанию	Описание
statistic_sample_collect_rows	INT	200000	Минимальное количество строк для сбора. Если значение параметра превышает фактическое количество строк в вашей таблице, выполняется полный сбор.
histogram_buckets_size	LONG	64	Количество бакетов по умолчанию для гистограммы.
histogram_mcv_size	INT	100	Количество наиболее часто встречающихся значений (MCV) для гистограммы.
histogram_sample_ratio	FLOAT	0.1	Коэффициент выборки для гистограммы.
histogram_max_sample_row_cou	LONG	10000000	Максимальное количество строк для сбора для гистограммы.

PROPERTIES	Тип	Значение по умолчанию	Описание
nt			

Количество строк для сбора для гистограммы контролируется несколькими параметрами. Это большее значение между `statistic_sample_collect_rows` и количеством строк таблицы * `histogram_sample_ratio`. Количество не может превышать значение, указанное в `histogram_max_sample_row_count`. Если значение превышено, приоритет имеет `histogram_max_sample_row_count`.

Примеры

```
-- Ручной сбор гистограмм для v1 с использованием настроек по умолчанию.
ANALYZE TABLE tbl_name UPDATE HISTOGRAM ON v1;

-- Ручной сбор гистограмм для v1 и v2 с 32 бакетами, 32 MCV и 50%
коэффициентом выборки.
ANALYZE TABLE tbl_name UPDATE HISTOGRAM ON v1,v2 WITH 32 BUCKETS
PROPERTIES(
   "histogram_mcv_size" = "32",
   "histogram_sample_ratio" = "0.5"
);
```

Ссылки

SHOW ANALYZE STATUS: просмотр статуса задачи ручного сбора.

KILL ANALYZE: отмена выполняющейся задачи ручного сбора.

Подробнее о сборе статистики для CBO см. Сбор статистики для CBO.

CREATE ANALYZE

CREATE ANALYZE настраивает задачу автоматического сбора статистики для CBO.

По умолчанию Selena автоматически собирает полную статистику таблицы. Система проверяет наличие обновлений данных каждые 5 минут. Если обнаружено изменение данных, автоматически запускается сбор данных. Если вы не хотите использовать автоматический полный сбор, вы можете установить параметр конфигурации FE `enable_collect_full_statistic` в `false` и настроить задачу сбора.

Различия в поведении:

- До версий 3.2.12 и 3.3.4 для создания пользовательской задачи автоматического сбора необходимо отключить автоматический полный сбор (`enable_collect_full_statistic = false`). В противном случае пользовательские задачи не будут вступать в силу.
- Начиная с этих версий вы можете создавать задачи анализа и сохранять системные задачи. Пользовательские задачи переопределяют системные задачи в случае конфликтов.

Этот запрос поддерживается начиная с v1.5.2.

Синтаксис

```
-- Автоматический сбор статистики для всех баз данных.
CREATE ANALYZE [FULL|SAMPLE] ALL PROPERTIES (property [,property])

-- Автоматический сбор статистики для всех таблиц в базе данных.
CREATE ANALYZE [FULL|SAMPLE] DATABASE db_name
PROPERTIES (property [,property])

-- Автоматический сбор статистики для указанных столбцов в таблице.
CREATE ANALYZE [FULL|SAMPLE] [IF NOT EXISTS] TABLE tbl_name (col_name
[,col_name])
PROPERTIES (property [,property])
```

Описание параметров

- Тип сбора
  - FULL: указывает на полный сбор.
  - SAMPLE: указывает на выборочный сбор.
  - Если тип сбора не указан, по умолчанию используется выборочный сбор.
- `col_name`: столбцы, из которых собирается статистика. Несколько столбцов разделяются запятыми (`,`). Если этот параметр не указан, сбор выполняется для всей таблицы.
- `PROPERTIES`: пользовательские параметры. Если `PROPERTIES` не указан, используются настройки по умолчанию из `fe.conf`. Фактически используемые свойства можно посмотреть в столбце `Properties` в выводе SHOW ANALYZE JOB.

PROPERTIES	Тип	Значение по умолчанию	Описание
statistic_auto_collect_ratio	FLOAT	0.8	Порог для определения того, является ли статистика для автоматического сбора здоровой. Если состояние статистики ниже этого порога, запускается автоматический сбор.
statistics_max_full_collect_data_size	INT	100	Размер наибольшей partition для автоматического сбора данных. Единица измерения: ГБ. Если partition превышает это значение, полный сбор отменяется, и вместо этого выполняется выборочный сбор.
statistic_sample_collect_rows	INT	200000	Минимальное количество строк для сбора. Если значение параметра превышает фактическое количество строк в вашей таблице, выполняется полный сбор.

Примеры

Пример 1: Автоматический полный сбор

```
-- Автоматический сбор полной статистики для всех баз данных.
CREATE ANALYZE ALL;

-- Автоматический сбор полной статистики для базы данных.
CREATE ANALYZE DATABASE db_name;

-- Автоматический сбор полной статистики для всех таблиц в базе данных.
CREATE ANALYZE FULL DATABASE db_name;

-- Автоматический сбор полной статистики для указанных столбцов в таблице.
CREATE ANALYZE TABLE tbl_name(c1, c2, c3);
```

Пример 2: Автоматический выборочный сбор

```
-- Автоматический сбор статистики для всех таблиц в базе данных с использованием настроек по умолчанию.
CREATE ANALYZE SAMPLE DATABASE db_name;

-- Автоматический сбор статистики для указанных столбцов в таблице с указанием состояния статистики и количества строк для сбора.
CREATE ANALYZE SAMPLE TABLE tbl_name(c1, c2, c3) PROPERTIES(
   "statistic_auto_collect_ratio" = "0.5",
   "statistic_sample_collect_rows" = "1000000"
);
```

Ссылки

SHOW ANALYZE JOB: просмотр статуса пользовательской задачи сбора.

DROP ANALYZE: удаление пользовательской задачи сбора.

KILL ANALYZE: отмена выполняющейся пользовательской задачи сбора.

Подробнее о сборе статистики для CBO см. Сбор статистики для CBO.

DROP ANALYZE

DROP ANALYZE удаляет пользовательскую задачу сбора статистики.

По умолчанию Selena автоматически собирает полную статистику таблицы. Проверка обновлений данных выполняется каждые 5 минут. Если обнаружено изменение данных, автоматически запускается сбор данных. Если вы не хотите использовать автоматический полный сбор, вы можете установить параметр конфигурации FE `enable_collect_full_statistic` в `false` и настроить пользовательскую задачу сбора.

Эта команда поддерживается начиная с версии v1.5.2.

Синтаксис

```
DROP ANALYZE <ID>
```

Идентификатор задачи можно получить с помощью команды SHOW ANALYZE JOB.

Примеры

```
DROP ANALYZE 266030;
```

Ссылки

CREATE ANALYZE: настройка автоматической задачи сбора статистики.

SHOW ANALYZE JOB: просмотр статуса автоматической задачи сбора статистики.

KILL ANALYZE: отмена выполняющейся пользовательской задачи сбора статистики.

Для получения дополнительной информации о сборе статистики для CBO см. Сбор статистики для CBO.

DROP STATS

DROP STATS удаляет статистику CBO, которая включает базовую статистику и гистограммы. Для получения дополнительной информации см. Сбор статистики для CBO.

Вы можете удалить статистическую информацию, которая вам не нужна. При удалении статистики удаляются как данные, так и метаданные статистики, а также статистика из просроченного кэша. Обратите внимание, что если выполняется автоматическая задача сбора, ранее удалённая статистика может быть собрана снова. Вы можете использовать `SHOW ANALYZE STATUS` для просмотра истории задач сбора.

Эта команда поддерживается начиная с версии v1.5.2.

Синтаксис

Удаление базовой статистики

```
DROP STATS tbl_name
```

Удаление гистограмм

```
ANALYZE TABLE tbl_name DROP HISTOGRAM ON col_name [, col_name];
```

Ссылки

Для получения дополнительной информации о сборе статистики для CBO см. Сбор статистики для CBO.

KILL ANALYZE

KILL ANALYZE отменяет **выполняющуюся** задачу сбора статистики, включая ручные и пользовательские автоматические задачи.

Эта команда поддерживается начиная с версии v1.5.2.

Синтаксис

```
KILL ANALYZE <ID>
```

Идентификатор задачи для ручной задачи сбора можно получить из SHOW ANALYZE STATUS. Идентификатор задачи для пользовательской задачи сбора можно получить из SHOW ANALYZE JOB.

Ссылки

SHOW ANALYZE STATUS

SHOW ANALYZE JOB

Для получения дополнительной информации о сборе статистики для CBO см. Сбор статистики для CBO.

SHOW ANALYZE JOB

SHOW ANALYZE JOB просматривает информацию и статус пользовательских задач сбора статистики.

По умолчанию Selena автоматически собирает полную статистику таблицы. Она проверяет наличие обновлений данных каждые 5 минут. Если обнаружено изменение данных, сбор данных запускается автоматически. Если вы не хотите использовать автоматический полный сбор, вы можете установить параметр конфигурации FE `enable_collect_full_statistic` в `false` и настроить пользовательскую задачу сбора.

Эта команда поддерживается начиная с версии v1.5.2.

Синтаксис

```
SHOW ANALYZE JOB [WHERE]
```

Вы можете фильтровать результаты, используя предложение WHERE. Команда возвращает следующие столбцы.

Столбец	Описание
Id	ID задачи сбора статистики.
Database	Имя базы данных.
Table	Имя таблицы.
Columns	Имена столбцов.
Type	Тип статистики, включая `FULL` и `SAMPLE`.
Schedule	Тип планирования. Тип `SCHEDULE` для автоматической задачи.
Properties	Пользовательские параметры.
Status	Статус задачи, включая PENDING, RUNNING, SUCCESS и FAILED.
LastWorkTime	Время последнего сбора.
Reason	Причина сбоя задачи. Возвращается NULL, если задача выполнена успешно.

Примеры

```
-- Просмотреть все пользовательские задачи сбора статистики.
SHOW ANALYZE JOB

-- Просмотреть пользовательские задачи сбора статистики базы данных `test`.
SHOW ANALYZE JOB where `database` = 'test';
```

Ссылки

CREATE ANALYZE: настроить автоматическую задачу сбора статистики.

DROP ANALYZE: удалить пользовательскую задачу сбора статистики.

KILL ANALYZE: отменить выполняющуюся пользовательскую задачу сбора статистики.

Для получения дополнительной информации о сборе статистики для CBO см. Gather statistics for CBO.

SHOW ANALYZE STATUS

SHOW ANALYZE STATUS отображает статус задач сбора статистики.

Эта команда не может использоваться для просмотра статуса пользовательских задач сбора статистики. Для просмотра статуса пользовательских задач сбора статистики используйте SHOW ANALYZE JOB.

Эта команда поддерживается начиная с версии v1.5.2.

Синтаксис

```
SHOW ANALYZE STATUS [WHERE]
```

Вы можете использовать `LIKE` или `WHERE` для фильтрации возвращаемой информации.

Эта команда возвращает следующие столбцы.

Имя столбца	Описание
Id	ID задачи сбора статистики.
Database	Имя базы данных.
Table	Имя таблицы.
Columns	Имена столбцов.
Type	Тип статистики, включая FULL, SAMPLE и HISTOGRAM.
Schedule	Тип планирования. `ONCE` означает ручной запуск, `SCHEDULE` означает автоматический.
Status	Статус задачи.
StartTime	Время начала выполнения задачи.
EndTime	Время окончания выполнения задачи.
Properties	Пользовательские параметры.
Reason	Причина неудачного выполнения задачи. Возвращается NULL, если выполнение было успешным.

См. также

ANALYZE TABLE: создание задачи ручного сбора статистики.

KILL ANALYZE: отмена выполняющейся пользовательской задачи сбора статистики.

Для получения дополнительной информации о сборе статистики для CBO см. Сбор статистики для CBO.

SHOW META

SHOW META отображает метаданные статистики CBO, включая базовую статистику и гистограммы.

Эта команда поддерживается начиная с версии v1.5.2.

Просмотр метаданных базовой статистики

Синтаксис

```
SHOW STATS META [WHERE]
```

Эта команда возвращает следующие столбцы.

Столбец	Описание
Database	Имя базы данных.
Table	Имя таблицы.
Columns	Имена столбцов.
Type	Тип статистики. `FULL` означает полный сбор, `SAMPLE` означает выборочный сбор.
UpdateTime	Время последнего обновления статистики для текущей таблицы.
Properties	Пользовательские параметры.
Healthy	Состояние статистической информации.

Просмотр метаданных гистограмм

Синтаксис

```
SHOW HISTOGRAM META [WHERE];
```

Эта команда возвращает следующие столбцы.

Столбец	Описание
Database	Имя базы данных.

Столбец	Описание
Table	Имя таблицы.
Column	Столбцы.
Type	Тип статистики. Для гистограмм значение равно `HISTOGRAM`.
UpdateTime	Время последнего обновления статистики для текущей таблицы.
Properties	Пользовательские параметры.

См. также

Для получения дополнительной информации о сборе статистики для CBO см. Сбор статистики для CBO.

3.12 Backup and restore

BACKUP

Selena поддерживает резервное копирование и восстановление следующих объектов:

- Внутренние базы данных, таблицы (всех типов и стратегий партиционирования) и партиции
- Метаданные внешних каталогов (поддерживается начиная с v1.5.2)
- Синхронные материализованные представления и асинхронные материализованные представления
- Логические представления (поддерживается начиная с v1.5.2)
- Пользовательские функции (поддерживается начиная с v1.5.2)

ПОДСКАЗКА

Для обзора резервного копирования и восстановления см. руководство по резервному копированию и восстановлению.

BACKUP — это асинхронная операция. Вы можете проверить статус задачи BACKUP с помощью SHOW BACKUP или отменить задачу BACKUP с помощью CANCEL BACKUP. Вы можете просмотреть информацию о снимке с помощью SHOW SNAPSHOT.

> **ВНИМАНИЕ**
>
> - Shared-data кластеры Selena не поддерживают BACKUP и RESTORE данных.

> - В каждой базе данных одновременно может выполняться только одна задача BACKUP или RESTORE. В противном случае Selena возвращает ошибку.
> - Selena не поддерживает указание алгоритма сжатия данных для резервного копирования.

Требования к привилегиям

В версиях ранее v1.5.2 пользователи с привилегией `admin_priv` могут выполнять эту операцию. В более поздних версиях для резервного копирования конкретного объекта пользователи должны иметь привилегию REPOSITORY на уровне системы и привилегию EXPORT для соответствующей таблицы или всех таблиц в соответствующей базе данных. Например:

- Предоставить роли разрешение на экспорт данных из указанной таблицы.

```
GRANT REPOSITORY ON SYSTEM TO ROLE backup_tbl;
GRANT EXPORT ON TABLE <table_name> TO ROLE backup_tbl;
```

- Предоставить роли разрешение на экспорт данных из всех таблиц в указанной базе данных.

```
GRANT REPOSITORY ON SYSTEM TO ROLE backup_db;
GRANT EXPORT ON ALL TABLES IN DATABASE <database_name> TO ROLE
backup_db;
```

- Предоставить роли разрешение на экспорт данных из всех таблиц во всех базах данных.

```
GRANT REPOSITORY ON SYSTEM TO ROLE backup;
GRANT EXPORT ON ALL TABLES IN ALL DATABASES TO ROLE backup;
```

Синтаксис, совместимый с более ранними версиями

```
BACKUP SNAPSHOT <db_name>.<snapshot_name>
TO <repository_name>
[ ON ( <table_name> [ PARTITION ( <partition_name> [, ...] ) ]
       [, ...] ) ]
[ PROPERTIES ("key"="value" [, ...] ) ]
```

Параметры

Параметр	Описание
db_name	Имя базы данных, в которой хранятся данные для резервного копирования.
snapshot_name	Укажите имя для снимка данных. Глобально уникальное.
repository_name	Имя репозитория. Вы можете создать репозиторий с помощью CREATE REPOSITORY.
ON	Имя таблиц для резервного копирования. Если этот параметр не указан, создается резервная копия всей базы данных.
PARTITION	Имя партиций для резервного копирования. Если этот параметр не указан, создается резервная копия всей таблицы.
PROPERTIES	Свойства снимка данных. Допустимые ключи: • `type`: Тип резервного копирования. В настоящее время поддерживается только полное резервное копирование `FULL`. По умолчанию: `FULL`. • `timeout`: Тайм-аут задачи. Единица измерения: секунды. По умолчанию: `86400`.

Синтаксис, поддерживаемый начиная с v1.5.2

```
-- Резервное копирование метаданных внешнего каталога.
BACKUP { ALL EXTERNAL CATALOGS | EXTERNAL CATALOG[S] (<catalog_name> [,
...]) }
[ DATABASE <db_name> ] SNAPSHOT [<db_name>.]<snapshot_name>
TO <repository_name>
[ PROPERTIES ("key"="value" [, ...] ) ]

-- Резервное копирование баз данных, таблиц, партиций, материализованных
представлений, логических представлений или UDF.

BACKUP [ DATABASE <db_name> ] SNAPSHOT [<db_name>.]<snapshot_name>
TO <repository_name>
[ ON ( backup_object [, ...] )]
[ PROPERTIES ("key"="value" [, ...] ) ]

backup_object ::= [
    { ALL TABLE[S]             | TABLE[S] <table_name>[, TABLE[S]
<table_name> ...] } |
    { ALL MATERIALIZED VIEW[S] | MATERIALIZED VIEW[S] <mv_name>[,
MATERIALIZED VIEW[S] <mv_name> ...] } |
    { ALL VIEW[S]              | VIEW[S] <view_name>[, VIEW[S] <view_name>
...] } |
    { ALL FUNCTION[S]          | FUNCTION[S] <udf_name>[, FUNCTION[S]
<udf_name> ...] } |
     <table_name> PARTITION (<partition_name>[, ...]) ]
```

Параметры

Параметр	Описание
ALL EXTERNAL CATALOGS	Создает резервную копию метаданных всех внешних каталогов.
catalog_name	Имя внешнего каталога(ов), которые необходимо скопировать.
DATABASE db_name	Имя базы данных, к которой принадлежит объект(ы) или снимок. Вы можете указать либо `DATABASE <db_name>`, либо `<db_name>.`.
db_name.	Имя базы данных, к которой принадлежит объект(ы) или снимок. Вы можете указать либо `DATABASE <db_name>`, либо `<db_name>.`.
snapshot_name	Имя снимка данных. Глобально уникальное.
repository_name	Имя репозитория. Вы можете создать репозиторий с помощью CREATE REPOSITORY.
ON	Объект для резервного копирования. Если этот параметр не указан, создается резервная копия всей базы данных.

Параметр	Описание
table_name	Имя таблицы(таблиц) для резервного копирования.
mv_name	Имя материализованного представления(представлений) для резервного копирования.
view_name	Имя логического представления(представлений) для резервного копирования.
udf_name	Имя UDF(UDF-ов) для резервного копирования.
PARTITION	Имя партиций для резервного копирования. Если этот параметр не указан, создается резервная копия всей таблицы.
PROPERTIES	Свойства снимка данных. Допустимые ключи: • `type`: Тип резервного копирования. В настоящее время поддерживается только полное резервное копирование `FULL`. По умолчанию: `FULL`. • `timeout`: Тайм-аут задачи. Единица измерения: секунды. По умолчанию: `86400`.

Примеры

Примеры с синтаксисом, совместимым с более ранними версиями

Пример 1: Резервное копирование базы данных `example_db` в репозиторий `example_repo`.

```
BACKUP SNAPSHOT example_db.snapshot_label1
TO example_repo
PROPERTIES ("type" = "full");
```

Пример 2: Резервное копирование таблицы `example_tbl` в `example_db` в `example_repo`.

```
BACKUP SNAPSHOT example_db.snapshot_label2
TO example_repo
ON (example_tbl);
```

Пример 2: Резервное копирование партиций `p1` и `p2` таблицы `example_tbl` и таблицы `example_tbl2` в `example_db` в `example_repo`.

```
BACKUP SNAPSHOT example_db.snapshot_label3
TO example_repo
ON(
    example_tbl PARTITION (p1, p2),
    example_tbl2
);
```

Примеры с синтаксисом, поддерживаемым начиная с v1.5.2

Пример 1: Резервное копирование базы данных.

```
BACKUP DATABASE sr_hub SNAPSHOT sr_hub_backup TO test_repo;
```

Пример 2: Резервное копирование таблицы(таблиц) в базе данных.

```
-- Резервное копирование одной таблицы.
BACKUP DATABASE sr_hub SNAPSHOT sr_member_backup
TO test_repo
ON (TABLE sr_member);

-- Резервное копирование нескольких таблиц.
BACKUP DATABASE sr_hub SNAPSHOT sr_core_backup
TO test_repo
ON (TABLE sr_member, TABLE sr_pmc);

-- Резервное копирование всех таблиц.
BACKUP DATABASE sr_hub SNAPSHOT sr_all_backup
TO test_repo
ON (ALL TABLES);
```

Пример 3: Резервное копирование партиции(партиций) в таблице.

```
-- Резервное копирование одной партиции.
BACKUP DATABASE sr_hub SNAPSHOT sr_par_backup
TO test_repo
ON (TABLE sr_member PARTITION (p1));

-- Резервное копирование нескольких партиций.
BACKUP DATABASE sr_hub SNAPSHOT sr_par_backup
TO test_repo
ON (TABLE sr_member PARTITION (p1,p2,p3));
```

Пример 4: Резервное копирование материализованного представления(представлений) в базе данных.

```
-- Резервное копирование одного материализованного представления.
BACKUP DATABASE sr_hub SNAPSHOT sr_mv1_backup
TO test_repo
ON (MATERIALIZED VIEW sr_mv1);

-- Резервное копирование нескольких материализованных представлений.
BACKUP DATABASE sr_hub SNAPSHOT sr_mv2_backup
TO test_repo
ON (MATERIALIZED VIEW sr_mv1, MATERIALIZED VIEW sr_mv2);

-- Резервное копирование всех материализованных представлений.
BACKUP DATABASE sr_hub SNAPSHOT sr_mv3_backup
TO test_repo
ON (ALL MATERIALIZED VIEWS);
```

Пример 5: Резервное копирование логического представления(представлений) в базе данных.

```
-- Резервное копирование одного логического представления.
BACKUP DATABASE sr_hub SNAPSHOT sr_view1_backup
TO test_repo
ON (VIEW sr_view1);

-- Резервное копирование нескольких логических представлений.
BACKUP DATABASE sr_hub SNAPSHOT sr_view2_backup
TO test_repo
ON (VIEW sr_view1, VIEW sr_view2);

-- Резервное копирование всех логических представлений.
BACKUP DATABASE sr_hub SNAPSHOT sr_view3_backup
TO test_repo
ON (ALL VIEWS);
```

Пример 6: Резервное копирование UDF(UDF-ов) в базе данных.

```
-- Резервное копирование одного UDF.
BACKUP DATABASE sr_hub SNAPSHOT sr_udf1_backup
TO test_repo
ON (FUNCTION sr_udf1);

-- Резервное копирование нескольких UDF.
BACKUP DATABASE sr_hub SNAPSHOT sr_udf2_backup
TO test_repo
ON (FUNCTION sr_udf1, FUNCTION sr_udf2);

-- Резервное копирование всех UDF.
BACKUP DATABASE sr_hub SNAPSHOT sr_udf3_backup
TO test_repo
ON (ALL FUNCTIONS);
```

Примечания по использованию

- Выполнение операций резервного копирования и восстановления на глобальном уровне, уровне базы данных, таблицы и партиции требует различных привилегий.
- В каждой базе данных одновременно может выполняться только одна задача BACKUP или RESTORE. В противном случае Selena возвращает ошибку.
- Поскольку задачи BACKUP и RESTORE занимают много ресурсов вашего кластера Selena, вы можете выполнять резервное копирование и восстановление данных, когда ваш кластер Selena не сильно загружен.
- Selena не поддерживает указание алгоритмов сжатия данных для резервного копирования.
- Поскольку данные копируются как снимки, данные, загруженные после создания снимка, не включаются в снимок. Поэтому, если вы загружаете данные в старый кластер после создания снимка и до завершения задачи RESTORE, вам также необходимо загрузить данные в кластер, в который восстанавливаются данные. Рекомендуется загружать данные в оба кластера параллельно в течение некоторого времени после завершения миграции данных, а затем переносить ваше приложение в новый кластер после проверки правильности данных и сервисов.
- До завершения задачи RESTORE вы не можете выполнять операции с восстанавливаемой таблицей.
- Таблицы Primary Key не могут быть восстановлены в кластер Selena версии ранее v1.5.2.
- Вам не нужно создавать восстанавливаемую таблицу в новом кластере перед восстановлением. Задача RESTORE создает её автоматически.

- Если существует существующая таблица с именем, совпадающим с именем восстанавливаемой таблицы, Selena сначала проверяет, совпадает ли схема существующей таблицы со схемой восстанавливаемой таблицы. Если схемы совпадают, Selena перезаписывает существующую таблицу данными из снимка. Если схема не совпадает, задача RESTORE завершается с ошибкой. Вы можете либо переименовать восстанавливаемую таблицу с помощью ключевого слова `AS`, либо удалить существующую таблицу перед восстановлением данных.
- Если задача RESTORE перезаписывает существующую базу данных, таблицу или партицию, перезаписанные данные не могут быть восстановлены после того, как задача переходит в фазу COMMIT. Если задача RESTORE завершается неудачей или отменяется на этом этапе, данные могут быть повреждены и недоступны. В этом случае вы можете только повторить операцию RESTORE и дождаться завершения задачи. Поэтому мы рекомендуем не восстанавливать данные путем перезаписи, если вы не уверены, что текущие данные больше не используются. Операция перезаписи сначала проверяет согласованность метаданных между снимком и существующей базой данных, таблицей или партицией. Если обнаружено несоответствие, операция RESTORE не может быть выполнена.
- В настоящее время Selena не поддерживает резервное копирование и восстановление данных конфигурации, связанных с учетными записями пользователей, привилегиями и группами ресурсов.
- В настоящее время Selena не поддерживает резервное копирование и восстановление отношения Colocate Join между таблицами.

CANCEL BACKUP

Отменяет выполняющуюся задачу BACKUP в указанной базе данных.

Синтаксис

```
CANCEL BACKUP { FROM <db_name> | FOR EXTERNAL CATALOG }
```

Параметры

Параметр	Описание
db_name	Имя базы данных, к которой относится задача BACKUP.
FOR EXTERNAL CATALOG	Отменяет выполняющуюся задачу BACKUP для метаданных внешнего catalog.

Примеры

Пример 1: Отменяет задачу BACKUP в базе данных `example_db`.

```
CANCEL BACKUP FROM example_db;
```

CANCEL RESTORE

Отменяет выполняющуюся задачу RESTORE в указанной базе данных.

> **ВНИМАНИЕ**
>
> Если задача RESTORE отменена во время фазы COMMIT, восстановленные данные будут повреждены и недоступны. В этом случае вы можете только выполнить операцию RESTORE снова и дождаться завершения задачи.

Синтаксис

```
CANCEL RESTORE { FROM <db_name> | FOR EXTERNAL CATALOG }
```

Параметры

Параметр	Описание
db_name	Имя базы данных, к которой относится задача RESTORE.
FOR EXTERNAL CATALOG	Отменяет выполняющуюся задачу RESTORE для метаданных внешнего catalog.

Примеры

Пример 1: Отменяет задачу RESTORE в базе данных `example_db`.

```
CANCEL RESTORE FROM example_db;
```

CREATE REPOSITORY

Создает репозиторий в удаленной системе хранения, который используется для хранения снимков данных для резервного копирования и восстановления данных.

ПОДСКАЗКА

Для обзора резервного копирования и восстановления, пожалуйста, см. руководство по резервному копированию и восстановлению.

> **ВНИМАНИЕ**
>
> Только пользователи с привилегией ADMIN могут создавать репозиторий.

Подробные инструкции по удалению репозитория см. в DROP REPOSITORY.

Синтаксис

```
CREATE [READ ONLY] REPOSITORY <repository_name>
WITH BROKER
ON LOCATION "<repository_location>"
PROPERTIES ("key"="value", ...)
```

Параметры

Параметр	Описание
READ ONLY	Создать репозиторий только для чтения. Обратите внимание, что вы можете только восстанавливать данные из репозитория только для чтения. При создании одного и того же репозитория для двух cluster для миграции данных вы можете создать репозиторий только для чтения для нового cluster и предоставить ему только права RESTORE.
repository_name	Имя репозитория. О соглашениях по именованию см. Системные ограничения.
repository_location	Расположение репозитория в удаленной системе хранения.
PROPERTIES	Метод учетных данных для доступа к удаленной системе хранения.

PROPERTIES:

Selena поддерживает создание репозиториев в HDFS, AWS S3 и Google GCS.

- Для HDFS:

- "username": Имя пользователя учетной записи, которую вы хотите использовать для доступа к NameNode cluster HDFS.
- "password": Пароль учетной записи, которую вы хотите использовать для доступа к NameNode cluster HDFS.

- Для AWS S3:
  - "aws.s3.use_instance_profile": Разрешить ли использовать instance profile и assumed role в качестве методов учетных данных для доступа к AWS S3. По умолчанию: `false`.
    - Если вы используете учетные данные на основе IAM пользователя (Access Key и Secret Key) для доступа к AWS S3, вам не нужно указывать этот параметр, и вам нужно указать "aws.s3.access_key", "aws.s3.secret_key" и "aws.s3.region".
    - Если вы используете Instance Profile для доступа к AWS S3, вам нужно установить этот параметр в `true` и указать "aws.s3.region".
    - Если вы используете Assumed Role для доступа к AWS S3, вам нужно установить этот параметр в `true` и указать "aws.s3.iam_role_arn" и "aws.s3.region".
  - "aws.s3.access_key": Access Key ID, который вы можете использовать для доступа к Amazon S3 bucket.
  - "aws.s3.secret_key": Secret Access Key, который вы можете использовать для доступа к Amazon S3 bucket.
  - "aws.s3.iam_role_arn": ARN IAM роли, которая имеет привилегии на AWS S3 bucket, где хранятся ваши файлы данных. Если вы хотите использовать assumed role в качестве метода учетных данных для доступа к AWS S3, вы должны указать этот параметр. Затем Selena примет эту роль при анализе ваших данных Hive с помощью Hive catalog.
  - "aws.s3.region": Регион, в котором находится ваш AWS S3 bucket. Пример: `us-west-1`.

> **ПРИМЕЧАНИЕ**
>
> Selena поддерживает создание репозиториев в AWS S3 только по протоколу S3A. Поэтому при создании репозиториев в AWS S3 вы должны заменить `s3://` в S3 URI,

> который вы передаете как расположение репозитория в `ON LOCATION`, на `s3a://`.

- Для Google GCS:
  - "fs.s3a.access.key": Access Key, который вы можете использовать для доступа к Google GCS bucket.
  - "fs.s3a.secret.key": Secret Key, который вы можете использовать для доступа к Google GCS bucket.
  - "fs.s3a.endpoint": Endpoint, который вы можете использовать для доступа к Google GCS bucket. Не указывайте `https` в адресе endpoint.

> **ПРИМЕЧАНИЕ**
>
> Selena поддерживает создание репозиториев в Google GCS только по протоколу S3A. Поэтому при создании репозиториев в Google GCS вы должны заменить префикс в GCS URI, который вы передаете как расположение репозитория в `ON LOCATION`, на `s3a://`.

- Для MinIO:
  - "aws.s3.access_key": Access Key, который вы можете использовать для доступа к MinIO bucket.
  - "aws.s3.secret_key": Secret Key, который вы можете использовать для доступа к MinIO bucket.
  - "aws.s3.endpoint": Endpoint, который вы можете использовать для доступа к MinIO bucket.

Примеры

Пример 1: Создать репозиторий с именем `hdfs_repo` в Apache™ Hadoop® cluster.

```
CREATE REPOSITORY hdfs_repo
WITH BROKER
ON LOCATION "hdfs://x.x.x.x:yyyy/repo_dir/backup"
PROPERTIES(
    "username" = "xxxxxxxx",
    "password" = "yyyyyyyy"
);
```

Пример 2: Создать репозиторий только для чтения с именем `s3_repo` в Amazon S3 bucket `bucket_s3`.

```
CREATE READ ONLY REPOSITORY s3_repo
WITH BROKER
ON LOCATION "s3a://bucket_s3/backup"
PROPERTIES(
    "aws.s3.access_key" = "XXXXXXXXXXXXXXXXX",
    "aws.s3.secret_key" = "yyyyyyyyyyyyyyyyy",
    "aws.s3.region" = "us-east-1"
);
```

Пример 3: Создать репозиторий с именем `gcs_repo` в Google GCS bucket `bucket_gcs`.

```
CREATE REPOSITORY gcs_repo
WITH BROKER
ON LOCATION "s3a://bucket_gcs/backup"
PROPERTIES(
    "fs.s3a.access.key" = "xxxxxxxxxxxxxxxxxxxx",
    "fs.s3a.secret.key" = "yyyyyyyyyyyyyyyyyyyy",
    "fs.s3a.endpoint" = "storage.googleapis.com"
);
```

Пример 4: Создать репозиторий с именем `minio_repo` в MinIO bucket `bucket_minio`.

```
CREATE REPOSITORY minio_repo
WITH BROKER
ON LOCATION "s3://bucket_minio/backup"
PROPERTIES(
    "aws.s3.access_key" = "XXXXXXXXXXXXXXXXX",
    "aws.s3.secret_key" = "yyyyyyyyyyyyyyyyy",
    "aws.s3.endpoint" = "http://minio:9000"
);
```

Пример 5: Создать репозиторий с именем `kerberos_repo` в Apache™ Hadoop® cluster с включенным Kerberos.

```
CREATE REPOSITORY kerberos_repo
WITH BROKER
ON LOCATION "hdfs://nameservices1/user/hive/backup"
PROPERTIES(
 "hadoop.security.authentication" = "kerberos",
 "kerberos_principal" = "sr/cs01.selena.com@SELENA.COM",
 "kerberos_keytab" = "/home/disk1/user1/sr.keytab"
);
```

DROP REPOSITORY

Удаляет репозиторий. Репозитории используются для хранения снимков данных для резервного копирования и восстановления данных.

> **ВНИМАНИЕ**
>
> - Только пользователь root и superuser могут удалить репозиторий.
> - Эта операция удаляет только mapping репозитория в Selena и не удаляет фактические данные. Вам нужно удалить их вручную в удаленной системе хранения. После удаления вы можете снова создать mapping к этому репозиторию, указав тот же путь к удаленной системе хранения.

Синтаксис

```
DROP REPOSITORY <repository_name>
```

Параметры

Параметр	Описание
repository_name	Имя репозитория для удаления.

Пример

Пример 1: удаление репозитория с именем `oss_repo`.

```
DROP REPOSITORY `oss_repo`;
```

RECOVER

Восстанавливает базу данных, таблицу или partition, которые были удалены с помощью команды DROP. Удаленная база данных, таблица или partition могут быть восстановлены в течение периода, указанного параметром FE `catalog_trash_expire_second` (по умолчанию 1 день).

Данные, удаленные с помощью TRUNCATE TABLE, не могут быть восстановлены.

Синтаксис

1. Восстановление базы данных.

```
RECOVER DATABASE <db_name>
```

2. Восстановление таблицы.

```
RECOVER TABLE [<db_name>.]<table_name>
```

3. Восстановление partition.

```
RECOVER PARTITION <partition_name> FROM [<db_name>.]<table_name>
```

Примечание:

1. Эта команда может восстановить только метаданные, удаленные некоторое время назад (по умолчанию 1 день). Вы можете изменить продолжительность, настроив параметр FE `catalog_trash_expire_second`.
2. Если метаданные удалены, а затем созданы идентичные метаданные, предыдущие не будут восстановлены.

Примеры

1. Восстановление базы данных `example_db`.

```
RECOVER DATABASE example_db;
```

2. Восстановление таблицы `example_tbl`.

```
RECOVER TABLE example_db.example_tbl;
```

3. Восстановление partition `p1` в таблице `example_tbl`.

```
RECOVER PARTITION p1 FROM example_tbl;
```

RESTORE

Selena поддерживает резервное копирование и восстановление следующих объектов:

- Внутренние базы данных, таблицы (всех типов и стратегий partitioning) и partition
- Метаданные внешних catalog (поддерживается начиная с v1.5.2)
- Синхронные материализованные представления и асинхронные материализованные представления
- Логические представления (поддерживается начиная с v1.5.2)
- Пользовательские функции (поддерживается начиная с v1.5.2)

ПОДСКАЗКА

Обзор резервного копирования и восстановления см. в руководстве по резервному копированию и восстановлению.

RESTORE - это асинхронная операция. Вы можете проверить статус задания RESTORE с помощью SHOW RESTORE или отменить задание RESTORE с помощью CANCEL RESTORE.

> **ВНИМАНИЕ**
>
> - Shared-data cluster Selena не поддерживают BACKUP и RESTORE данных.
> - В каждой базе данных одновременно может выполняться только одно задание BACKUP или RESTORE. В противном случае Selena возвращает ошибку.

Требования к привилегиям

В версиях ранее v1.5.2 пользователи с привилегией `admin_priv` могут выполнять эту операцию. В версиях v1.5.2 и более поздних, чтобы выполнить резервное копирование

определенного объекта, пользователи должны иметь привилегию REPOSITORY на уровне System и привилегию EXPORT для соответствующей таблицы или всех таблиц в соответствующей базе данных. Например:

- Предоставление роли разрешения на экспорт данных из указанной таблицы.

```
GRANT REPOSITORY ON SYSTEM TO ROLE backup_tbl;
GRANT EXPORT ON TABLE <table_name> TO ROLE backup_tbl;
```

- Предоставление роли разрешения на экспорт данных из всех таблиц в указанной базе данных.

```
GRANT REPOSITORY ON SYSTEM TO ROLE backup_db;
GRANT EXPORT ON ALL TABLES IN DATABASE <database_name> TO ROLE
backup_db;
```

- Предоставление роли разрешения на экспорт данных из всех таблиц во всех базах данных.

```
GRANT REPOSITORY ON SYSTEM TO ROLE backup;
GRANT EXPORT ON ALL TABLES IN ALL DATABASES TO ROLE backup;
```

Синтаксис, совместимый с более ранними версиями

```
RESTORE SNAPSHOT <db_name>.<snapshot_name>
FROM <repository_name>
[ ON ( <table_name> [ PARTITION ( <partition_name> [, ...] ) ]
    [ AS <table_alias>] [, ...] ) ]
PROPERTIES ("key"="value", ...)
```

Параметры

Параметр	Описание
db_name	Имя базы данных, в которую восстанавливаются данные.
snapshot_na	Имя снимка данных.

Параметр	Описание
me	
repository_name	Имя репозитория.
ON	Имя таблиц для восстановления. Если этот параметр не указан, восстанавливается вся база данных.
PARTITION	Имя partition для восстановления. Если этот параметр не указан, восстанавливается вся таблица. Вы можете просмотреть имя partition с помощью SHOW PARTITIONS.
PROPERTIES	Свойства операции RESTORE. Допустимые ключи: • `backup_timestamp`: Временная метка резервной копии. **Обязательно**. Вы можете просмотреть временные метки резервных копий с помощью SHOW SNAPSHOT. • `replication_num`: Указывает количество replica для восстановления. По умолчанию: `3`. • `meta_version`: Этот параметр используется только как временное решение для восстановления данных, резервная копия которых была создана более ранней версией Selena. Последняя версия резервных данных уже содержит `meta version`, и вам не нужно его указывать. • `timeout`: Тайм-аут задачи. Единица: секунда. По умолчанию: `86400`.

Синтаксис, поддерживаемый начиная с v1.5.2

```
-- Восстановление метаданных внешних catalog.
RESTORE SNAPSHOT [<db_name>.]<snapshot_name> FROM <repository_name>
{ ALL EXTERNAL CATALOGS | EXTERNAL CATALOG[S] <catalog_name>[, EXTERNAL
CATALOG[S] <catalog_name> ...] [ AS <alias> ] }
[ DATABASE <db_name_in_snapshot> [AS <target_db>] ]
[ PROPERTIES ("key"="value" [, ...] ) ]

-- Восстановление баз данных, таблиц, partition, материализованных
представлений, логических представлений или UDF.
RESTORE SNAPSHOT [<db_name>.]<snapshot_name> FROM <repository_name>
[ DATABASE <db_name_in_snapshot> [AS <target_db>] ]
[ ON ( restore_object [, ...] ) ]
[ PROPERTIES ("key"="value" [, ...] ) ]

restore_object ::= [
    { ALL TABLE[S]             | TABLE[S] <table_name>[, TABLE[S]
<table_name> ...] [AS <alias>] } |
    { ALL MATERIALIZED VIEW[S] | MATERIALIZED VIEW[S] <mv_name>[,
MATERIALIZED VIEW[S] <mv_name> ...] [AS <alias>] } |
    { ALL VIEW[S]              | VIEW[S] <view_name>[, VIEW[S] <view_name>
...] [AS <alias>] } |
    { ALL FUNCTION[S]          | FUNCTION[S] <udf_name>[, FUNCTION[S]
<udf_name> ...] [AS <alias>] } |
     <table_name> PARTITION (<partition_name>[, ...]) [AS <alias>] ]
```

Параметры

Параметр	Описание
db_name.	Имя базы данных, в которую восстанавливается объект(ы) или снимок в целевом cluster. Если база данных не существует, система создаст ее. Вы можете указать либо `AS <target_db>`, либо `<db_name>.`.
snapshot_name	Имя снимка данных.
repository_name	Имя репозитория.
ALL EXTERNAL CATALOGS	Восстанавливает метаданные всех внешних catalog.
catalog_name	Имя внешних catalog, которые необходимо восстановить.

Параметр	Описание
DATABASE db_name_in_snapshot	Имя базы данных, к которой принадлежал объект(ы) или снимок при резервном копировании в исходном cluster.
AS target_db	Имя базы данных, в которую восстанавливается объект(ы) или снимок в целевом cluster. Если база данных не существует, система создаст ее. Вы можете указать либо `AS <target_db>`, либо `<db_name>.`.
ON	Объект для восстановления. Если этот параметр не указан, восстанавливается вся база данных.
table_name	Имя таблицы(таблиц) для восстановления.
mv_name	Имя материализованного представления(представлений) для восстановления.
view_name	Имя логического представления(представлений) для восстановления.
udf_name	Имя UDF для восстановления.
PARTITION	Имя partition для восстановления. Если этот параметр не указан, восстанавливается вся таблица.
AS alias	Устанавливает новое имя для восстанавливаемого объекта в целевом cluster.
PROPERTIES	Свойства операции RESTORE. Допустимые ключи: • `backup_timestamp`: Временная метка резервной копии. **Обязательно**. Вы можете просмотреть временные метки резервных копий с помощью SHOW SNAPSHOT. • `replication_num`: Указывает количество replica для восстановления. По умолчанию: `3`. • `meta_version`: Этот параметр используется только как временное решение для восстановления данных, резервная копия которых была создана более ранней версией Selena. Последняя версия резервных данных уже содержит `meta version`, и вам не нужно его указывать. • `timeout`: Тайм-аут задачи. Единица: секунда. По умолчанию: `86400`.

Примеры

Примеры с синтаксисом, совместимым с более ранними версиями

Пример 1: Восстанавливает таблицу `backup_tbl` из снимка `snapshot_label1` из репозитория `example_repo` в базу данных `example_db`, временная метка резервной копии - `2018-05-04-16-45-08`. Восстанавливает одну replica.

```
RESTORE SNAPSHOT example_db.snapshot_label1
FROM example_repo
ON ( backup_tbl )
PROPERTIES
(
    "backup_timestamp"="2018-05-04-16-45-08",
    "replication_num" = "1"
);
```

Пример 2: Восстанавливает partition `p1` и `p2` таблицы `backup_tbl` из `snapshot_label2` и таблицу `backup_tbl2` из `example_repo` в базу данных `example_db`, переименовывая `backup_tbl2` в `new_tbl`. Временная метка резервной копии - `2018-05-04-17-11-01`. По умолчанию восстанавливает три replica.

```
RESTORE SNAPSHOT example_db.snapshot_label2
FROM example_repo
ON(
    backup_tbl PARTITION (p1, p2),
    backup_tbl2 AS new_tbl
)
PROPERTIES
(
    "backup_timestamp"="2018-05-04-17-11-01"
);
```

Примеры с синтаксисом, поддерживаемым начиная с v1.5.2

Пример 1: Восстановление базы данных.

```
-- Восстанавливает базу данных с ее исходным именем.
RESTORE SNAPSHOT sr_hub_backup
FROM test_repo
DATABASE sr_hub
PROPERTIES("backup_timestamp" = "2024-12-09-10-25-58-842");

-- Восстанавливает базу данных с новым именем.
RESTORE SNAPSHOT sr_hub_backup
FROM test_repo
DATABASE sr_hub AS sr_hub_new
PROPERTIES("backup_timestamp" = "2024-12-09-10-25-58-842");
```

Пример 2: Восстановление таблицы(таблиц) в базе данных.

```
-- Восстанавливает одну таблицу с ее исходным именем.
RESTORE SNAPSHOT sr_member_backup
FROM test_repo
DATABASE sr_hub
ON (TABLE sr_member)
PROPERTIES ("backup_timestamp" = "2024-12-09-10-52-10-940");

-- Восстанавливает одну таблицу с новым именем.
RESTORE SNAPSHOT sr_member_backup
FROM test_repo
DATABASE sr_hub  AS sr_hub_new
ON (TABLE sr_member AS sr_member_new)
PROPERTIES ("backup_timestamp" = "2024-12-09-10-52-10-940");

-- Восстанавливает несколько таблиц.
RESTORE SNAPSHOT sr_core_backup
FROM test_repo
DATABASE sr_hub
ON (TABLE sr_member, TABLE sr_pmc)
PROPERTIES ("backup_timestamp" = "2024-12-09-10-52-10-940");

-- Восстанавливает все таблицы.
RESTORE SNAPSHOT sr_all_backup
FROM test_repo
DATABASE sr_hub
ON (ALL TABLES);

-- Восстанавливает одну таблицу из всех таблиц.
RESTORE SNAPSHOT sr_all_backup
FROM test_repo
DATABASE sr_hub
ON (TABLE sr_member)
PROPERTIES ("backup_timestamp" = "2024-12-09-10-52-10-940");
```

Пример 3: Восстановление partition в таблице.

```
-- Восстанавливает одну partition.
RESTORE SNAPSHOT sr_par_backup
FROM test_repo
DATABASE sr_hub
ON (TABLE sr_member PARTITION (p1))
PROPERTIES ("backup_timestamp" = "2024-12-09-10-52-10-940");

-- Восстанавливает несколько partition.
RESTORE SNAPSHOT sr_par_backup
FROM test_repo
DATABASE sr_hub
ON (TABLE sr_member PARTITION (p1,p2))
PROPERTIES ("backup_timestamp" = "2024-12-09-10-52-10-940");
```

Пример 4: Восстановление материализованного представления(представлений) в базе данных.

```
-- Восстанавливает одно материализованное представление.
RESTORE SNAPSHOT sr_mv1_backup
FROM test_repo
DATABASE sr_hub
ON (MATERIALIZED VIEW sr_mv1)
PROPERTIES ("backup_timestamp" = "2024-12-09-10-52-10-940");

-- Восстанавливает несколько материализованных представлений.
RESTORE SNAPSHOT sr_mv2_backup
FROM test_repo
DATABASE sr_hub
ON (MATERIALIZED VIEW sr_mv1, MATERIALIZED VIEW sr_mv2)
PROPERTIES ("backup_timestamp" = "2024-12-09-10-52-10-940");

-- Восстанавливает все материализованные представления.
RESTORE SNAPSHOT sr_mv3_backup
FROM test_repo
DATABASE sr_hub
ON (ALL MATERIALIZED VIEWS)
PROPERTIES ("backup_timestamp" = "2024-12-09-10-52-10-940");

-- Восстанавливает одно из всех материализованных представлений.
RESTORE SNAPSHOT sr_mv3_backup
FROM test_repo
DATABASE sr_hub
ON (MATERIALIZED VIEW sr_mv1)
PROPERTIES ("backup_timestamp" = "2024-12-09-10-52-10-940");
```

Пример 5: Восстановление логического представления(представлений) в базе данных.

```
-- Восстанавливает одно логическое представление.
RESTORE SNAPSHOT sr_view1_backup
FROM test_repo
DATABASE sr_hub
ON (VIEW sr_view1)
PROPERTIES ("backup_timestamp" = "2024-12-09-10-52-10-940");

-- Восстанавливает несколько логических представлений.
RESTORE SNAPSHOT sr_view2_backup
FROM test_repo
DATABASE sr_hub
ON (VIEW sr_view1, VIEW sr_view2)
PROPERTIES ("backup_timestamp" = "2024-12-09-10-52-10-940");

-- Восстанавливает все логические представления.
RESTORE SNAPSHOT sr_view3_backup
FROM test_repo
DATABASE sr_hub
ON (ALL VIEWS)
PROPERTIES ("backup_timestamp" = "2024-12-09-10-52-10-940");

-- Восстанавливает одно из всех логических представлений.
RESTORE SNAPSHOT sr_view3_backup
FROM test_repo
DATABASE sr_hub
ON (VIEW sr_view1)
PROPERTIES ("backup_timestamp" = "2024-12-09-10-52-10-940");
```

Пример 6: Восстановление UDF в базе данных.

```
-- Восстанавливает одну UDF.
RESTORE SNAPSHOT sr_udf1_backup
FROM test_repo
DATABASE sr_hub
ON (FUNCTION sr_udf1)
PROPERTIES ("backup_timestamp" = "2024-12-09-10-52-10-940");

-- Восстанавливает несколько UDF.
RESTORE SNAPSHOT sr_udf2_backup
FROM test_repo
DATABASE sr_hub
ON (FUNCTION sr_udf1, FUNCTION sr_udf2)
PROPERTIES ("backup_timestamp" = "2024-12-09-10-52-10-940");

-- Восстанавливает все UDF.
RESTORE SNAPSHOT sr_udf3_backup
FROM test_repo
DATABASE sr_hub
ON (ALL FUNCTIONS)
PROPERTIES ("backup_timestamp" = "2024-12-09-10-52-10-940");

-- Восстанавливает одну из всех UDF.
RESTORE SNAPSHOT sr_udf3_backup
FROM test_repo
DATABASE sr_hub
ON (FUNCTION sr_udf1)
PROPERTIES ("backup_timestamp" = "2024-12-09-10-52-10-940");
```

Примечания по использованию

- Выполнение операций резервного копирования и восстановления на глобальном уровне, уровне базы данных, таблицы и partition требует различных привилегий.
- В каждой базе данных одновременно может выполняться только одно задание BACKUP или RESTORE. В противном случае Selena возвращает ошибку.
- Поскольку задания BACKUP и RESTORE занимают много ресурсов вашего cluster Selena, вы можете выполнять резервное копирование и восстановление данных, когда ваш cluster Selena не сильно загружен.
- Selena не поддерживает указание алгоритмов сжатия данных для резервного копирования данных.
- Поскольку данные копируются как снимки, данные, загруженные при создании снимка, не включаются в снимок. Поэтому, если вы загружаете данные в старый cluster после создания снимка и до завершения задания RESTORE, вам также

необходимо загрузить данные в cluster, в который восстанавливаются данные. Рекомендуется загружать данные в оба cluster параллельно в течение некоторого времени после завершения миграции данных, а затем переносить приложение в новый cluster после проверки правильности данных и сервисов.

- До завершения задания RESTORE вы не можете выполнять операции с таблицей, которую необходимо восстановить.
- Таблицы Primary Key не могут быть восстановлены в cluster Selena версии ранее v1.5.2.
- Вам не нужно создавать таблицу для восстановления в новом cluster перед восстановлением. Задание RESTORE автоматически создает ее.
- Если существует таблица с дублирующимся именем с таблицей, которую необходимо восстановить, Selena сначала проверяет, совпадает ли схема существующей таблицы со схемой таблицы, которую необходимо восстановить. Если схемы совпадают, Selena перезаписывает существующую таблицу данными из снимка. Если схема не совпадает, задание RESTORE завершается неудачно. Вы можете либо переименовать таблицу для восстановления с помощью ключевого слова `AS`, либо удалить существующую таблицу перед восстановлением данных.
- Если задание RESTORE перезаписывает существующую базу данных, таблицу или partition, перезаписанные данные не могут быть восстановлены после того, как задание переходит в фазу COMMIT. Если задание RESTORE завершается неудачно или отменяется в этот момент, данные могут быть повреждены и недоступны. В этом случае вы можете только снова выполнить операцию RESTORE и дождаться завершения задания. Поэтому мы рекомендуем не восстанавливать данные с перезаписью, если вы не уверены, что текущие данные больше не используются. Операция перезаписи сначала проверяет согласованность метаданных между снимком и существующей базой данных, таблицей или partition. Если обнаружено несоответствие, операция RESTORE не может быть выполнена.
- В настоящее время Selena не поддерживает резервное копирование и восстановление конфигурационных данных, связанных с учетными записями пользователей, привилегиями и группами ресурсов.
- В настоящее время Selena не поддерживает резервное копирование и восстановление отношений Colocate Join между таблицами.

SHOW BACKUP

Просматривает последнюю задачу BACKUP в указанной базе данных.

> **ПРИМЕЧАНИЕ**
>
> В Selena сохраняется информация только о последней задаче BACKUP.

Синтаксис

```
SHOW BACKUP [FROM <db_name>]
```

Параметры

Параметр	Описание
db_name	Имя базы данных, которой принадлежит задача BACKUP.

Возвращаемые значения

Возвращаемое значение	Описание
JobId	Уникальный идентификатор задачи.
SnapshotName	Имя snapshot данных.
DbName	Имя базы данных, которой принадлежит задача BACKUP.
State	Текущее состояние задачи BACKUP: • PENDING: Начальное состояние после отправки задачи. • SNAPSHOTING: Создание snapshot. • UPLOAD_SNAPSHOT: Snapshot завершен, готов к загрузке. • UPLOADING: Загрузка snapshot. • SAVE_META: Создание локальных файлов метаданных. • UPLOAD_INFO: Загрузка файлов метаданных и информации о задаче BACKUP. • FINISHED: Задача BACKUP завершена. • CANCELLED: Задача BACKUP не выполнена или отменена.
BackupObjs	Объекты резервного копирования.
CreateTime	Время отправки задачи.
SnapshotFinishedTime	Время завершения snapshot.
UploadFinishedTime	Время завершения загрузки snapshot.
FinishedTime	Время завершения задачи.
UnfinishedTasks	Идентификаторы незавершенных подзадач на этапах SNAPSHOTING и UPLOADING.
Progress	Прогресс выполнения задач загрузки snapshot.
TaskErrMsg	Сообщения об ошибках.

Возвращаемое значение	Описание
Status	Информация о статусе.
Timeout	Таймаут задачи. Единица измерения: секунда.

Примеры

Пример 1: Просмотр последней задачи BACKUP в базе данных `example_db`.

```
SHOW BACKUP FROM example_db;
```

SHOW REPOSITORIES

Просматривает репозитории, созданные в Selena.

Синтаксис

```
SHOW REPOSITORIES
```

Возвращаемые значения

Возвращаемое значение	Описание
RepoId	Идентификатор репозитория.
RepoName	Имя репозитория.
CreateTime	Время создания репозитория.
IsReadOnly	Является ли репозиторий доступным только для чтения.
Location	Расположение репозитория в удаленной системе хранения.
Broker	Broker, используемый для создания репозитория.
ErrMsg	Сообщение об ошибке при подключении к репозиторию.

Примеры

Пример 1: Просмотр репозиториев, созданных в Selena.

```
SHOW REPOSITORIES;
```

SHOW RESTORE

Просматривает последнюю задачу RESTORE в указанной базе данных.

> **ПРИМЕЧАНИЕ**
>
> В Selena сохраняется информация только о последней задаче RESTORE.

Синтаксис

```
SHOW RESTORE [FROM <db_name>]
```

Параметры

Параметр	Описание
db_name	Имя базы данных, которой принадлежит задача RESTORE.

Возвращаемые значения

Возвращаемое значение	Описание
JobId	Уникальный идентификатор задачи.
Label	Имя snapshot данных.
Timestamp	Временная метка резервной копии.
DbName	Имя базы данных, которой принадлежит задача RESTORE.
State	Текущее состояние задачи RESTORE: • PENDING: Начальное состояние после отправки задачи. • SNAPSHOTING: Выполнение локального snapshot. • DOWNLOAD: Отправка задачи загрузки snapshot. • DOWNLOADING: Загрузка snapshot. • COMMIT: Подготовка к фиксации загруженного snapshot. • COMMITTING: Фиксация загруженного snapshot. • FINISHED: Задача RESTORE завершена. • CANCELLED: Задача RESTORE не выполнена или отменена.
AllowLoad	Разрешена ли загрузка данных во время выполнения задачи RESTORE.
ReplicationNum	Количество реплик для восстановления.
RestoreObjs	Восстанавливаемые объекты (таблицы и партиции).
CreateTime	Время отправки задачи.
MetaPreparedTime	Время завершения подготовки локальных метаданных.

Возвращаемое значение	Описание
SnapshotFinishedTime	Время завершения snapshot.
DownloadFinishedTime	Время завершения загрузки snapshot.
FinishedTime	Время завершения задачи.
UnfinishedTasks	Идентификаторы незавершенных подзадач на этапах SNAPSHOTTING, DOWNLOADING и COMMITTING.
Progress	Прогресс выполнения задач загрузки snapshot.
TaskErrMsg	Сообщения об ошибках.
Status	Информация о статусе.
Timeout	Таймаут задачи. Единица измерения: секунда.

Примеры

Пример 1: Просмотр последней задачи RESTORE в базе данных `example_db`.

```
SHOW RESTORE FROM example_db;
```

SHOW SNAPSHOT

Просматривает снимки данных в указанном репозитории. Подробнее см. резервное копирование и восстановление данных.

Синтаксис

```
SHOW SNAPSHOT ON <repo_name>
[WHERE SNAPSHOT = <snapshot_name> [AND TIMESTAMP = <backup_timestamp>]]
```

Параметры

Параметр	Описание
repo_name	Имя репозитория, которому принадлежит снимок.
snapshot_nam	Имя снимка.
backup_timestamp	Временная метка резервной копии снимка.

Возвращаемые значения

Возвращаемое значение	Описание
Snapshot	Имя снимка.
Timestamp	Временная метка резервной копии снимка.
Status	Отображает `OK`, если снимок в порядке. Отображает сообщение об ошибке, если снимок неисправен.
Database	Имя базы данных, которой принадлежит снимок.
Details	Директория и структура снимка в формате JSON.

Пример

Пример 1: Просмотр снимков в репозитории `example_repo`.

```
SHOW SNAPSHOT ON example_repo;
```

Пример 2: Просмотр снимка с именем `backup1` в репозитории `example_repo`.

```
SHOW SNAPSHOT ON example_repo
WHERE SNAPSHOT = "backup1";
```

Пример 3: Просмотр снимка с именем `backup1` и временной меткой резервной копии `2018-05-05-15-34-26` в репозитории `example_repo`.

```
SHOW SNAPSHOT ON example_repo
WHERE SNAPSHOT = "backup1" AND TIMESTAMP = "2018-05-05-15-34-26";
```

3.13 Translate Trino SQL

> *Beta* **BETA FEATURE**
>
> Рекомендации по использованию бета-функций

Начиная с версии v1.5.2, Selena поддерживает перевод SQL-запросов Trino в SQL-запросы Selena.

Синтаксис

```
TRANSLATE TRINO <SELECT_statement>
```

Параметры

`SELECT_statement`: SQL-запрос Trino, который вы хотите перевести.

Возвращаемое значение

Возвращает SQL-запрос Selena.

Примеры

```
mysql> TRANSLATE TRINO SELECT id, name, category FROM products WHERE name =
'Dell XPS 13'  AND category = "Electronics"  AND `price` > 500;
+-----------------------------------------------------------------------
-------------------------------------------------------------+
| Translated SQL
|
+-----------------------------------------------------------------------
-------------------------------------------------------------+
| SELECT `id`, `name`, `category`
FROM `products`
WHERE ((`name` = 'Dell XPS 13') AND (`category` = 'Electronics')) AND
(`price` > 500) |
+-----------------------------------------------------------------------
-------------------------------------------------------------+
1 row in set (0.30 sec)
```

3.14 Генерируемые столбцы

Начиная с версии 3.1, Selena поддерживает генерируемые столбцы. Генерируемые столбцы можно использовать для ускорения запросов со сложными выражениями. Эта функция поддерживает предварительное вычисление и сохранение результатов выражений и перезапись запросов, что значительно ускоряет запросы с теми же сложными выражениями.

Вы можете определить один или несколько генерируемых столбцов для хранения результатов выражений при создании таблицы. Таким образом, при выполнении запросов, содержащих выражение, результаты которого хранятся в определённом вами

генерируемом столбце, CBO перезаписывает запрос для чтения данных непосредственно из генерируемого столбца. Также вы можете напрямую запрашивать данные из генерируемого столбца.

Также рекомендуется **оценить влияние генерируемых столбцов на производительность загрузки, поскольку вычисление выражений занимает некоторое время**. Кроме того, рекомендуется **создавать генерируемые столбцы при создании таблицы вместо добавления или изменения их после создания таблицы**. Это связано с тем, что добавление или изменение генерируемых столбцов после создания таблицы требует много времени и ресурсов.

Однако обратите внимание, что при загрузке данных в таблицы с генерируемым столбцом время и накладные расходы могут увеличиться, поскольку Selena необходимо выполнить дополнительные вычисления для расчёта результатов на основе выражения и записать результаты в генерируемый столбец.

Начиная с версии 3.5.0, cluster Selena с разделяемыми данными поддерживают генерируемые столбцы.

Базовые операции

Создание генерируемых столбцов

Синтаксис

```
<col_name> <data_type> [NULL] AS <expr> [COMMENT 'string']
```

Создание генерируемых столбцов при создании таблицы

Создайте таблицу с именем `test_tbl1` с пятью столбцами, из которых столбцы `newcol1` и `newcol2` являются генерируемыми столбцами, значения которых вычисляются с использованием указанных выражений и ссылаются на значения обычных столбцов `data_array` и `data_json` соответственно.

```
CREATE TABLE test_tbl1
(
    id INT NOT NULL,
    data_array ARRAY<int> NOT NULL,
    data_json JSON NOT NULL,
    newcol1 DOUBLE AS array_avg(data_array),
    newcol2 String AS json_string(json_query(data_json, "$.a"))
)
PRIMARY KEY (id)
DISTRIBUTED BY HASH(id);
```

ПРИМЕЧАНИЕ:

- Генерируемые столбцы должны быть определены после обычных столбцов.
- Агрегатные функции не могут использоваться в выражениях для генерируемых столбцов.
- Выражения для генерируемых столбцов не могут ссылаться на другие генерируемые столбцы или столбцы с автоинкрементом, но выражения могут ссылаться на несколько обычных столбцов.
- Тип данных генерируемого столбца должен соответствовать типу данных результата, генерируемого выражением для генерируемого столбца.
- Генерируемые столбцы не могут быть созданы для таблиц Aggregate.

Добавление генерируемых столбцов после создания таблицы

ПОДСКАЗКА

В большинстве случаев часто используемые при запросах выражения определяются после создания таблицы, поэтому генерируемые столбцы часто добавляются после создания таблицы. Для повышения производительности внутренняя логика Selena для добавления генерируемых столбцов после создания таблицы оптимизирована. Таким образом, при добавлении генерируемого столбца Selena не нужно перезаписывать все данные. Вместо этого Selena нужно только записать данные нового генерируемого столбца и связать эти данные с существующими физическими файлами данных, что значительно повышает эффективность добавления генерируемых столбцов после создания таблицы.

1. Создайте таблицу с именем `test_tbl2` с тремя обычными столбцами `id`, `data_array` и `data_json`. Вставьте строку данных в таблицу.

```
-- Создание таблицы.
CREATE TABLE test_tbl2
(
    id INT NOT NULL,
    data_array ARRAY<int> NOT NULL,
    data_json JSON NOT NULL
)
PRIMARY KEY (id)
DISTRIBUTED BY HASH(id);

-- Вставка строки данных.
INSERT INTO test_tbl2 VALUES (1, [1,2], parse_json('{"a" : 1, "b" :
2}'));

-- Запрос к таблице.
MySQL [example_db]> select * from test_tbl2;
+------+------------+------------------+
| id   | data_array | data_json        |
+------+------------+------------------+
|    1 | [1,2]      | {"a": 1, "b": 2} |
+------+------------+------------------+
1 row in set (0.04 sec)
```

2. Выполните ALTER TABLE ... ADD COLUMN ... для добавления генерируемых столбцов `newcol1` и `newcol2`, которые создаются путём вычисления выражений на основе значений обычных столбцов `data_array` и `data_json`.

```
ALTER TABLE test_tbl2
ADD COLUMN newcol1 DOUBLE AS array_avg(data_array);

ALTER TABLE test_tbl2
ADD COLUMN newcol2 String AS json_string(json_query(data_json, "$.a"));
```

ПРИМЕЧАНИЕ:

- Добавление генерируемых столбцов в таблицы Aggregate не поддерживается.
- Обычные столбцы должны быть определены перед генерируемыми столбцами. При использовании оператора ALTER TABLE ... ADD COLUMN ... для добавления обычного столбца без указания его позиции система автоматически размещает его перед генерируемыми столбцами. Кроме того, нельзя использовать AFTER для явного размещения обычного столбца после генерируемого столбца.

3. Запрос данных таблицы.

```
MySQL [example_db]> SELECT * FROM test_tbl2;
+------+------------+------------------+---------+---------+
| id   | data_array | data_json        | newcol1 | newcol2 |
+------+------------+------------------+---------+---------+
|    1 | [1,2]      | {"a": 1, "b": 2} |     1.5 | 1       |
+------+------------+------------------+---------+---------+
1 row in set (0.04 sec)
```

Результат показывает, что генерируемые столбцы `newcol1` и `newcol2` добавлены в таблицу, и Selena автоматически вычислила их значения на основе выражения.

Загрузка данных в генерируемые столбцы

Во время загрузки данных Selena автоматически вычисляет значения для генерируемых столбцов на основе выражений. Вы не можете указать значения генерируемых столбцов. В следующем примере используется оператор INSERT INTO для загрузки данных:

1. Используйте INSERT INTO для вставки записи в таблицу `test_tbl1`. Обратите внимание, что нельзя указывать значения для генерируемых столбцов в предложении `VALUES ()`.

```
INSERT INTO test_tbl1 (id, data_array, data_json)
VALUES (1, [1,2], parse_json('{"a" : 1, "b" : 2}'));
```

2. Запрос данных таблицы.

```
MySQL [example_db]> SELECT * FROM test_tbl1;
+------+------------+------------------+---------+---------+
| id   | data_array | data_json        | newcol1 | newcol2 |
+------+------------+------------------+---------+---------+
|    1 | [1,2]      | {"a": 1, "b": 2} |     1.5 | 1       |
+------+------------+------------------+---------+---------+
1 row in set (0.01 sec)
```

Результаты показывают, что Selena автоматически вычислила значения для генерируемых столбцов `newcol1` и `newcol2` на основе выражений.

ПРИМЕЧАНИЕ:

Следующая ошибка возвращается, если вы указываете значения для генерируемых столбцов во время загрузки данных:

```
MySQL [example_db]> INSERT INTO test_tbl1 (id, data_array, data_json,
newcol1, newcol2)
VALUES (2, [3,4], parse_json('{"a" : 3, "b" : 4}'), 3.5, "3");
ERROR 1064 (HY000): Getting analyzing error. Detail message:
materialized column 'newcol1' can not be specified.

MySQL [example_db]> INSERT INTO test_tbl1 VALUES (2, [3,4],
parse_json('{"a" : 3, "b" : 4}'), 3.5, "3");
ERROR 1064 (HY000): Getting analyzing error. Detail message: Column
count doesn't match value count.
```

Изменение генерируемых столбцов

ПОДСКАЗКА

При изменении генерируемого столбца Selena необходимо перезаписать все данные, что требует много времени и ресурсов. Если использование ALTER TABLE для изменения генерируемых столбцов неизбежно, рекомендуется заранее оценить затраты и время.

Вы можете изменить тип данных и выражение генерируемого столбца.

1. Создайте таблицу `test_tbl3` с пятью столбцами, из которых столбцы `newcol1` и `newcol2` являются генерируемыми столбцами, значения которых вычисляются с использованием указанных выражений и ссылаются на значения обычных столбцов `data_array` и `data_json` соответственно. Вставьте строку данных в таблицу.

```
-- Создание таблицы.
MySQL [example_db]> CREATE TABLE test_tbl3
(
    id INT NOT NULL,
    data_array ARRAY<int> NOT NULL,
    data_json JSON NOT NULL,
    -- Типы данных и выражения генерируемых столбцов указаны следующим
образом:
    newcol1 DOUBLE AS array_avg(data_array),
    newcol2 String AS json_string(json_query(data_json, "$.a"))
)
PRIMARY KEY (id)
DISTRIBUTED BY HASH(id);

-- Вставка строки данных.
INSERT INTO test_tbl3 (id, data_array, data_json)
VALUES (1, [1,2], parse_json('{"a" : 1, "b" : 2}'));

-- Запрос к таблице.
MySQL [example_db]> select * from test_tbl3;
+------+------------+------------------+---------+---------+
| id   | data_array | data_json        | newcol1 | newcol2 |
+------+------------+------------------+---------+---------+
|    1 | [1,2]      | {"a": 1, "b": 2} |     1.5 | 1       |
+------+------------+------------------+---------+---------+
1 row in set (0.01 sec)
```

2. Изменение генерируемых столбцов `newcol1` и `newcol2`:

   - Изменение типа данных генерируемого столбца `newcol1` на `ARRAY<INT>` и изменение его выражения на `data_array`.

     ```
     ALTER TABLE test_tbl3
     MODIFY COLUMN newcol1 ARRAY<INT> AS data_array;
     ```

   - Изменение выражения генерируемого столбца `newcol2` для извлечения значений поля `b` из обычного столбца `data_json`.

```
ALTER TABLE test_tbl3
MODIFY COLUMN newcol2 String AS json_string(json_query(data_json,
"$.b"));
```

3. Просмотр изменённой схемы и данных в таблице.

   - Просмотр изменённой схемы.

```
MySQL [example_db]> show create table test_tbl3\G
**** 1. row ****
    Table: test_tbl3
Create Table: CREATE TABLE test_tbl3 (
id int(11) NOT NULL COMMENT "",
data_array array<int(11)> NOT NULL COMMENT "",
data_json json NOT NULL COMMENT "",
-- После изменения типы данных и выражения генерируемых столбцов
следующие:
newcol1 array<int(11)> NULL AS example_db.test_tbl3.data_array
COMMENT "",
newcol2 varchar(65533) NULL AS
json_string(json_query(example_db.test_tbl3.data_json, '$.b'))
COMMENT ""
) ENGINE=OLAP
PRIMARY KEY(id)
DISTRIBUTED BY HASH(id)
PROPERTIES (...);
1 row in set (0.00 sec)
```

   - Запрос данных таблицы после изменения. Результат показывает, что Selena пересчитала значения генерируемых столбцов `newcol1` и `newcol2` на основе изменённых выражений.

```
MySQL [example_db]> select * from test_tbl3;
+------+------------+------------------+---------+---------+
| id   | data_array | data_json        | newcol1 | newcol2 |
+------+------------+------------------+---------+---------+
|    1 | [1,2]      | {"a": 1, "b": 2} | [1,2]   | 2       |
+------+------------+------------------+---------+---------+
1 row in set (0.01 sec)
```

Удаление генерируемого столбца

Удаление столбца `newcol1` из таблицы `test_tbl3`

```
ALTER TABLE test_tbl3 DROP COLUMN newcol1;
```

> **ПРИМЕЧАНИЕ**:
>
> Если генерируемый столбец ссылается на обычный столбец в выражении, вы не можете напрямую удалить или изменить этот обычный столбец. Вместо этого вам нужно сначала удалить генерируемый столбец, а затем удалить или изменить обычный столбец.

Перезапись запросов

Если выражение в запросе совпадает с выражением генерируемого столбца, оптимизатор автоматически перезаписывает запрос для прямого чтения значений генерируемого столбца.

1. Предположим, что вы создаёте таблицу `test_tbl4` со следующей схемой:

   ```
   CREATE TABLE test_tbl4
   (
       id INT NOT NULL,
       data_array ARRAY<int> NOT NULL,
       data_json JSON NOT NULL,
       newcol1 DOUBLE AS array_avg(data_array),
       newcol2 String AS json_string(json_query(data_json, "$.a"))
   )
   PRIMARY KEY (id) DISTRIBUTED BY HASH(id);
   ```

2. Если вы запрашиваете данные в таблице `test_tbl4` с помощью оператора `SELECT array_avg(data_array), json_string(json_query(data_json, "$.a")) FROM test_tbl4;`, запрос включает только обычные столбцы `data_array` и `data_json`. Однако выражения в запросе совпадают с выражениями генерируемых столбцов `newcol1` и `newcol2`. В этом случае план выполнения показывает, что CBO автоматически перезаписывает запрос для чтения значений генерируемых столбцов `newcol1` и `newcol2`.

```
MySQL [example_db]> EXPLAIN SELECT array_avg(data_array),
json_string(json_query(data_json, "$.a")) FROM test_tbl4;
+----------------------------------------+
| Explain String                         |
+----------------------------------------+
| PLAN FRAGMENT 0                        |
|  OUTPUT EXPRS:4: newcol1 | 5: newcol2 | -- Запрос перезаписан для
чтения данных из генерируемых столбцов newcol1 и newcol2.
|   PARTITION: RANDOM                    |
|                                        |
|   RESULT SINK                          |
|                                        |
|   0:OlapScanNode                       |
|      TABLE: test_tbl4                  |
|      PREAGGREGATION: ON                |
|      partitions=0/1                    |
|      rollup: test_tbl4                 |
|      tabletRatio=0/0                   |
|      tabletList=                       |
|      cardinality=1                     |
|      avgRowSize=2.0                    |
+----------------------------------------+
15 rows in set (0.00 sec)
```

Частичные обновления и генерируемые столбцы

Для выполнения частичных обновлений в таблице с первичным ключом необходимо указать все обычные столбцы, на которые ссылаются генерируемые столбцы, в параметре `columns`. В следующем примере используется Stream Load для выполнения частичных обновлений.

1. Создайте таблицу `test_tbl5` с пятью столбцами, из которых столбцы `newcol1` и `newcol2` являются генерируемыми столбцами, значения которых вычисляются с использованием указанных выражений и ссылаются на значения обычных столбцов `data_array` и `data_json` соответственно. Вставьте строку данных в таблицу.

```
-- Создание таблицы.
CREATE TABLE test_tbl5
(
    id INT NOT NULL,
    data_array ARRAY<int> NOT NULL,
    data_json JSON NULL,
    newcol1 DOUBLE AS array_avg(data_array),
    newcol2 String AS json_string(json_query(data_json, "$.a"))
)
PRIMARY KEY (id)
DISTRIBUTED BY HASH(id);

-- Вставка строки данных.
INSERT INTO test_tbl5 (id, data_array, data_json)
VALUES (1, [1,2], parse_json('{"a" : 1, "b" : 2}'));

-- Запрос к таблице.
MySQL [example_db]> select * from test_tbl5;
+------+------------+------------------+---------+---------+
| id   | data_array | data_json        | newcol1 | newcol2 |
+------+------------+------------------+---------+---------+
|    1 | [1,2]      | {"a": 1, "b": 2} |     1.5 | 1       |
+------+------------+------------------+---------+---------+
1 row in set (0.01 sec)
```

2. Подготовьте CSV-файл `my_data1.csv` для обновления некоторых столбцов в таблице `test_tbl5`.

```
1|[3,4]|{"a": 3, "b": 4}
2|[3,4]|{"a": 3, "b": 4}
```

3. Используйте Stream Load с файлом `my_data1.csv` для обновления некоторых столбцов таблицы `test_tbl5`. Необходимо установить `partial_update:true` и указать все обычные столбцы, на которые ссылаются генерируемые столбцы, в параметре `columns`.

```
curl --location-trusted -u <username>:<password> -H "label:1" \
    -H "column_separator:|" \
    -H "partial_update:true" \
    -H "columns:id,data_array,data_json" \
    -T my_data1.csv -XPUT \
    http://<fe_host>:
<fe_http_port>/api/example_db/test_tbl5/_stream_load
```

4. Запрос данных таблицы.

```
[example_db]> select * from test_tbl5;
+------+------------+------------------+---------+---------+
| id   | data_array | data_json        | newcol1 | newcol2 |
+------+------------+------------------+---------+---------+
|    1 | [3,4]      | {"a": 3, "b": 4} |     3.5 | 3       |
|    2 | [3,4]      | {"a": 3, "b": 4} |     3.5 | 3       |
+------+------------+------------------+---------+---------+
2 rows in set (0.01 sec)
```

Ошибка возвращается Stream Load, если вы выполняете частичные обновления без указания всех обычных столбцов, на которые ссылаются генерируемые столбцы.

1. Подготовьте CSV-файл `my_data2.csv`.

```
1|[3,4]
2|[3,4]
```

2. При выполнении частичных обновлений столбцов с помощью Stream Load с файлом `my_data2.csv`, если значения для столбца `data_json` не предоставлены в `my_data2.csv` и параметр `columns` в задаче Stream Load не включает столбец `data_json`, даже если столбец `data_json` допускает нулевые значения, Stream Load возвращает ошибку, поскольку на столбец `data_json` ссылается генерируемый столбец `newcol2`.

3.15 Ключевые слова

В этой теме описываются незарезервированные и зарезервированные ключевые слова. Приводится список зарезервированных ключевых слов в Selena.

Введение

Ключевые слова в SQL-операторах, такие как `CREATE` и `DROP`, имеют особое значение при разборе Selena. Ключевые слова делятся на незарезервированные и зарезервированные.

- **Незарезервированные ключевые слова** могут напрямую использоваться в качестве идентификаторов без специальной обработки, таких как имена таблиц и столбцов. Например, `DB` является незарезервированным ключевым словом. Вы можете создать базу данных с именем `DB`.

  ```
  CREATE DATABASE DB;
  Query OK, 0 rows affected (0.00 sec)
  ```

- **Зарезервированные ключевые слова** могут использоваться в качестве идентификаторов только после специальной обработки. Например, `LIKE` является зарезервированным ключевым словом. Если вы хотите использовать его для идентификации базы данных, заключите его в пару обратных кавычек (`).

  ```
  CREATE DATABASE `LIKE`;
  Query OK, 0 rows affected (0.01 sec)
  ```

  Если оно не заключено в обратные кавычки, возвращается ошибка:

  ```
  CREATE DATABASE LIKE;
  ERROR 1064 (HY000): Getting syntax error at line 1, column 16. Detail
  message: Unexpected input 'like', the most similar input is {a legal
  identifier}.
  ```

Зарезервированные ключевые слова

Ниже приведены зарезервированные ключевые слова Selena в алфавитном порядке. Вы должны заключить их в пару обратных кавычек (`), если хотите использовать их в качестве идентификаторов. Зарезервированные ключевые слова могут различаться в зависимости от версий Selena.

A

- ADD

- ALL
- ALTER
- ANALYZE
- AND
- ARRAY
- AS
- ASC

B

- BETWEEN
- BIGINT
- BITMAP
- BOTH
- BY

C

- CASE
- CHAR
- CHARACTER
- CHECK
- COLLATE
- COLUMN
- COMPACTION (v1.5.2 и позже)
- CONVERT
- CREATE
- CROSS
- CUBE
- CURRENT_DATE
- CURRENT_TIME
- CURRENT_TIMESTAMP
- CURRENT_USER
- CURRENT_ROLE (v1.5.2 и позже)

D

- DATABASE
- DATABASES

© ООО «Платформа Данных Селена», 2026

- DECIMAL
- DECIMALV2
- DECIMAL32
- DECIMAL64
- DECIMAL128
- DEFAULT
- DELETE
- DENSE_RANK
- DESC
- DESCRIBE
- DISTINCT
- DOUBLE
- DROP
- DUAL
- DEFERRED (v1.5.2 и позже)

E

- ELSE
- EXCEPT
- EXCLUDE
- EXISTS
- EXPLAIN

F

- FALSE
- FIRST_VALUE
- FLOAT
- FOR
- FORCE
- FROM
- FULL
- FUNCTION

G

- GRANT
- GROUP

- GROUPS
- GROUPING
- GROUPING_ID

H

- HAVING
- HLL
- HOST

I

- IF
- IGNORE
- IN
- INDEX
- INFILE
- INNER
- INSERT
- INT
- INTEGER
- INTERSECT
- INTO
- IS
- IMMEDIATE (v1.5.2 и позже)

J

- JOIN
- JSON

K

- KEY
- KEYS
- KILL

L

- LAG
- LARGEINT

- LAST_VALUE
- LATERAL
- LEAD
- LEFT
- LIKE
- LIMIT
- LOAD
- LOCALTIME
- LOCALTIMESTAMP

M

- MAXVALUE
- MINUS
- MOD

N

- NTILE
- NOT
- NULL

O

- ON
- OR
- ORDER
- OUTER
- OUTFILE
- OVER

P

- PARTITION
- PERCENTILE
- PRIMARY
- PROCEDURE

Q

- QUALIFY

© ООО «Платформа Данных Селена», 2026

R

- RANGE
- RANK
- READ
- REGEXP
- RELEASE
- RENAME
- REPLACE
- REVOKE
- RIGHT
- RLIKE
- ROW
- ROWS
- ROW_NUMBER

S

- SCHEMA
- SCHEMAS
- SELECT
- SET
- SET_VAR
- SHOW
- SMALLINT
- SYSTEM

T

- TABLE
- TERMINATED
- TEXT (v1.5.2 и позже)
- THEN
- TINYINT
- TO
- TRUE

U

- UNION

- UNIQUE
- UNSIGNED
- UPDATE
- USE
- USING

V

- VALUES
- VARCHAR

W

- WHEN
- WHERE
- WITH

3.16 Подготовленные операторы (Prepared statements)

Начиная с версии v1.5.2, Selena предоставляет подготовленные операторы для выполнения SQL-операторов несколько раз с одинаковой структурой, но разными переменными. Эта функция значительно повышает эффективность выполнения и предотвращает SQL-инъекции.

Описание

Подготовленные операторы работают следующим образом:

1. **Подготовка**: Пользователи подготавливают SQL-оператор, где переменные представлены заполнителями `?`. FE анализирует SQL-оператор и генерирует план выполнения.
2. **Выполнение**: После объявления переменных пользователи передают эти переменные в оператор и выполняют его. Пользователи могут выполнять один и тот же оператор несколько раз с разными переменными.

Преимущества

- **Экономия на разборе**: В реальных бизнес-сценариях приложение часто выполняет оператор несколько раз с одинаковой структурой, но разными переменными. При поддержке подготовленных операторов Selena нужно разобрать оператор только один

раз на этапе подготовки. Последующие выполнения того же оператора с разными переменными могут напрямую использовать предварительно сгенерированный результат разбора. Таким образом, производительность выполнения оператора значительно повышается, особенно для сложных запросов.

- **Предотвращение атак SQL-инъекций**: Отделяя оператор от переменных и передавая данные, введенные пользователем, как параметры, а не напрямую объединяя переменные в оператор, Selena может предотвратить выполнение вредоносного SQL-кода злоумышленниками.

Использование

Подготовленные операторы действительны только в текущей сессии и не могут использоваться в других сессиях. После завершения текущей сессии подготовленные операторы, созданные в этой сессии, автоматически удаляются.

Синтаксис

Выполнение подготовленного оператора состоит из следующих фаз:

- PREPARE: Подготовить оператор, где переменные представлены заполнителями `?`.
- SET: Объявить переменные в операторе.
- EXECUTE: Передать объявленные переменные в оператор и выполнить его.
- DROP PREPARE или DEALLOCATE PREPARE: Удалить подготовленный оператор.

PREPARE

Синтаксис:

```
PREPARE <stmt_name> FROM <preparable_stmt>
```

Параметры:

- `stmt_name`: имя, присвоенное подготовленному оператору, которое впоследствии используется для выполнения или удаления этого подготовленного оператора. Имя должно быть уникальным в рамках одной сессии.
- `preparable_stmt`: SQL-оператор для подготовки, где заполнителем для переменных является знак вопроса (`?`). В настоящее время поддерживается **только оператор** `SELECT`.

Пример:

Подготовить оператор `SELECT` с конкретными значениями, представленными заполнителями `?`.

```
PREPARE select_by_id_stmt FROM 'SELECT * FROM users WHERE id = ?';
```

SET

Синтаксис:

```
SET @var_name = expr [, ...];
```

Параметры:

- `var_name`: имя пользовательской переменной.
- `expr`: пользовательская переменная.

Пример: Объявить переменные.

```
SET @id1 = 1, @id2 = 2;
```

Для получения дополнительной информации см. пользовательские переменные.

EXECUTE

Синтаксис:

```
EXECUTE <stmt_name> [USING @var_name [, @var_name] ...]
```

Параметры:

- `var_name`: имя переменной, объявленной в операторе `SET`.
- `stmt_name`: имя подготовленного оператора.

Пример:

Передать переменные в оператор `SELECT` и выполнить этот оператор.

```
EXECUTE select_by_id_stmt USING @id1;
```

DROP PREPARE или DEALLOCATE PREPARE

Синтаксис:

```
{DEALLOCATE | DROP} PREPARE <stmt_name>
```

Параметры:

- `stmt_name`: Имя подготовленного оператора.

Пример:

Удалить подготовленный оператор.

```
DROP PREPARE select_by_id_stmt;
```

Примеры

Использование подготовленных операторов

Следующий пример демонстрирует, как использовать подготовленные операторы для вставки, удаления, обновления и запроса данных из таблицы Selena:

Предполагается, что следующая база данных с именем `demo` и таблица с именем `users` уже созданы:

```
CREATE DATABASE IF NOT EXISTS demo;
USE demo;
CREATE TABLE users (
  id BIGINT NOT NULL,
  country STRING,
  city STRING,
  revenue BIGINT
)
PRIMARY KEY (id)
DISTRIBUTED BY HASH(id);
```

1. Подготовить операторы для выполнения.

```
PREPARE select_all_stmt FROM 'SELECT * FROM users';
PREPARE select_by_id_stmt FROM 'SELECT * FROM users WHERE id = ?';
```

2. Объявить переменные в этих операторах.

```
SET @id1 = 1, @id2 = 2;
```

3. Использовать объявленные переменные для выполнения операторов.

```
-- Запросить все данные из таблицы.
EXECUTE select_all_stmt;

-- Запросить данные с ID 1 или 2 отдельно.
EXECUTE select_by_id_stmt USING @id1;
EXECUTE select_by_id_stmt USING @id2;
```

Использование подготовленных операторов в Java-приложении

Следующий пример демонстрирует, как Java-приложение может использовать JDBC-драйвер для вставки, удаления, обновления и запроса данных из таблицы Selena:

1. При указании URL подключения к Selena в JDBC необходимо включить подготовленные операторы на стороне сервера:

```
jdbc:mysql://<fe_ip>:<fe_query_port>/useServerPrepStmts=true
```

2. Проект Selena на GitHub предоставляет пример Java-кода, который объясняет, как вставлять, удалять, обновлять и запрашивать данные из таблицы Selena через JDBC-драйвер.

© ООО «Платформа Данных Селена», 2026

4 Коды ошибок

В этом разделе описываются распространенные коды ошибок для запросов.

Порядковый номер	Код ошибки	Сообщение об ошибке
1005	Не удалось создать таблицу. Возвращается конкретная причина.	
1007	Невозможно создать базу данных с таким же именем.	
1008	Невозможно удалить несуществующую базу данных.	
1044	Невозможно получить доступ к неавторизованной базе данных.	
1045	Имя пользователя и пароль не совпадают, поэтому невозможно получить доступ к системе.	
1046	Целевая база данных не указана.	
1047	Указана недопустимая команда.	
1049	Указана недопустимая база данных.	
1050	Таблица с таким именем уже существует.	
1051	Указана недопустимая таблица.	
1052	Указанное имя столбца неоднозначно, поэтому соответствующий столбец невозможно однозначно идентифицировать.	
1053	Для запроса Semi-Join/Anti-Join указан недопустимый столбец данных.	
1054	Указанный столбец не существует в таблице.	
1058	Количество выбранных столбцов в запросе не соответствует количеству столбцов в результате запроса.	
1060	Обнаружены дублирующиеся имена столбцов.	
1064	Нет работающих узлов BE.	
1066	В запросе обнаружен дублирующийся псевдоним таблицы.	
1094	Недопустимый ID потока.	
1095	Владелец, не являющийся владельцем потока, не может завершить поток.	
1096	В запросе не указана таблица для запроса.	
1102	Неверное имя базы данных.	
1104	Неверное имя таблицы.	
1105	Прочие ошибки.	
1110	В подзапросе указаны дублирующиеся столбцы.	
1111	Недопустимое использование агрегатной функции в предложении `WHERE`	
1113	Набор столбцов в новой таблице не может быть пустым.	
1115	Используются неподдерживаемые наборы символов.	

Порядковый номер	Код ошибки	Сообщение об ошибке
1130	Клиент использует неавторизованный IP-адрес.	
1132	Нет разрешения на изменение пароля пользователя.	
1141	У указанных записей нет привилегий для отзыва.	
1142	Выполнено неавторизованное действие.	
1166	Неверное имя столбца данных.	
1193	Системная переменная имеет недопустимое имя	
1203	Превышено ограничение на количество активных соединений.	
1211	Создание новых пользователей не разрешено.	
1227	Пользователь выполнил операцию, выходящую за пределы его полномочий.	
1228	Переменные сеанса нельзя изменить командой `SET GLOBAL`.	
1229	Глобальные переменные следует изменять командой `SET GLOBAL`.	
1230	Связанные системные переменные не имеют значений по умолчанию.	
1231	Для системной переменной установлено недопустимое значение.	
1232	Для системной переменной установлено значение неверного типа данных.	
1248	Не установлен псевдоним для встроенного представления.	
1251	Клиент не поддерживает протокол аутентификации пользователя, требуемый сервером.	
1286	Движок хранения настроен неверно.	
1298	Часовой пояс настроен неверно.	
1347	Объект не соответствует ожидаемому типу.	
1353	Указанное количество столбцов в предложении `Select` представления не равно определенному количеству столбцов.	
1364	Для столбцов, не допускающих значения NULL, не установлено значение по умолчанию.	
1372	Пароль недостаточно длинный.	
1396	Выполненная операция не удалась.	
1471	В указанную таблицу не разрешена вставка данных.	
1507	Удаление несуществующего раздела, и не указано условие для удаления только существующих разделов.	
1508	Все разделы должны быть удалены операцией удаления таблицы.	
1517	Обнаружены дублирующиеся имена разделов.	
1524	Указанный плагин не был загружен.	
1567	Неверное имя раздела.	
1621	Указанная системная переменная доступна только для чтения.	

Порядковый номер	Код ошибки	Сообщение об ошибке
1735	Указанное имя раздела не существует в таблице.	
1748	Невозможно вставить данные в таблицу, не имеющую раздела.	
1749	Указанный раздел не существует.	
5000	Указанная таблица не является таблицей OLAP.	
5001	Указанный путь хранения недопустим.	
5002	Должно отображаться имя указанного столбца.	
5003	Столбец измерения должен предшествовать столбцу индекса.	
5004	Таблица должна содержать как минимум 1 столбец измерения.	
5005	Недопустимый ID cluster.	
5006	Недопустимый план запроса.	
5007	Конфликтующие планы запросов.	
5008	Вставка данных доступна только для таблиц данных с разделами.	
5009	Предложение `PARTITION` нельзя использовать для вставки данных в таблицы без разделов.	
5010	Количество столбцов в создаваемой таблице не равно количеству столбцов в предложении `SELECT`.	
5011	Невозможно получить доступ к ссылке на таблицу.	
5012	Указанное значение не является допустимым числом.	
5013	Единица времени не поддерживается.	
5014	Статус таблицы не является нормальным.	
5015	Статус раздела не является нормальным.	
5016	В разделе существует задание импорта данных.	
5017	Указанный столбец не является столбцом измерения.	
5018	Формат значения недопустим.	
5019	Replica данных не соответствует версии.	
5021	Узел BE отключен.	
5022	Количество разделов в нераздельной таблице не равно 1	
5023	В инструкции, используемой для изменения таблицы или данных, не указано действие.	
5024	Время выполнения задания истекло.	
5025	Не удалось вставить данные.	
5026	При создании таблицы с помощью инструкции `SELECT` использовался неподдерживаемый тип данных.	
5027	Указанный параметр не был установлен.	

Порядковый номер	Код ошибки	Сообщение об ошибке
5028	Указанный cluster не найден.	
5030	У пользователя нет разрешения на доступ к cluster.	
5031	Параметр не указан или недопустим.	
5032	Количество экземпляров cluster не было указано.	
5034	Cluster с таким именем уже существует.	
5035	Количество экземпляров cluster настроено неверно.	
5036	Недостаточно узлов BE в cluster.	
5037	Перед удалением cluster необходимо удалить все базы данных.	
5038	Узел BE с указанным ID не существует в cluster.	
5040	Cluster с таким именем не существует.	
5042	Нет разрешений.	
5043	Количество экземпляров должно быть больше 0.	
5046	Исходный cluster не существует.	
5047	Целевой cluster не существует.	
5048	Исходная база данных не существует.	
5049	Целевая база данных не существует.	
5050	Cluster не выбран.	
5051	Сначала необходимо связать исходную базу данных с целевой базой данных.	
5052	Внутрикластерная ошибка: неверная информация об узле BE.	
5053	Нет задачи миграции из исходной базы данных в целевую базу данных.	
5054	Указанная база данных уже связана с целевой базой данных, или выполняется миграция данных.	
5055	Связывание баз данных или миграция данных не могут выполняться внутри одного cluster.	
5056	Невозможно удалить базу данных: она связана с другой базой данных или выполняется миграция данных.	
5056	Невозможно переименовать базу данных: она связана с другой базой данных или выполняется миграция данных.	
5056	Недостаточно узлов BE в cluster.	
5056	Указанное количество узлов BE уже существует в cluster.	
5059	В cluster есть узлы BE в автономном состоянии.	
5063	Неверное имя типа.	
5064	Общее сообщение об ошибке.	
5063	Функция Colocate была отключена администратором.	

Порядковый номер	Код ошибки	Сообщение об ошибке
5063	Таблица данных colocate с таким именем не существует.	
5063	Таблица Colocate должна быть таблицей OLAP.	
5063	Таблицы Colocate должны иметь одинаковое количество replica.	
5063	Таблицы Colocate должны иметь одинаковое количество разделенных buckets.	
5063	Таблицы Colocate должны иметь одинаковое количество столбцов разделов.	
5063	Таблицы Colocate должны иметь одинаковый тип данных разделенных столбцов.	
5064	Указанная таблица не является таблицей colocate.	
5065	Указанная операция недопустима.	
5065	Указанная единица времени недопустима. Правильные единицы: `DAY`, `WEEK` и `MONTH`.	
5066	Начальное значение динамического раздела должно быть меньше 0.	
5066	Начальное значение динамического раздела не является допустимым числом.	
5066	Конечное значение динамического раздела должно быть больше 0.	
5066	Конечное значение динамического раздела не является допустимым числом.	
5066	Конечное значение динамического раздела равно null.	
5067	Количество buckets динамического раздела должно быть больше 0.	
5067	Количество buckets динамического раздела не является допустимым числом.	
5066	Количество buckets динамического раздела пусто.	
5068	Значение для разрешения динамического раздела не является допустимым логическим значением: true или false.	
5069	Имя указанного динамического раздела имеет недопустимый префикс.	
5070	Указанная операция отключена.	
5071	Количество replica динамического раздела должно быть больше 0.	
5072	Количество replica динамического раздела не является допустимым числом.	

5 Convert_tz

> *- При написании документации для новых функций предоставляйте полные примеры данных, чтобы пользователи могли протестировать их, включая примеры CREATE TABLE, INSERT/LOAD данных. - В этой статье используется `CONVERT_TZ` в качестве примера для иллюстрации требований к написанию документации по функциям.*

Преобразует значение datetime из одного часового пояса в другой.

Эта функция поддерживается начиная с версии v1.5.2.

> *Что делает эта функция, сценарий использования, некоторые детали, различие с аналогичной функцией, поддерживаемая версия.*

5.1 Синтаксис

```
DATETIME CONVERT_TZ(DATETIME dt, VARCHAR from_tz, VARCHAR to_tz)
```

> *Синтаксис этой функции. Заключите синтаксис в блоки кода.*

5.2 Параметры

- `dt`: значение datetime для преобразования. Поддерживается DATETIME.
- `from_tz`: исходный часовой пояс. Поддерживается VARCHAR. Часовой пояс может быть представлен в двух форматах: один — Time Zone Database (например, Asia/Shanghai), другой — смещение UTC (например, +08:00).
- `to_tz`: целевой часовой пояс. Поддерживается VARCHAR. Его формат такой же, как у `from_tz`.

> - Формат: `параметр`:`описание`+`тип данных` (перечислите все поддерживаемые типы данных).
> - Описание должно включать информацию, такую как описание параметра, формат значения, диапазон значений, является ли этот параметр обязательным, форматы и единицы при использовании разных типов данных.
> - Типы данных должны быть в верхнем регистре, например, DATETIME.

5.3 Возвращаемое значение

Возвращает значение типа данных DATETIME.

> *При необходимости можно предоставить дополнительные примечания:*
>
> - *Типы возвращаемого значения, если входное значение поддерживает несколько типов данных.*
> - *Тип возвращаемой ошибки (например, NULL), если тип данных входного значения не поддерживается.*

5.4 Примечания по использованию

Для Time Zone Database обратитесь к tz database time zones (в Wikipedia).

> *Вы также можете добавить некоторые примечания или меры предосторожности при использовании этой функции. (опционально)*

5.5 Примеры

Пример 1: Преобразование даты и времени в Шанхае в Los_Angeles.

```
MySQL > select convert_tz('2019-08-01 13:21:03', 'Asia/Shanghai',
'America/Los_Angeles');
        -> 2019-07-31 22:21:03
|
```

Пример 2: Преобразование даты и времени в UTC+08:00 в Los_Angeles.

```
MySQL > select convert_tz('2019-08-01 13:21:03', '+08:00',
'America/Los_Angeles');
+-----------------------------------------------------------------------+
| convert_tz('2019-08-01 13:21:03', '+08:00', 'America/Los_Angeles') |
+-----------------------------------------------------------------------+
| 2019-07-31 22:21:03                                                   |
+-----------------------------------------------------------------------+
```

> - *Приведите распространенные примеры использования этой функции и опишите назначение каждого примера, чтобы помочь пользователям быстро понять пример.*

> - *Пожалуйста, включите полные примеры данных, чтобы пользователи могли протестировать, включая CREATE TABLE, INSERT/LOAD и запрос.*
> - *Если вам нужно описать более одного сценария в примере, добавьте комментарий для каждого сценария, чтобы помочь пользователям понять.*
> - *Если пример сложный, объясните возвращаемый результат.*

6 Шаблон SQL-инструкции

> - *В этом шаблоне используется* `*ADMIN SET REPLICA STATUS*` *в качестве примера для иллюстрации требований к написанию тем по командам SQL.*
> - *Используйте заглавные буквы для команд и ключевых слов в тексте. Например, "Инструкция SELECT используется для запроса записей, соответствующих определенным условиям.", "Вы можете использовать GROUP BY для группировки данных в этом столбце.", "Ключевое слово LIMIT указывает максимальное количество записей, которые могут быть возвращены".*
> - *Если вам нужно сослаться на параметр или значение параметра в тексте, заключите его в два обратных апострофа (``), например,* `*cachesize*`*.*
> - *Предоставьте полный пример, включая CREATE TABLE, INSERT/LOAD данных, запрос, результат запроса, описание результата.*

6.1 ADMIN SET REPLICA STATUS

> *Заголовок темы. Используйте английское название команды в качестве заголовка темы. Используйте заглавные буквы для всех букв в названии команды. Убедитесь, что используете правильное написание.*

Описание

Указывает статус replica tablet. Эта команда используется для ручной установки статуса replica tablet на `bad` или `ok`.

> *Что делает эта команда. Вы можете добавить связанные описания или примечания по использованию.*

Синтаксис

```
ADMIN SET REPLICA STATUS

PROPERTIES ("key" = "value", ...);
```

> *Синтаксис этой команды. Заключите синтаксис в блок кода. Убедитесь, что синтаксис соответствует спецификациям кодирования.*

- > *Используйте правильный перенос строк и отступы.*
- > *Не используйте китайские символы в коде, такие как китайские точки с запятой или запятые.*
- > *Используйте заглавные буквы для ключевых слов в команде SQL. Пример:*

```
SELECT ta.x, count(ta.y) AS y, sum(tb.z) AS z
FROM (
    SELECT a AS x, b AS y
    FROM t) ta
    JOIN tb
        ON ta.x = tb.x
WHERE tb.a > 10
GROUP BY ta.x
ORDER BY ta.x, z
LIMIT 10
```

Параметры

`PROPERTIES`: Каждое свойство должно быть парой ключ-значение. Поддерживаемые свойства:

- `tablet_id`: ID tablet. Этот параметр является обязательным.
- `backend_id`: ID BE tablet. Этот параметр является обязательным.
- `status`: статус replica. Этот параметр является обязательным. Допустимые значения: `bad` и `ok`. Значение `ok` указывает, что система автоматически восстанавливает replica tablet. Если статус replica установлен в `bad`, replica могут быть немедленно удалены. Будьте осторожны при выполнении этой операции. Если указанный вами tablet не существует или статус replica — `bad`, система игнорирует эти replica.

> *Описание параметров в команде.*

- > *Предпочтительное описание параметра должно включать значение параметра, формат значения, диапазон значений, является ли этот параметр обязательным, и другие примечания при необходимости.*

- > Вы можете использовать неупорядоченный список для организации описания параметров. Если описание сложное, вы можете организовать информацию в виде таблицы. Таблица может состоять из следующих столбцов: имя параметра, тип значения (опционально), пример значения (опционально), описание параметра.

Возвращаемые поля

> *Опишите возвращаемые поля. Если поле может иметь несколько значений, перечислите эти значения и в каком сценарии будет возвращено определенное значение.*

Примечания по использованию (опционально)

> Вы можете *добавить некоторые примечания или меры предосторожности при использовании этой команды.*

Примеры

Пример 1: Установка статуса replica tablet 10003 на BE 10001 в `bad`.

```
ADMIN SET REPLICA STATUS PROPERTIES("tablet_id" = "10003", "backend_id" =
"10001", "status" = "bad");
```

Пример 2: Установка статуса replica tablet 10003 на BE 10001 в `ok`.

```
ADMIN SET REPLICA STATUS PROPERTIES("tablet_id" = "10003", "backend_id" =
"10001", "status" = "ok");
```

- > *Приведите примеры использования этой команды и объясните назначение каждого примера.*

- > *Вы можете предоставить несколько примеров.*

- > *Если вам нужно описать более одного сценария в примере, добавьте комментарий для каждого сценария в фрагменте кода, чтобы помочь пользователям быстро различать их.*

7 Системные ограничения

Этот раздел описывает правила и ограничения, которые применяются при использовании Selena.

- Selena использует протокол MySQL для коммуникации. Вы можете подключиться к cluster Selena через MySQL client или JDBC. Мы рекомендуем использовать MySQL client версии 5.1 или выше. Версии ниже 5.1 не поддерживают имена пользователей длиннее 16 символов.

- Соглашения об именовании для объектов таких как catalogs, databases, tables, views, асинхронные materialized views, partitions, columns, indexes, имена пользователей, roles, repositories, resources, storage volumes и pipes:

  - Имя может состоять только из цифр (0-9), букв (a-z или A-Z) и подчеркиваний (_). **Имена пользователей могут состоять только из цифр.**
  - Имя может начинаться с буквы или подчеркивания (_).
  - Имя не может превышать 64 символа в длину, **среди которых:**
    - Имя database не может превышать 256 символов.
    - Имя table и column не могут превышать 1024 символа.
    - Имя пользователя не может превышать 128 символов.
  - Имена columns (алиасы columns), имена partitions и имена indexes **не** чувствительны к регистру. Другие имена **чувствительны к регистру**.
- Параметр конфигурации FE `enable_table_name_case_insensitive` (поддерживается начиная с v2.0.0) позволяет контролировать, будут ли имена директорий данных, databases, tables, views и асинхронных materialized views нечувствительны к регистру. В настоящее время имена tables по умолчанию чувствительны к регистру.

  - После включения этой функции все связанные имена будут храниться в нижнем регистре, и все SQL команды, содержащие эти имена, будут автоматически преобразовывать их в нижний регистр.
  - Вы можете включить эту функцию только при создании cluster. **После запуска cluster значение этой конфигурации не может быть изменено никаким способом**. Любая попытка изменить его приведет к ошибке. FE не сможет запуститься, если обнаружит, что значение этого параметра конфигурации отличается от того, что было при первом запуске cluster.

  - В настоящее время эта функция не поддерживает JDBC catalog и имена tables. Не включайте эту функцию, если вы хотите выполнить обработку без учета регистра для источников данных JDBC или ODBC.
- Соглашения об именовании для labels: Вы можете указать label задачи при загрузке данных. Имя label может состоять из цифр (0-9), букв (a-z или A-Z) и подчеркиваний (_), и не может превышать 128 символов в длину. Имена labels могут начинаться с буквы или подчеркивания (_).

- При создании таблицы, ключевой column не может иметь тип FLOAT или DOUBLE. Вы можете использовать тип DECIMAL для представления десятичных чисел.

- Максимальная длина значения VARCHAR различается в разных версиях:

  - В версиях ранее Selena 2.1 длина составляет от 1 до 65533 байт.
  - [Preview] В Selena 2.1 и более поздних версиях длина составляет от 1 до 1048576 байт. Максимальная длина значения VARCHAR = Максимальный размер строки (1048578 байт) - Префикс длины (2 байта). Префикс длины указывает количество байтов в значении.
  - Длина по умолчанию составляет 1 байт.
- Selena поддерживает только кодировку UTF-8, но не GBK.

- Selena не поддерживает изменение типов tables существующей таблицы. Например, вы не можете изменить Duplicate Key table на Primary Key table. Вы должны создать новую таблицу.

- По умолчанию запрос может быть вложенным максимум с 10,000 подзапросами, что контролируется параметром FE `expr_children_limit`.

8 Системные переменные

Selena предоставляет множество системных переменных, которые можно устанавливать и изменять в соответствии с вашими требованиями. Этот раздел описывает переменные, поддерживаемые Selena. Вы можете просмотреть настройки этих переменных, выполнив команду SHOW VARIABLES на вашем MySQL client. Вы также можете использовать команду SET для динамической установки или изменения переменных. Вы можете сделать эти переменные действующими глобально на всей системе, только в текущей сессии или только в одном запросе.

Переменные в Selena относятся к наборам переменных в MySQL, но **некоторые переменные совместимы только с протоколом MySQL client и не функционируют в базе данных MySQL**.

> **ПРИМЕЧАНИЕ**
>
> Любой пользователь имеет привилегию выполнять SHOW VARIABLES и делать переменную действующей на уровне сессии. Однако только пользователи с привилегией OPERATE уровня SYSTEM могут делать переменную действующей глобально. Глобально действующие переменные вступают в силу для всех будущих сессий (исключая текущую сессию).
>
> Если вы хотите внести изменение настройки для текущей сессии, а также применить это изменение ко всем будущим сессиям, вы можете внести изменение дважды, один раз без модификатора `GLOBAL` и один раз с ним. Например:
>
> ```
> SET query_mem_limit = 137438953472; -- Применить к текущей сессии.
> SET GLOBAL query_mem_limit = 137438953472; -- Применить ко всем будущим сессиям.
> ```

8.1 Иерархия и типы переменных

Selena поддерживает три типа (уровня) переменных: глобальные переменные, переменные сессии и хинты `SET_VAR`. Их иерархическая связь следующая:

- Глобальные переменные действуют на глобальном уровне и могут быть переопределены переменными сессии и хинтами `SET_VAR`.

- Переменные сессии действуют только на текущую сессию и могут быть переопределены хинтами `SET_VAR`.
- Хинты `SET_VAR` действуют только на текущий запрос.

8.2 Просмотр переменных

Вы можете просмотреть все или некоторые переменные, используя `SHOW VARIABLES [LIKE 'xxx']`. Пример:

```
-- Показать все переменные в системе.
SHOW VARIABLES;

-- Показать переменные, соответствующие определённому шаблону.
SHOW VARIABLES LIKE '%time_zone%';
```

8.3 Установка переменных

Установка переменных глобально или для одной сессии

Вы можете установить переменные так, чтобы они действовали **глобально** или **только на текущую сессию**. При установке глобально новое значение будет использоваться для всех будущих сессий, в то время как текущая сессия по-прежнему использует исходное значение. При установке "только для текущей сессии" переменная будет действовать только на текущую сессию.

Переменная, установленная с помощью `SET <var_name> = xxx;`, действует только для текущей сессии. Пример:

```
SET query_mem_limit = 137438953472;

SET forward_to_master = true;

SET time_zone = "Asia/Shanghai";
```

Переменная, установленная с помощью `SET GLOBAL <var_name> = xxx;`, действует глобально. Пример:

```
SET GLOBAL query_mem_limit = 137438953472;
```

Следующие переменные действуют только глобально. Они не могут действовать для одной сессии, что означает, что вы должны использовать `SET GLOBAL <var_name> = xxx;` для этих переменных. Если вы попытаетесь установить такую переменную для одной сессии (`SET <var_name> = xxx;`), будет возвращена ошибка.

- activate_all_roles_on_login
- character_set_database
- default_rowset_type
- enable_query_queue_select
- enable_query_queue_statistic
- enable_query_queue_load
- init_connect
- lower_case_table_names
- license
- language
- query_cache_size
- query_queue_fresh_resource_usage_interval_ms
- query_queue_concurrency_limit
- query_queue_mem_used_pct_limit
- query_queue_cpu_used_permille_limit
- query_queue_pending_timeout_second
- query_queue_max_queued_queries
- system_time_zone
- version_comment
- version

Кроме того, настройки переменных также поддерживают константные выражения, такие как:

```
SET query_mem_limit = 10 * 1024 * 1024 * 1024;
```

```
SET forward_to_master = concat('tr', 'u', 'e');
```

Установка переменных в одном запросе

В некоторых сценариях вам может потребоваться установить переменные специально для определённых запросов. Используя хинт `SET_VAR`, вы можете установить переменные сессии, которые будут действовать только в рамках одного оператора.

Selena поддерживает использование `SET_VAR` в следующих операторах:

- SELECT
- INSERT (начиная с v1.5.2)
- UPDATE (начиная с v1.5.2)
- DELETE (начиная с v1.5.2)

`SET_VAR` может быть размещён только после указанных выше ключевых слов и заключён в `/*+...*/`.

Пример:

```
SELECT /*+ SET_VAR(query_mem_limit = 8589934592) */ name FROM people ORDER
BY name;

SELECT /*+ SET_VAR(query_timeout = 1) */ sleep(3);

UPDATE /*+ SET_VAR(insert_timeout=100) */ tbl SET c1 = 2 WHERE c1 = 1;

DELETE /*+ SET_VAR(query_mem_limit = 8589934592) */
FROM my_table PARTITION p1
WHERE k1 = 3;

INSERT /*+ SET_VAR(insert_timeout = 10000000) */
INTO insert_wiki_edit
    SELECT * FROM FILES(
        "path" = "s3://inserttest/parquet/insert_wiki_edit_append.parquet",
        "format" = "parquet",
        "aws.s3.access_key" = "XXXXXXXXXX",
        "aws.s3.secret_key" = "YYYYYYYYYY",
        "aws.s3.region" = "us-west-2"
);
```

Вы также можете установить несколько переменных в одном операторе. Пример:

```
SELECT /*+ SET_VAR
  (
  exec_mem_limit = 515396075520,
  query_timeout=10000000,
  batch_size=4096,
  parallel_fragment_exec_instance_num=32
  )
  */ * FROM TABLE;
```

Установка переменных как свойств пользователя

Вы можете установить переменные сессии как свойства пользователя, используя ALTER USER. Эта функция поддерживается начиная с v1.5.2.

Пример:

```
-- Установить переменную сессии `query_timeout` в `600` для пользователя jack.
ALTER USER 'jack' SET PROPERTIES ('session.query_timeout' = '600');
```

8.4 Описание переменных

Переменные описаны **в алфавитном порядке**. Переменные с меткой `global` могут действовать только глобально. Другие переменные могут действовать либо глобально, либо для одной сессии.

activate_all_roles_on_login (global)

- **Описание**: Включить ли все roles (включая roles по умолчанию и предоставленные roles) для пользователя Selena, когда пользователь подключается к cluster Selena.
  - Если включено (`true`), все roles пользователя активируются при входе пользователя. Это имеет приоритет над roles, установленными SET DEFAULT ROLE.
  - Если отключено (`false`), активируются roles, установленные SET DEFAULT ROLE.
- **По умолчанию**: false
- **Представлено в**: v1.5.2

Если вы хотите активировать roles, назначенные вам в сессии, используйте команду SET ROLE.

auto_increment_increment

Используется для совместимости с MySQL client. Не имеет практического применения.

autocommit

Используется для совместимости с MySQL client. Не имеет практического применения.

chunk_size

- **Описание**: Используется для указания количества строк одного пакета, передаваемого каждым узлом во время выполнения запроса. По умолчанию 4096, т.е. каждые 4096 строк данных, сгенерированных исходным узлом, упаковываются и отправляются на узел назначения. Большее количество строк улучшит пропускную способность запроса в сценариях с большим объёмом данных, но может увеличить задержку запроса в сценариях с малым объёмом данных. Также это может увеличить затраты памяти на запрос. Мы рекомендуем устанавливать `batch_size` между 1024 и 4096.
- **По умолчанию**: 4096

big_query_profile_threshold

- **Описание**: Используется для установки порога для больших запросов. Когда переменная сессии `enable_profile` установлена в `false` и количество времени, затраченное на запрос, превышает порог, указанный переменной `big_query_profile_threshold`, для этого запроса генерируется profile.

  Примечание: Мы ввели `big_query_profile_second_threshold` для установки порога для больших запросов. В v1.5.2 и последующих релизах этот параметр был заменён на `big_query_profile_threshold` для более гибких опций конфигурации.

- **По умолчанию**: 0

- **Единица измерения**: Секунда

- **Тип данных**: String

- **Представлено в**: v1.5.2

catalog

- **Описание**: Используется для указания catalog, к которому принадлежит сессия.
- **По умолчанию**: default_catalog
- **Тип данных**: String
- **Представлено в**: v1.5.2

cbo_decimal_cast_string_strict

- **Описание**: Контролирует, как CBO преобразует данные из типа DECIMAL в тип STRING. Если эта переменная установлена в `true`, преобладает логика, встроенная в v1.5.2.x и более поздних версиях, и система выполняет строгое преобразование (а именно, система усекает сгенерированную строку и заполняет нулями на основе длины шкалы). Если эта переменная установлена в `false`, преобладает логика, встроенная в версиях ранее v1.5.2.x, и система обрабатывает все действительные цифры для генерации строки.
- **По умолчанию**: true
- **Представлено в**: v1.5.2

cbo_enable_low_cardinality_optimize

- **Описание**: Whether to enable low cardinality optimization. After this feature is enabled, the performance of querying STRING columns improves by about three times.
- **По умолчанию**: true

cbo_eq_base_type

- **Описание**: Specifies the data type used for data comparison between DECIMAL data and STRING data. The default value is `DECIMAL`, and VARCHAR is also a valid value. **This variable takes effect only for `=` and `!=` comparison.**
- **Тип данных**: String
- **Представлено в**: v1.5.2

cbo_json_v2_dict_opt

- **Описание**: Whether to enable low-cardinality dictionary optimization for Flat JSON (JSON v2) extended string subcolumns created by JSON path rewrite. When enabled, the

optimizer may build and use global dictionaries for those subcolumns to accelerate string expressions, GROUP BY, and JOIN operations.

- **По умолчанию**: true
- **Тип данных**: Boolean

cbo_json_v2_rewrite

- **Описание**: Whether to enable JSON v2 path rewrite in the optimizer. When enabled, JSON functions (such as `get_json_*`) can be rewritten to direct access of Flat JSON subcolumns, enabling predicate pushdown, column pruning, and dictionary optimization.
- **По умолчанию**: true
- **Тип данных**: Boolean

cbo_materialized_view_rewrite_related_mvs_limit

- **Описание**: Specifies the maximum number of candidate materialized views allowed during query planning.
- **По умолчанию**: 64
- **Представлено в**: v1.5.2

cbo_prune_subfield

- **Описание**: Whether to enable JSON subfield pruning. This variable must be used with the BE dynamic parameter `enable_json_flat`. Otherwise, it may degrade JSON data query performance.
- **По умолчанию**: false
- **Тип данных**: Int
- **Представлено в**: v1.5.2

custom_query_id (session)

- **Описание**: Used to bind some external identifier to a current query. Can be set using `SET SESSION custom_query_id = 'my-query-id';` before executing a query. The value is reset after query is finished. This value can be passed to `KILL QUERY 'my-query-id'`. Value can be found in audit logs as a `customQueryId` field.
- **По умолчанию**: ""
- **Тип данных**: String
- **Представлено в**: v1.5.2

enable_sync_materialized_view_rewrite

- **Описание**: Whether to enable query rewrite based on synchronous materialized views.
- **По умолчанию**: true
- **Представлено в**: v1.5.2

query_including_mv_names

- **Описание**: Specifies the name of the asynchronous materialized views to include in query execution. You can use this variable to limit the number of candidate materialized views and improve the query rewrite performance in the optimizer. This item takes effect prior to `query_excluding_mv_names`.
- **По умолчанию**: empty
- **Тип данных**: String
- **Представлено в**: v1.5.2

query_excluding_mv_names

- **Описание**: Specifies the name of the asynchronous materialized views to exclude from query execution. You can use this variable to limit the number of candidate materialized views and reduce the time of query rewrite in the optimizer. `query_including_mv_names` takes effect prior to this item.
- **По умолчанию**: empty
- **Тип данных**: String
- **Представлено в**: v1.5.2

optimizer_materialized_view_timelimit

- **Описание**: Specifies the maximum time that one materialized view rewrite rule can consume. When the threshold is reached, this rule will not be used for query rewrite.
- **По умолчанию**: 1000
- **Единица измерения**: ms
- **Представлено в**: v1.5.2

enable_materialized_view_agg_pushdown_rewrite

- **Описание**: Whether to enable aggregation pushdown for materialized view query rewrite. If it is set to `true`, aggregate functions will be pushed down to Scan Operator during query execution and rewritten by the materialized view before the Join Operator is executed. This will relieve the data expansion caused by Join and thereby improve the query performance.

For detailed information about the scenarios and limitations of this feature, see Aggregation pushdown.

- **По умолчанию**: false
- **Представлено в**: v1.5.2

enable_materialized_view_text_match_rewrite

- **Описание**: Whether to enable text-based materialized view rewrite. When this item is set to true, the optimizer will compare the query with the existing materialized views. A query will be rewritten if the abstract syntax tree of the materialized view's definition matches that of the query or its sub-query.
- **По умолчанию**: true
- **Представлено в**: v1.5.2

materialized_view_subuqery_text_match_max_count

- **Описание**: Specifies the maximum number of times that the system checks whether a query's sub-query matches the materialized views' definition.
- **По умолчанию**: 4
- **Представлено в**: v1.5.2

enable_force_rule_based_mv_rewrite

- **Описание**: Whether to enable query rewrite for queries against multiple tables in the optimizer's rule-based optimization phase. Enabling this feature will improve the robustness of the query rewrite. However, it will also increase the time consumption if the query misses the materialized view.
- **По умолчанию**: true
- **Представлено в**: v1.5.2

enable_view_based_mv_rewrite

- **Описание**: Whether to enable query rewrite for logical view-based materialized views. If this item is set to `true`, the logical view is used as a unified node to rewrite the queries against itself for better performance. If this item is set to `false`, the system transcribes the queries against logical views into queries against physical tables or materialized views and then rewrites them.
- **По умолчанию**: false
- **Представлено в**: v1.5.2

enable_materialized_view_union_rewrite

- **Описание**: Whether to enable materialized view union rewrite. If this item is set to `true`, the system seeks to compensate the predicates using UNION ALL when the predicates in the materialized view cannot satisfy the query's predicates.
- **По умолчанию**: true
- **Представлено в**: v1.5.2

enable_materialized_view_plan_cache

- **Описание**: Whether to enable materialized view plan cache, which can optimize the automatic rewrite performance of materialized views. Setting it to `true` indicates enabling it.
- **По умолчанию**: true
- **Представлено в**: v1.5.2

enable_cbo_based_mv_rewrite

- **Описание**: Whether to enable materialized view rewrite in CBO phase which can maximize the likelihood of successful query rewriting (e.g., when the join order differs between materialized views and queries), but it will increase the execution time of the optimizer phase.
- **По умолчанию**: true
- **Представлено в**: v1.5.2, v2.0.0

enable_parquet_reader_bloom_filter

- **Описание**: Whether to enable the bloom filter of Parquet file to improve performance. `true` indicates enabling the bloom filter, and `false` indicates disabling it. You can also control this behavior on system level using the BE configuration `parquet_reader_bloom_filter_enable`. Bloom filters in Parquet are maintained **at the column level within each row group**. If a Parquet file contains bloom filters for certain columns, queries can use predicates on those columns to efficiently skip row groups.
- **По умолчанию**: true
- **Представлено в**: v1.5.2

enable_plan_advisor

- **Описание**: Whether to enable Query Feedback feature for slow queries and manually marked queries.

- **По умолчанию**: true
- **Представлено в**: v1.5.2

enable_plan_analyzer

- **Описание**: Whether to enable Query Feedback feature for all queries. This variable takes effect only when `enable_plan_advisor` is set to `true`.
- **По умолчанию**: false
- **Представлено в**: v1.5.2

enable_parquet_reader_bloom_filter

- **По умолчанию**: true
- **Тип**: Boolean
- **Единица измерения**: -
- **Описание**: Whether to enable Bloom Filter optimization when reading Parquet files.
  - `true` (Default): Enable Bloom Filter optimization when reading Parquet files.
  - `false`: Disable Bloom Filter optimization when reading Parquet files.
- **Представлено в**: v1.5.2

enable_parquet_reader_page_index

- **По умолчанию**: true
- **Тип**: Boolean
- **Единица измерения**: -
- **Описание**: Whether to enable Page Index optimization when reading Parquet files.
  - `true` (Default): Enable Page Index optimization when reading Parquet files.
  - `false`: Disable Page Index optimization when reading Parquet files.
- **Представлено в**: v1.5.2

follower_query_forward_mode

- **Описание**: Specifies to which FE nodes the query statements are routed.

  - Valid values:

    - `default`: Routes the query statement to the Leader FE or Follower FEs, depending on the Follower's replay progress. If the Follower FE nodes have not completed replay progress, queries will be routed to the Leader FE node. If the

replay progress is complete, queries will be preferentially routed to the Follower FE node.
    - `leader`: Routes the query statement to the Leader FE.
    - `follower`: Routes the query statement to Follower FE.
- **По умолчанию**: default
- **Тип данных**: String
- **Представлено в**: v1.5.2

character_set_database (global)

- **Тип данных**: StringThe character set supported by Selena. Only UTF8 (`utf8`) is supported.
- **По умолчанию**: utf8
- **Тип данных**: String

connector_io_tasks_per_scan_operator

- **Описание**: The maximum number of concurrent I/O tasks that can be issued by a scan operator during external table queries. The value is an integer. Currently, Selena can adaptively adjust the number of concurrent I/O tasks when querying external tables. This feature is controlled by the variable `enable_connector_adaptive_io_tasks`, which is enabled by default.
- **По умолчанию**: 16
- **Тип данных**: Int
- **Представлено в**: v1.5.2

connector_sink_compression_codec

- **Описание**: Specifies the compression algorithm used for writing data into Hive tables or Iceberg tables, or exporting data with Files(). This parameter only takes effect in the following situations:
    - The `compression_codec` property does not exist in the Hive tables.
    - The `write.parquet.compression-codec` properties do not exist in the Iceberg tables.
    - The `compression` property is not set for `INSERT INTO FILES`.
- **Допустимые значения**: `uncompressed`, `snappy`, `lz4`, `zstd`, and `gzip`.
- **По умолчанию**: uncompressed

- **Тип данных**: String
- **Представлено в**: v1.5.2

connector_sink_target_max_file_size

- **Описание**: Specifies the maximum size of target file for writing data into Hive tables or Iceberg tables, or exporting data with Files(). The limit is not exact and is applied on a best-effort basis.
- **Единица измерения**: Bytes
- **По умолчанию**: 1073741824
- **Тип данных**: Long
- **Представлено в**: v1.5.2

count_distinct_column_buckets

- **Описание**: The number of buckets for the COUNT DISTINCT column in a group-by-count-distinct query. This variable takes effect only when `enable_distinct_column_bucketization` is set to `true`.
- **По умолчанию**: 1024
- **Представлено в**: v1.5.2

default_rowset_type (global)

Used to set the default storage format used by the storage engine of the computing node. The currently supported storage formats are `alpha` and `beta`.

default_table_compression

- **Описание**: The default compression algorithm for table storage. Supported compression algorithms are `snappy, lz4, zlib, zstd`.

  Note that if you specified the `compression` property in a CREATE TABLE statement, the compression algorithm specified by `compression` takes effect.

- **По умолчанию**: lz4_frame

- **Представлено в**: v1.5.2

disable_colocate_join

- **Описание**: Used to control whether the Colocation Join is enabled. The default value is `false`, meaning the feature is enabled. When this feature is disabled, query planning will not attempt to execute Colocation Join.
- **По умолчанию**: false

disable_streaming_preaggregations

Used to enable the streaming pre-aggregations. The default value is `false`, meaning it is enabled.

div_precision_increment

Используется для совместимости с MySQL client. Не имеет практического применения.

dynamic_overwrite

- **Описание**: Whether to enable the Dynamic Overwrite semantic for INSERT OVERWRITE with partitioned tables. Valid values:
  - `true`: Enables Dynamic Overwrite.
  - `false`: Disables Dynamic Overwrite and uses the default semantic.
- **По умолчанию**: false
- **Представлено в**: v1.5.2

enable_datacache_async_populate_mode

- **Описание**: Whether to populate the data cache in asynchronous mode. By default, the system uses the synchronous mode to populate data cache, that is, populating the cache while querying data.
- **По умолчанию**: false
- **Представлено в**: v1.5.2

enable_connector_adaptive_io_tasks

- **Описание**: Whether to adaptively adjust the number of concurrent I/O tasks when querying external tables. Default value is `true`. If this feature is not enabled, you can manually set the number of concurrent I/O tasks using the variable `connector_io_tasks_per_scan_operator`.
- **По умолчанию**: true
- **Представлено в**: v1.5.2

enable_distinct_column_bucketization

- **Описание**: Whether to enable bucketization for the COUNT DISTINCT colum in a group-by-count-distinct query. Use the `select a, count(distinct b) from t group by a;` query as an example. If the GROUP BY colum `a` is a low-cardinality column and the COUNT DISTINCT column `b` is a high-cardinality column which has severe data skew, performance bottleneck will occur. In this situation, you can split data in the COUNT DISTINCT column into multiple buckets to balance data and prevent data skew. You must use this variable with the variable `count_distinct_column_buckets`.

  You can also enable bucketization for the COUNT DISTINCT column by adding the `skew` hint to your query, for example, `select a,count(distinct [skew] b) from t group by a;`.

- **По умолчанию**: false, which means this feature is disabled.

- **Представлено в**: v1.5.2

enable_group_by_compressed_key

- **Описание**: Whether to use accurate statistical information to compress the GROUP BY Key column. Valid values: `true` and `false`.
- **По умолчанию**: true
- **Представлено в**: v2.0.0

enable_gin_filter

- **Описание**: Whether to utilize the fulltext inverted index during queries.
- **По умолчанию**: true
- **Представлено в**: v1.5.2

enable_group_execution

- **Описание**: Whether to enable Colocate Group Execution. Colocate Group Execution is an execution pattern that leverages physical data partitioning, where a fixed number of threads sequentially process their respective data ranges to enhance locality and throughput. Enabling this feature can reduce memory usage.
- **По умолчанию**: true
- **Представлено в**: v1.5.2

enable_group_level_query_queue (global)

- **Описание**: Whether to enable resource group-level query queue.
- **По умолчанию**: false, which means this feature is disabled.
- **Представлено в**: v1.5.2

enable_insert_partial_update

- **Описание**: Whether to enable Partial Update for INSERT statements on Primary Key tables. When this item is set to `true` (default), if an INSERT statement specifies only a subset of columns (fewer than the number of all non-generated columns in the table), the system performs a Partial Update to update only the specified columns while preserving existing values in other columns. When set to `false`, the system uses default values for unspecified columns instead of preserving existing values. This feature is particularly useful for updating specific columns in Primary Key tables without affecting other column values.
- **По умолчанию**: true
- **Представлено в**: v1.5.2, v2.0.0

enable_iceberg_metadata_cache

- **Описание**: Whether to cache pointers and partition names for Iceberg tables. From v1.5.2 to v1.5.2, this parameter is set to `true` by default, regardless of what metastore service is used. In v1.5.2 and later, if the Iceberg cluster uses AWS Glue as metastore, this parameter still defaults to `true`. However, if the Iceberg cluster uses other metastore service such as Hive metastore, this parameter defaults to `false`.
- **Представлено в**: v1.5.2

max_unknown_string_meta_length (global)

- **Описание**: Fallback length for string columns in query result metadata when the max length is unknown. Clients that rely on the metadata may return empty values or truncation if the reported length is smaller than actual values. Valid range is `1` to `1048576`.
- **По умолчанию**: 64
- **Data Type**: int
- **Представлено в**: v1.5.2

enable_metadata_profile

- **Описание**: Whether to enabled Profile for Iceberg Catalog metadata.
- **По умолчанию**: true

- **Представлено в**: v1.5.2

plan_mode

- **Описание**: The metadata retrieval strategy of Iceberg Catalog. For more information, see Iceberg Catalog metadata retrieval strategy. Valid values:
  - `auto`: The system will automatically select the retrieval plan.
  - `local`: Use the local cache plan.
  - `distributed`: Use the distributed plan.
- **По умолчанию**: auto
- **Представлено в**: v1.5.2

enable_iceberg_column_statistics

- **Описание**: Whether to obtain column statistics, such as `min`, `max`, `null count`, `row size`, and `ndv` (if a puffin file exists). When this item is set to `false`, only the row count information will be collected.
- **По умолчанию**: false
- **Представлено в**: v1.5.2

enable_parallel_merge

- **Описание**: Whether to enable parallel merge for sorting. When this feature is enabled, the merge phase of sorting will utilize multiple threads for merge operations.
- **По умолчанию**: true
- **Представлено в**: v1.5.2

enable_per_bucket_optimize

- **Описание**: Whether to enable bucketed computation. When this feature is enabled, stage-one aggregation can be computed in bucketed order, reducing memory usage.
- **По умолчанию**: true
- **Представлено в**: v1.5.2

metadata_collect_query_timeout

- **Описание**: The timeout duration for Iceberg Catalog metadata collection queries.
- **Единица измерения**: Second
- **По умолчанию**: 60
- **Представлено в**: v1.5.2

enable_insert_strict

- **Описание**: Whether to enable strict mode while loading data using INSERT from files(). Valid values: `true` and `false` (Default). When strict mode is enabled, the system loads only qualified rows. It filters out unqualified rows and returns details about the unqualified rows. For more information, see Strict mode. In versions earlier than v1.5.2, when `enable_insert_strict` is set to `true`, the INSERT jobs fails when there is an unqualified rows.
- **По умолчанию**: true

insert_max_filter_ratio

- **Описание**: The maximum error tolerance of INSERT from files(). It's the maximum ratio of data records that can be filtered out due to inadequate data quality. When the ratio of unqualified data records reaches this threshold, the job fails. Range: [0, 1].
- **По умолчанию**: 0
- **Представлено в**: v1.5.2

insert_timeout

- **Описание**: The timeout duration of the INSERT job. Unit: Seconds. From v1.5.2 onwards, `insert_timeout` applies to operations involved INSERT (for example, UPDATE, DELETE, CTAS, materialized view refresh, statistics collection, and PIPE), replacing `query_timeout`.
- **По умолчанию**: 14400
- **Представлено в**: v1.5.2

enable_materialized_view_for_insert

- **Описание**: Whether to allow Selena to rewrite queries in INSERT INTO SELECT statements.
- **По умолчанию**: false, which means Query Rewrite in such scenarios is disabled by default.
- **Представлено в**: v1.5.2

enable_rule_based_materialized_view_rewrite

- **Описание**: Controls whether to enable rule-based materialized view query rewrite. This variable is mainly used in single-table query rewrite. * **По умолчанию**: true
- **Тип данных**: Boolean

- **Представлено в**: v1.5.2

enable_short_circuit

- **Описание**: Whether to enable short circuiting for queries. Default: `false`. If it is set to `true`, when the query meets the criteria (to evaluate whether the query is a point query): the conditional columns in the WHERE clause include all primary key columns, and the operators in the WHERE clause are `=` or `IN`, the query takes the short circuit.
- **По умолчанию**: false
- **Представлено в**: v1.5.2

enable_spm_rewrite

- **Описание**: Whether to enable SQL Plan Manager (SPM) query rewrite. When enabled, Selena automatically rewrites queries to use bound query plans, improving query performance and stability.
- **По умолчанию**: false

enable_spill

- **Описание**: Whether to enable intermediate result spilling. Default: `false`. If it is set to `true`, Selena spills the intermediate results to disk to reduce the memory usage when processing aggregate, sort, or join operators in queries.
- **По умолчанию**: false
- **Представлено в**: v1.5.2

enable_spill_to_remote_storage

- **Описание**: Whether to enable intermediate result spilling to object storage. If it is set to `true`, Selena spills the intermediate results to the storage volume specified in `spill_storage_volume` after the capacity limit of the local disk is reached. For more information, see Spill to object storage.
- **По умолчанию**: false
- **Представлено в**: v1.5.2

enable_strict_order_by

- **Описание**: Used to check whether the column name referenced in ORDER BY is ambiguous. When this variable is set to the default value `TRUE`, an error is reported for such a query pattern: Duplicate alias is used in different expressions of the query and this alias is also a sorting field in ORDER BY, for example, `select distinct t1.* from tbl1`

`t1 order by t1.k1;`. The logic is the same as that in v1.5.2 and earlier. When this variable is set to `FALSE`, a loose deduplication mechanism is used, which processes such queries as valid SQL queries.

- **По умолчанию**: true
- **Представлено в**: v1.5.2

enable_profile

- **Описание**: Specifies whether to send the profile of a query for analysis. The default value is `false`, which means no profile is required.

  By default, a profile is sent to the FE only when a query error occurs in the BE. Profile sending causes network overhead and therefore affects high concurrency.

  If you need to analyze the profile of a query, you can set this variable to `true`. After the query is completed, the profile can be viewed on the web page of the currently connected FE (address: `fe_host:fe_http_port/query`). This page displays the profiles of the latest 100 queries with `enable_profile` turned on.

- **По умолчанию**: false

enable_query_queue_load (global)

- **Описание**: Boolean value to enable query queues for loading tasks.
- **По умолчанию**: false

enable_query_queue_select (global)

- **Описание**: Whether to enable query queues for SELECT queries.
- **По умолчанию**: false

enable_query_queue_statistic (global)

- **Описание**: Whether to enable query queues for statistics queries.
- **По умолчанию**: false

enable_query_tablet_affinity

- **Описание**: Boolean value to control whether to direct multiple queries against the same tablet to a fixed replica.

In scenarios where the table to query has a large number of tablets, this feature significantly improves query performance because the meta information and data of the tablet can be cached in memory more quickly.

However, if there are some hotspot tablets, this feature may degrade the query performance because it directs the queries to the same BE, making it unable to fully use the resources of multiple BEs in high concurrency scenarios.

- **По умолчанию**: false, which means the system selects a replica for each query.

- **Представлено в**: v1.5.2.

enable_lake_tablet_internal_parallel

- **Описание**: Whether to enable Parallel Scan for Cloud-native tables in a shared-data cluster.
- **По умолчанию**: true
- **Тип данных**: Boolean
- **Представлено в**: v1.5.2

tablet_internal_parallel_mode

- **Описание**: Internal Parallel Scan strategy of tablets. Valid Values:
  - `auto`: When the number of Tablets to be scanned on BE or CN nodes is less than the Degree of Parallelism (DOP), the system automatically determines whether Parallel Scan is needed based on the estimated size of the Tablets.
  - `force_split`: Forces the splitting of Tablets and performs Parallel Scan.
- **По умолчанию**: auto
- **Тип данных**: String
- **Представлено в**: v1.5.2

enable_scan_datacache

- **Описание**: Specifies whether to enable the Data Cache feature. After this feature is enabled, Selena caches hot data read from external storage systems into blocks, which accelerates queries and analysis. For more information, see Data Cache. In versions prior to 3.2, this variable was named as `enable_scan_block_cache`.
- **По умолчанию**: true
- **Представлено в**: v1.5.2

populate_datacache_mode

- **Описание**: Specifies the population behavior of Data Cache when reading data blocks from external storage systems. Valid values:
  - `auto` (default): the system automatically caches data selectively based on the population rule.
  - `always`: Always cache the data.
  - `never`: Never cache the data.
- **По умолчанию**: auto
- **Представлено в**: v1.5.2

enable_datacache_io_adaptor

- **Описание**: Whether to enable the Data Cache I/O Adaptor. Setting this to `true` enables the feature. When this feature is enabled, the system automatically routes some cache requests to remote storage when the disk I/O load is high, reducing disk pressure.
- **По умолчанию**: true
- **Представлено в**: v1.5.2

enable_file_metacache

- **Описание**: Whether to enable metadata cache for files in remote storage (Footer Cache). Setting this to `true` enables the feature. Footer Cache directly caches the parsed Footer object in memory. When the same file's Footer is accessed in subsequent queries, the object descriptor can be obtained directly from the cache, avoiding repetitive parsing. This feature uses the memory module of the Data Cache for data caching. Therefore, you must ensure that the BE parameter `datacache_enable` is set to `true` and configure a reasonable value for `datacache_mem_size`.
- **По умолчанию**: true
- **Представлено в**: v1.5.2

enable_file_pagecache

- **Описание**: Whether to enable Page Cache for files in remote storage. Setting this to `true` enables the feature. Page Cache stores decompressed Parquet page data in memory. When the same page is accessed in subsequent queries, the data can be obtained directly from the cache, avoiding repetitive I/O operations and decompression. This feature works together with the Data Cache and uses the same memory module. When enabled, Page

Caache can significantly improve query performance for workloads with repetitive page access patterns.

- **По умолчанию**: true
- **Представлено в**: v2.0.0

enable_datacache_sharing

- **Описание**: Whether to enable Cache Sharing. Setting this to `true` enables the feature. Cache Sharing is used to support accessing cache data from other nodes through the network, which can help to reduce performance jitter caused by cache invalidation during cluster scaling. This variable takes effect only when the FE parameter `enable_trace_historical_node` is set to `true`.
- **По умолчанию**: true
- **Представлено в**: v1.5.2

datacache_sharing_work_period

- **Описание**: The period of time that Cache Sharing takes effect. After each cluster scaling operation, only the requests within this period of time will try to access the cache data from other nodes if the Cache Sharing feature is enabled.
- **По умолчанию**: 600
- **Единица измерения**: Seconds
- **Представлено в**: v1.5.2

historical_nodes_min_update_interval

- **Описание**: The minimum interval between two updates of historical node records. If the nodes of a cluster change frequently in a short period of time (that is, less than the value set in this variable), some intermediate states will not be recorded as valid historical node snapshots. The historical nodes are the main basis for the Cache Sharing feature to choose the right cache nodes during cluster scaling.
- **По умолчанию**: 600
- **Единица измерения**: Seconds
- **Представлено в**: v1.5.2

enable_tablet_internal_parallel

- **Описание**: Whether to enable adaptive parallel scanning of tablets. After this feature is enabled, multiple threads can be used to scan one tablet by segment, increasing the scan concurrency.

- **По умолчанию**: true
- **Представлено в**: v1.5.2

enable_query_cache

- **Описание**: Specifies whether to enable the Query Cache feature. Valid values: true and false. `true` specifies to enable this feature, and `false` specifies to disable this feature. When this feature is enabled, it works only for queries that meet the conditions specified in the application scenarios of Query Cache.
- **По умолчанию**: false
- **Представлено в**: v1.5.2

enable_adaptive_sink_dop

- **Описание**: Specifies whether to enable adaptive parallelism for data loading. After this feature is enabled, the system automatically sets load parallelism for INSERT INTO and Broker Load jobs, which is equivalent to the mechanism of `pipeline_dop`. For a newly deployed v1.5.2 Selena cluster, the value is `true` by default. For a v1.5.2 cluster upgraded from v1.5.2, the value is `false`.
- **По умолчанию**: false
- **Представлено в**: v1.5.2

enable_bucket_aware_execution_on_lake

- **Описание**: Whether to enable bucket-aware execution for queries against data lakes (such as Iceberg tables). When this feature is enabled, the system optimizes query execution by leveraging bucketing information to reduce data shuffling and improve performance. This optimization is particularly effective for join operations and aggregations on bucketed tables.
- **По умолчанию**: true
- **Тип данных**: Boolean
- **Представлено в**: v2.0.0

lake_bucket_assign_mode

- **Описание**: The bucket assignment mode for queries against tables in data lakes. This variable controls how buckets are distributed among worker nodes when bucket-aware execution takes effect during query execution. Valid values:
  - `balance`: Distributes buckets evenly across worker nodes to achieve balanced workload and better performance.

  - `elastic`: Uses consistent hashing to assign buckets to worker nodes, which can provide better load distribution in elastic environments.
- **По умолчанию**: balance
- **Тип данных**: String
- **Представлено в**: v2.0.0

enable_pipeline_engine

- **Описание**: Specifies whether to enable the pipeline execution engine. `true` indicates enabled and `false` indicates the opposite. Default value: `true`.
- **По умолчанию**: true

enable_sort_aggregate

- **Описание**: Specifies whether to enable sorted streaming. `true` indicates sorted streaming is enabled to sort data in data streams.
- **По умолчанию**: false
- **Представлено в**: v1.5.2

enable_global_runtime_filter

Whether to enable global runtime filter (RF for short). RF filters data at runtime. Data filtering often occurs in the Join stage. During multi-table joins, optimizations such as predicate pushdown are used to filter data, in order to reduce the number of scanned rows for Join and the I/O in the Shuffle stage, thereby speeding up the query.

Selena offers two types of RF: Local RF and Global RF. Local RF is suitable for Broadcast Hash Join and Global RF is suitable for Shuffle Join.

Default value: `true`, which means global RF is enabled. If this feature is disabled, global RF does not take effect. Local RF can still work.

enable_multicolumn_global_runtime_filter

Whether to enable multi-column global runtime filter. Default value: `false`, which means multi-column global RF is disabled.

If a Join (other than Broadcast Join and Replicated Join) has multiple equi-join conditions:

- If this feature is disabled, only Local RF works.

- If this feature is enabled, multi-column Global RF takes effect and carries `multi-column` in the partition by clause.

enable_write_hive_external_table

- **Описание**: Whether to allow for sinking data to external tables of Hive.
- **По умолчанию**: false
- **Представлено в**: v1.5.2

enable_query_trigger_analyze

- **По умолчанию**: true
- **Тип**: Boolean
- **Описание**: Whether to enable query-trigger ANALYZE tasks on tables under external catalogs.
- **Представлено в**: v1.5.2

event_scheduler

Используется для совместимости с MySQL client. Не имеет практического применения.

enable_strict_type

- **Описание**: Whether to allow implicit conversions for all compound predicates and for all expressions in the WHERE clause.
- **По умолчанию**: false
- **Представлено в**: v1.5.2

force_streaming_aggregate

Used to control whether the aggregation node enables streaming aggregation for computing. The default value is false, meaning the feature is not enabled.

forward_to_leader

Used to specify whether some commands will be forwarded to the leader FE for execution. Alias: `forward_to_master`. The default value is `false`, meaning not forwarding to the leader FE. There are multiple FEs in a Selena cluster, one of which is the leader FE. Normally, users can connect to any FE for full-featured operations. However, some information is only available on the leader FE.

For example, if the SHOW BACKENDS command is not forwarded to the leader FE, only basic information (for example, whether the node is alive) can be viewed. Forwarding to the leader FE can get more detailed information including the node start time and last heartbeat time.

The commands affected by this variable are as follows:

- SHOW FRONTENDS: Forwarding to the leader FE allows users to view the last heartbeat message.
- SHOW BACKENDS: Forwarding to the leader FE allows users to view the boot time, last heartbeat information, and disk capacity information.
- SHOW BROKER: Forwarding to the leader FE allows users to view the boot time and last heartbeat information.
- SHOW TABLET
- ADMIN SHOW REPLICA DISTRIBUTION
- ADMIN SHOW REPLICA STATUS: Forwarding to the leader FE allows users to view the tablet information stored in the metadata of the leader FE. Normally, the tablet information should be the same in the metadata of different FEs. If an error occurs, you can use this method to compare the metadata of the current FE and the leader FE.
- Show PROC: Forwarding to the leader FE allows users to view the PROC information stored in the metadata. This is mainly used for metadata comparison.

group_concat_max_len

- **Описание**: The maximum length of string returned by the group_concat function.
- **По умолчанию**: 1024
- **Минимальное значение**: 4
- **Единица измерения**: Characters
- **Тип данных**: Long

group_execution_max_groups

- **Описание**: Maximum number of groups allowed for Group Execution. It is used to limit the granularity of splitting, preventing excessive scheduling overhead caused by an excessive number of groups.

- **По умолчанию**: 128
- **Представлено в**: v1.5.2

group_execution_min_scan_rows

- **Описание**: Minimum number of rows processed per group for Group Execution.
- **По умолчанию**: 5000000
- **Представлено в**: v1.5.2

hash_join_push_down_right_table

- **Описание**: Used to control whether the data of the left table can be filtered by using the filter condition against the right table in the Join query. If so, it can reduce the amount of data that needs to be processed during the query. **По умолчанию**: `true` indicates the operation is allowed and the system decides whether the left table can be filtered. `false` indicates the operation is disabled. The default value is `true`.

init_connect (global)

Используется для совместимости с MySQL client. Не имеет практического применения.

interactive_timeout

Используется для совместимости с MySQL client. Не имеет практического применения.

interpolate_passthrough

- **Описание**: Whether to add local-exchange-passthrough for certain operators. Currently supported operators include streaming aggregates, etc. Adding local-exchange can mitigate the impact of data skew on computation, but will slightly increase memory usage.
- **По умолчанию**: true
- **Представлено в**: v1.5.2

io_tasks_per_scan_operator

- **Описание**: The number of concurrent I/O tasks that can be issued by a scan operator. Increase this value if you want to access remote storage systems such as HDFS or S3 but the latency is high. However, a larger value causes more memory consumption.
- **По умолчанию**: 4
- **Тип данных**: Int
- **Представлено в**: v1.5.2

jit_level

- **Описание**: The level at which JIT compilation for expressions is enabled. Valid values:
  - `1`: The system adaptively enables JIT compilation for compilable expressions.
  - `-1`: JIT compilation is enabled for all compilable, non-constant expressions.
  - `0`: JIT compilation is disabled. You can disable it manually if any error is returned for this feature.
- **По умолчанию**: 1
- **Тип данных**: Int
- **Представлено в**: -

language (global)

Используется для совместимости с MySQL client. Не имеет практического применения.

license (global)

- **Описание**: Отображает информацию о лицензии Selena.
- **По умолчанию**: Proprietary

load_mem_limit

Specifies the memory limit for the import operation. The default value is 0, meaning that this variable is not used and `query_mem_limit` is used instead.

This variable is only used for the `INSERT` operation which involves both query and import. If the user does not set this variable, the memory limit for both query and import will be set as `exec_mem_limit`. Otherwise, the memory limit for query will be set as `exec_mem_limit` and the memory limit for import will be as `load_mem_limit`.

Other import methods such as `BROKER LOAD`, `STREAM LOAD` still use `exec_mem_limit` for memory limit.

log_rejected_record_num (v1.5.2 and later)

Specifies the maximum number of unqualified data rows that can be logged. Valid values: `0`, `-1`, and any non-zero positive integer. Default value: `0`.

- The value `0` specifies that data rows that are filtered out will not be logged.
- The value `-1` specifies that all data rows that are filtered out will be logged.

- A non-zero positive integer such as `n` specifies that up to `n` data rows that are filtered out can be logged on each BE.

lower_case_table_names (global)

Используется для совместимости с MySQL client. Не имеет практического применения. Table names in Selena are case-sensitive.

lower_upper_support_utf8

- **По умолчанию**: false
- **Тип**: Boolean
- **Единица измерения**: -
- **Описание**: Whether to support case conversion for UTF-8 characters in `lower` and `upper` functions. Valid values:
  - `true`: Support case conversion for UTF-8 characters.
  - `false` (Default): Not to support case conversion for UTF-8 characters.
- **Представлено в**: v1.5.2

low_cardinality_optimize_on_lake

- **По умолчанию**: true
- **Тип**: Boolean
- **Единица измерения**: -
- **Описание**: Whether to enable low cardinality optimization on data lake queries. Valid values:
  - `true` (Default): Enable low cardinality optimization on data lake queries.
  - `false`: Disable low cardinality optimization on data lake queries.
- **Представлено в**: v1.5.2

materialized_view_rewrite_mode (v1.5.2 and later)

Specifies the query rewrite mode of asynchronous materialized views. Valid values:

- `disable`: Disable automatic query rewrite of asynchronous materialized views.
- `default` (Default value): Enable automatic query rewrite of asynchronous materialized views, and allow the optimizer to decide whether a query can be rewritten using the materialized view based on the cost. If the query cannot be rewritten, it directly scans the data in the base table.

- `default_or_error`: Enable automatic query rewrite of asynchronous materialized views, and allow the optimizer to decide whether a query can be rewritten using the materialized view based on the cost. If the query cannot be rewritten, an error is returned.
- `force`: Enable automatic query rewrite of asynchronous materialized views, and the optimizer prioritizes query rewrite using the materialized view. If the query cannot be rewritten, it directly scans the data in the base table.
- `force_or_error`: Enable automatic query rewrite of asynchronous materialized views, and the optimizer prioritizes query rewrite using the materialized view. If the query cannot be rewritten, an error is returned.

max_allowed_packet

- **Описание**: Used for compatibility with the JDBC connection pool C3P0. This variable specifies the maximum size of packets that can be transmitted between the client and server.
- **По умолчанию**: 33554432 (32 MB). You can raise this value if the client reports "PacketTooBigException".
- **Единица измерения**: Byte
- **Тип данных**: Int

max_pushdown_conditions_per_column

- **Описание**: The maximum number of predicates that can be pushed down for a column.
- **По умолчанию**: -1, indicating that the value in the `be.conf` file is used. If this variable is set to a value greater than 0, the value in `be.conf` is ignored.
- **Тип данных**: Int

max_scan_key_num

- **Описание**: The maximum number of scan key segmented by each query.
- **По умолчанию**: -1, indicating that the value in the `be.conf` file is used. If this variable is set to a value greater than 0, the value in `be.conf` is ignored.

nested_mv_rewrite_max_level

- **Описание**: The maximum levels of nested materialized views that can be used for query rewrite.
- **Диапазон значений**: [1, +∞). The value of `1` indicates that only materialized views created on base tables can be used for query rewrite.
- **По умолчанию**: 3

- **Тип данных**: Int

net_buffer_length

Используется для совместимости с MySQL client. Не имеет практического применения.

net_read_timeout

Используется для совместимости с MySQL client. Не имеет практического применения.

net_write_timeout

Используется для совместимости с MySQL client. Не имеет практического применения.

new_planner_optimize_timeout

- **Описание**: The timeout duration of the query optimizer. When the optimizer times out, an error is returned and the query is stopped, which affects the query performance. You can set this variable to a larger value based on your query or contact Selena technical support for troubleshooting. A timeout often occurs when a query has too many joins.
- **По умолчанию**: 3000
- **Единица измерения**: ms

parallel_exchange_instance_num

Used to set the number of exchange nodes that an upper-level node uses to receive data from a lower-level node in the execution plan. The default value is -1, meaning the number of exchange nodes is equal to the number of execution instances of the lower-level node. When this variable is set to be greater than 0 but smaller than the number of execution instances of the lower-level node, the number of exchange nodes equals the set value.

In a distributed query execution plan, the upper-level node usually has one or more exchange nodes to receive data from the execution instances of the lower-level node on different BEs. Usually the number of exchange nodes is equal to the number of execution instances of the lower-level node.

In some aggregation query scenarios where the amount of data decreases drastically after aggregation, you can try to modify this variable to a smaller value to reduce the resource overhead. An example would be running aggregation queries using the Duplicate Key table.

parallel_fragment_exec_instance_num

Used to set the number of instances used to scan nodes on each BE. The default value is 1.

A query plan typically produces a set of scan ranges. This data is distributed across multiple BE nodes. A BE node will have one or more scan ranges, and by default, each BE node's set of scan ranges is processed by only one execution instance. When machine resources suffice, you can increase this variable to allow more execution instances to process a scan range simultaneously for efficiency purposes.

The number of scan instances determines the number of other execution nodes in the upper level, such as aggregation nodes and join nodes. Therefore, it increases the concurrency of the entire query plan execution. Modifying this variable will help improve efficiency, but larger values will consume more machine resources, such as CPU, memory, and disk IO.

parallel_merge_late_materialization_mode

- **Описание**: The late materialization mode of parallel merge for sorting. Valid values:
  - `AUTO`
  - `ALWAYS`
  - `NEVER`
- **По умолчанию**: `AUTO`
- **Представлено в**: v1.5.2

partial_update_mode

- **Описание**: Used to control the mode of partial updates. Valid values:

  - `auto` (default): The system automatically determines the mode of partial updates by analyzing the UPDATE statement and the columns involved.
  - `column`: The column mode is used for the partial updates, which is particularly suitable for the partial updates which involve a small number of columns and a large number of rows.

  For more information, see UPDATE.

- **По умолчанию**: auto

- **Представлено в**: v1.5.2

performance_schema (global)

Used for compatibility with MySQL JDBC versions 8.0.16 and above. No practical usage.

prefer_compute_node

- **Описание**: Specifies whether the FEs distribute query execution plans to CN nodes. Valid values:
  - `true`: indicates that the FEs distribute query execution plans to CN nodes.
  - `false`: indicates that the FEs do not distribute query execution plans to CN nodes.
- **По умолчанию**: false
- **Представлено в**: v1.5.2

pipeline_dop

- **Описание**: The parallelism of a pipeline instance, which is used to adjust the query concurrency. Default value: 0, indicating the system automatically adjusts the parallelism of each pipeline instance. This variable also controls the parallelism of loading jobs on OLAP tables. You can also set this variable to a value greater than 0. Generally, set the value to half the number of physical CPU cores. From v1.5.2 onwards, Selena adaptively adjusts this variable based on query parallelism.
- **По умолчанию**: 0
- **Тип данных**: Int

pipeline_sink_dop

- **Описание**: The parallelism of sink for loading data into Iceberg tables and Hive tables, and unloading data using INSERT INTO FILES(). It is used to adjust the concurrency of these loading jobs. Default value: 0, indicating the system automatically adjusts the parallelism. You can also set this variable to a value greater than 0.
- **По умолчанию**: 0
- **Тип данных**: Int

pipeline_profile_level

- **Описание**: Controls the level of the query profile. A query profile often has five layers: Fragment, FragmentInstance, Pipeline, PipelineDriver, and Operator. Different levels provide different details of the profile:

  - 0: Selena combines metrics of the profile and shows only a few core metrics.
  - 1: default value. Selena simplifies the profile and combines metrics of the profile to reduce profile layers.
  - 2: Selena retains all the layers of the profile. The profile size is large in this scenario, especially when the SQL query is complex. This value is not recommended.
- **По умолчанию**: 1

- **Тип данных**: Int

query_cache_entry_max_bytes

- **Описание**: The threshold for triggering the Passthrough mode. When the number of bytes or rows from the computation results of a specific tablet accessed by a query exceeds the threshold specified by `query_cache_entry_max_bytes` or `query_cache_entry_max_rows`, the query is switched to Passthrough mode.
- **Допустимые значения**: 0 to 9223372036854775807
- **По умолчанию**: 4194304
- **Единица измерения**: Byte
- **Представлено в**: v1.5.2

query_cache_entry_max_rows

- **Описание**: The upper limit of rows that can be cached. See the description in `query_cache_entry_max_bytes`. Default value: .
- **По умолчанию**: 409600
- **Представлено в**: v1.5.2

query_cache_agg_cardinality_limit

- **Описание**: The upper limit of cardinality for GROUP BY in Query Cache. Query Cache is not enabled if the rows generated by GROUP BY exceeds this value. Default value: 5000000. If `query_cache_entry_max_bytes` or `query_cache_entry_max_rows` is set to 0, the Passthrough mode is used even when no computation results are generated from the involved tablets.
- **По умолчанию**: 5000000
- **Тип данных**: Long
- **Представлено в**: v1.5.2

query_cache_size (global)

Used for MySQL client compatibility. No practical use.

query_cache_type

Used for compatibility with JDBC connection pool C3P0. No practical use.

query_mem_limit

- **Описание**: Used to set the memory limit of a query on each BE node. The default value is 0, which means no limit for it. This item takes effect only after Pipeline Engine is enabled. When the `Memory Exceed Limit` error happens, you could try to increase this variable. Setting it to `0` indicates no limit is imposed.
- **По умолчанию**: 0
- **Единица измерения**: Byte

query_queue_concurrency_limit (global)

- **Описание**: The upper limit of concurrent queries on a BE. It takes effect only after being set greater than `0`. Setting it to `0` indicates no limit is imposed.
- **По умолчанию**: 0
- **Тип данных**: Int

query_queue_cpu_used_permille_limit (global)

- **Описание**: The upper limit of CPU usage permille (CPU usage * 1000) on a BE. It takes effect only after being set greater than `0`. Setting it to `0` indicates no limit is imposed.
- **Диапазон значений**: [0, 1000]
- **По умолчанию**: `0`

query_queue_max_queued_queries (global)

- **Описание**: The upper limit of queries in a queue. When this threshold is reached, incoming queries are rejected. It takes effect only after being set greater than `0`. Setting it to `0` indicates no limit is imposed.
- **По умолчанию**: `1024`.

query_queue_mem_used_pct_limit (global)

- **Описание**: The upper limit of memory usage percentage on a BE. It takes effect only after being set greater than `0`. Setting it to `0` indicates no limit is imposed.

- **Диапазон значений**: [0, 1]
- **По умолчанию**: 0

query_queue_pending_timeout_second (global)

- **Описание**: The maximum timeout of a pending query in a queue. When this threshold is reached, the corresponding query is rejected.
- **По умолчанию**: 300
- **Единица измерения**: Second

query_timeout

- **Описание**: Used to set the query timeout in "seconds". This variable will act on all query statements in the current connection. The default value is 300 seconds. From v1.5.2 onwards, `query_timeout` does not apply to operations involved INSERT (for example, UPDATE, DELETE, CTAS, materialized view refresh, statistics collection, and PIPE).
- **Диапазон значений**: [1, 259200]
- **По умолчанию**: 300
- **Тип данных**: Int
- **Единица измерения**: Second

range_pruner_max_predicate

- **Описание**: The maximum number of IN predicates that can be used for Range partition pruning. Default value: 100. A value larger than 100 may cause the system to scan all tablets, which compromises the query performance.
- **По умолчанию**: 100
- **Представлено в**: v1.5.2

rewrite_count_distinct_to_bitmap_hll

Used to decide whether to rewrite count distinct queries to bitmap_union_count and hll_union_agg.

runtime_filter_on_exchange_node

- **Описание**: Whether to place GRF on Exchange Node after GRF is pushed down across the Exchange operator to a lower-level operator. The default value is `false`, which means GRF will not be placed on Exchange Node after it is pushed down across the Exchange

operator to a lower-level operator. This prevents repetitive use of GRF and reduces the computation time.

However, GRF delivery is a "try-best" process. If the lower-level operator fails to receive the GRF but the GRF is not placed on Exchange Node, data cannot be filtered, which compromises filter performance. `true` means GRF will still be placed on Exchange Node even after it is pushed down across the Exchange operator to a lower-level operator.

- **По умолчанию**: false

runtime_join_filter_push_down_limit

- **Описание**: The maximum number of rows allowed for the Hash table based on which Bloom filter Local RF is generated. Local RF will not be generated if this value is exceeded. This variable prevents the generation of an excessively long Local RF.
- **По умолчанию**: 1024000
- **Тип данных**: Int

runtime_profile_report_interval

- **Описание**: The time interval at which runtime profiles are reported.
- **По умолчанию**: 10
- **Единица измерения**: Second
- **Тип данных**: Int
- **Представлено в**: v1.5.2

scan_olap_partition_num_limit

- **Описание**: The number of partitions allowed to be scanned for a single table in the execution plan.
- **По умолчанию**: 0 (No limit)
- **Представлено в**: v1.5.2

spill_mode (3.0 and later)

The execution mode of intermediate result spilling. Valid values:

- `auto`: Spilling is automatically triggered when the memory usage threshold is reached.
- `force`: Selena forcibly executes spilling for all relevant operators, regardless of memory usage.

This variable takes effect only when the variable `enable_spill` is set to `true`.

spill_storage_volume

- **Описание**: The storage volume with which you want to store the intermediate results of queries that triggered spilling. For more information, see Spill to object storage.
- **По умолчанию**: Empty string
- **Представлено в**: v1.5.2

SQL_AUTO_IS_NULL

Used for compatibility with the JDBC connection pool C3P0. No practical usage.

sql_dialect

- **Описание**: The SQL dialect that is used. For example, you can run the `set sql_dialect = 'trino';` command to set the SQL dialect to Trino, so you can use Trino-specific SQL syntax and functions in your queries.

  > **NOTICE**
  >
  > After you configure Selena to use the Trino dialect, identifiers in queries are not case-sensitive by default. Therefore, you must specify names in lowercase for your databases and tables at database and table creation. If you specify database and table names in uppercase, queries against these databases and tables will fail.

- **Тип данных**: Selena

- **Представлено в**: v1.5.2

sql_mode

Used to specify the SQL mode to accommodate certain SQL dialects. Valid values include:

- `PIPES_AS_CONCAT`: The pipe symbol `|` is used to concatenate strings, for example, `select 'hello ' || 'world'`.
- `ONLY_FULL_GROUP_BY` (Default): The SELECT LIST can only contain GROUP BY columns or aggregate functions.
- `ALLOW_THROW_EXCEPTION`: returns an error instead of NULL when type conversion fails.
- `FORBID_INVALID_DATE`: prohibits invalid dates.
- `MODE_DOUBLE_LITERAL`: interprets floating-point types as DOUBLE rather than DECIMAL.

- `SORT_NULLS_LAST`: places NULL values at the end after sorting.
- `ERROR_IF_OVERFLOW`: returns an error instead of NULL in the case of arithmetic overflow. Currently, only the DECIMAL data type supports this option.
- `GROUP_CONCAT_LEGACY`: uses the `group_concat` syntax of earlier versions. This option is supported from v1.5.2.

You can set only one SQL mode, for example:

```
set sql_mode = 'PIPES_AS_CONCAT';
```

Or, you can set multiple modes at a time, for example:

```
set sql_mode = 'PIPES_AS_CONCAT,ERROR_IF_OVERFLOW,GROUP_CONCAT_LEGACY';
```

sql_safe_updates

Используется для совместимости с MySQL client. Не имеет практического применения.

sql_select_limit

- **Описание**: Used to limit the maximum number of rows returned by a query, which can prevent issues such as insufficient memory or network congestion caused by the query returning too much data.
- **По умолчанию**: Unlimited
- **Тип данных**: Long

statistic_collect_parallel

- **Описание**: Used to adjust the parallelism of statistics collection tasks that can run on BEs. Default value: 1. You can increase this value to speed up collection tasks.
- **По умолчанию**: 1
- **Тип данных**: Int

storage_engine

The types of engines supported by Selena:

- olap (default): Selena system-owned engine.
- mysql: MySQL external tables.

- broker: Access external tables through a broker program.
- elasticsearch or es: Elasticsearch external tables.
- hive: Hive external tables.
- iceberg: Iceberg external tables, supported from v1.5.2.
- hudi: Hudi external tables, supported from v1.5.2.
- jdbc: external table for JDBC-compatible databases, supported from v1.5.2.

streaming_preaggregation_mode

Used to specify the preaggregation mode for the first phase of GROUP BY. If the preaggregation effect in the first phase is not satisfactory, you can use the streaming mode, which performs simple data serialization before streaming data to the destination. Valid values:

- `auto`: The system first tries local preaggregation. If the effect is not satisfactory, it switches to the streaming mode. This is the default value.
- `force_preaggregation`: The system directly performs local preaggregation.
- `force_streaming`: The system directly performs streaming.

system_time_zone

Used to display the time zone of the current system. Cannot be changed.

time_zone

Used to set the time zone of the current session. The time zone can affect the results of certain time functions.

trace_log_mode

- **Описание**: Used to control where to output the logs of query trace profiles. Valid values:
  - `command`: Return query trace profile logs as the **Explain String** after executing TRACE LOGS.
  - `file`: Return query trace profile logs in the FE log file **fe.log** with the class name being `FileLogTracer`.
- **По умолчанию**: `command`
- **Тип данных**: String
- **Представлено в**: v1.5.2

transaction_read_only

- **Описание**: Used for MySQL 5.8 compatibility. The alias is `tx_read_only`. This variable specifies the transaction access mode. `ON` indicates read only and `OFF` indicates readable and writable.
- **По умолчанию**: OFF
- **Представлено в**: v1.5.2

tx_isolation

Используется для совместимости с MySQL client. Не имеет практического применения. The alias is `transaction_isolation`.

use_compute_nodes

- **Описание**: The maximum number of CN nodes that can be used. This variable is valid when `prefer_compute_node=true`. Valid values:
  - `-1`: indicates that all CN nodes are used.
  - `0`: indicates that no CN nodes are used.
- **По умолчанию**: -1
- **Тип данных**: Int
- **Представлено в**: v1.5.2

use_v2_rollup

Used to control the query to fetch data using the rollup index of the segment v2 storage format. This variable is used for validation when going online with segment v2. It is not recommended for other cases.

vectorized_engine_enable (deprecated from v1.5.2 onwards)

Used to control whether the vectorized engine is used to execute queries. A value of `true` indicates that the vectorized engine is used, otherwise the non-vectorized engine is used. The default is `true`. This feature is enabled by default from v1.5.2 onwards and therefore, is deprecated.

version (global)

The MySQL server version returned to the client. The value is the same as FE parameter `mysql_server_version`.

version_comment (global)

The Selena version. Cannot be changed.

wait_timeout

- **Описание**: The number of seconds the server waits for activity on a non-interactive connection before closing it. If a client does not interact with Selena for this length of time, Selena will actively close the connection.
- **По умолчанию**: 28800 (8 hours).
- **Единица измерения**: Second
- **Тип данных**: Int

orc_use_column_names

- **Описание**: Used to specify how columns are matched when Selena reads ORC files from Hive. The default value is `false`, which means columns in ORC files are read based on their ordinal positions in the Hive table definition. If this variable is set to `true`, columns are read based on their names.
- **По умолчанию**: false
- **Представлено в**: v1.5.2

enable_phased_scheduler

- **Описание**: Whether to enable multi-phased scheduling. When multi-phased scheduling is enabled, it will schedule fragments according to their dependencies. For example, the system will first schedule the fragment on the build side of a Shuffle Join, and then the fragment on the probe side (Note that, unlike stage-by-stage scheduling, phased scheduling is still under the MPP execution mode). Enabling multi-phased scheduling can significantly reduce memory usage for a large number of UNION ALL queries.
- **По умолчанию**: false
- **Представлено в**: v1.5.2

phased_scheduler_max_concurrency

- **Описание**: The concurrency for phased scheduler scheduling leaf node fragments. For example, the default value means that, in a large number of UNION ALL Scan queries, at most two scan fragments are allowed to be scheduled at the same time.
- **По умолчанию**: 2
- **Представлено в**: v1.5.2

enable_wait_dependent_event

- **Описание**: Whether Pipeline waits for a dependent operator to finish execution before continuing within the same fragment. For example, in a left join query, when this feature is enabled, the probe-side operator waits for the build-side operator to finish before it starts executing. Enabling this feature can reduce memory usage, but may increase the query latency. However, for queries reused in CTE, enabling this feature may increase memory usage.
- **По умолчанию**: false
- **Представлено в**: v1.5.2

enable_topn_runtime_filter

- **Описание**: Whether to enable TopN Runtime Filter. If this feature is enabled, a runtime filter will be dynamically constructed for ORDER BY LIMIT queries and pushed down to the scan for filtering.
- **По умолчанию**: true
- **Представлено в**: v1.5.2

enable_partition_hash_join

- **Описание**: Whether to enable adaptive Partition Hash Join.
- **По умолчанию**: true
- **Представлено в**: v1.5.2

spill_enable_direct_io

- **Описание**: Whether to skip Page Cache when reading or writing files for Spilling. If this feature is enabled, Spilling will use direct I/O mode to read or write files directly. Direct I/O mode can reduce the impact on the OS Page Cache, but it may cause increases in disk read/write time.
- **По умолчанию**: false

- **Представлено в**: v1.5.2

spill_enable_compaction

- **Описание**: Whether to enable Compaction for small files from Spilling. When this feature is enabled, it reduces memory usage for aggregation and sorting.
- **По умолчанию**: true
- **Представлено в**: v1.5.2

9 AWS IAM policies

Policy в AWS IAM объявляет набор разрешений на конкретный ресурс AWS. После создания policy вам нужно прикрепить её к IAM role или пользователю. Таким образом, IAM role или пользователь получает разрешения, объявленные в policy, для доступа к указанному ресурсу AWS.

Различные операции в Selena требуют, чтобы у вас были разрешения на доступ к различным ресурсам AWS, и поэтому вам нужно настроить различные policies.

Этот раздел предоставляет IAM policies, которые вам нужно настроить для интеграции Selena с различными ресурсами AWS в различных бизнес-сценариях, когда вы выбираете instance profile, assumed role или метод аутентификации на основе IAM пользователя.

9.1 Пакетная загрузка данных из AWS S3

Если вы хотите загрузить данные из вашего S3 bucket, настройте следующую IAM policy:

> **ПРИМЕЧАНИЕ**
>
> Замените `<bucket_name>` в следующем шаблоне JSON policy на имя вашего S3 bucket, в котором хранятся ваши файлы данных.

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Sid": "s3",
          "Effect": "Allow",
          "Action": [
              "s3:GetObject"
          ],
          "Resource": [
              "arn:aws:s3:::<bucket_name>/*"
          ]
      },
      {
          "Sid": "s3list",
          "Effect": "Allow",
          "Action": [
              "s3:ListBucket"
          ],
          "Resource": [
              "arn:aws:s3:::<bucket_name>"
          ]
      }
  ]
}
```

9.2 Чтение/запись AWS S3

Если вы хотите запросить данные из вашего S3 bucket, настройте следующую IAM policy:

> **ПРИМЕЧАНИЕ**
>
> Замените `<bucket_name>` в следующем шаблоне JSON policy на имя вашего S3 bucket, в котором хранятся ваши файлы данных.

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Sid": "s3",
          "Effect": "Allow",
          "Action": [
              "s3:PutObject",
              "s3:GetObject",
              "s3:DeleteObject"
          ],
          "Resource": [
              "arn:aws:s3:::<bucket_name>/*"
          ]
      },
      {
          "Sid": "s3list",
          "Effect": "Allow",
          "Action": [
              "s3:ListBucket"
          ],
          "Resource": [
              "arn:aws:s3:::<bucket_name>"
          ]
      }
  ]
}
```

9.3 Интеграция с AWS Glue

Если вы хотите интегрироваться с вашим AWS Glue Data Catalog, настройте следующую IAM policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
          "Effect": "Allow",
          "Action": [
                "glue:BatchCreatePartition",
                "glue:UpdateDatabase",
                "glue:GetConnections",
                "glue:CreateTable",
                "glue:DeleteDatabase",
                "glue:BatchUpdatePartition",
                "glue:GetTables",
                "glue:GetTableVersions",
                "glue:GetPartitions",
                "glue:UpdateTable",
                "glue:BatchGetPartition",
                "glue:DeleteTable",
                "glue:GetDatabases",
                "glue:GetDevEndpoint",
                "glue:GetTable",
                "glue:GetDatabase",
                "glue:GetPartition",
                "glue:GetDevEndpoints",
                "glue:GetConnection",
                "glue:CreateDatabase",
                "glue:CreatePartition",
                "glue:DeletePartition",
                "glue:UpdatePartition"
          ],
          "Resource": [
              "*"
            ]
        }
    ]
}
```

10 HTTP SQL API

Selena v1.5.2 представляет HTTP SQL API для выполнения различных типов запросов с использованием HTTP. В настоящее время этот API поддерживает операторы SELECT, SHOW, EXPLAIN и KILL.

Синтаксис с использованием команды curl:

```
curl -X POST 'http://<fe_ip>:
<fe_http_port>/api/v1/catalogs/<catalog_name>/databases/<database_name>/sql'
\
   -u '<username>:<password>'  -d '{"query": "<sql_query>;",
"sessionVariables":{"<var_name>":<var_value>}}' \
   --header "Content-Type: application/json"
```

10.1 Сообщение запроса

Строка запроса

```
POST 'http://<fe_ip>:
<fe_http_port>/api/v1/catalogs/<catalog_name>/databases/<database_name>/sql'
```

Поле	Описание
fe_ip	IP-адрес FE узла.
fe_http_port	HTTP порт FE.
catalog_name	Имя каталога. В v1.5.2 вы можете использовать этот API для запросов только к внутренним таблицам Selena, что означает, что `<catalog_name>` может быть установлен только в `default_catalog`. Начиная с v1.5.2, вы можете использовать этот API для запросов к таблицам во внешних каталогах.
database_name	Имя базы данных. Если в строке запроса не указано имя базы данных, а в SQL-запросе используется имя таблицы, вы должны добавить к имени таблицы префикс с именем базы данных, например, `database_name.table_name`.

- Запрос данных из разных баз данных в указанном каталоге. Если в SQL-запросе используется таблица, вы должны добавить к имени таблицы префикс с именем базы данных.

```
POST /api/v1/catalogs/<catalog_name>/sql
```

- Запрос данных из указанного каталога и базы данных.

```
POST /api/v1/catalogs/<catalog_name>/databases/<database_name>/sql
```

Метод аутентификации

```
Authorization: Basic <credentials>
```

Используется базовая аутентификация, то есть введите имя пользователя и пароль для `credentials` (`-u '<username>:<password>'`). Если для имени пользователя не установлен пароль, вы можете передать только `<username>:` и оставить пароль пустым. Например, если используется учетная запись root, вы можете ввести `-u 'root:'`.

Тело запроса

```
-d '{"query": "<sql_query>;", "sessionVariables":{"<var_name>":
<var_value>}}'
```

Поле	Описание
query	SQL-запрос в формате STRING. Поддерживаются только операторы SELECT, SHOW, EXPLAIN и KILL. Вы можете выполнить только один SQL-запрос для одного HTTP-запроса.
sessionVariables	Переменная сеанса, которую вы хотите установить для запроса, в формате JSON. Это поле необязательное. По умолчанию пустое. Переменная сеанса, которую вы устанавливаете, действует для того же соединения и становится недействительной при закрытии соединения.

Заголовок запроса

```
--header "Content-Type: application/json"
```

Этот заголовок указывает, что тело запроса является JSON-строкой.

10.2 Ответное сообщение

Код состояния

- 200: HTTP-запрос успешен, и сервер работает нормально до отправки данных клиенту.
- 4xx: Ошибка HTTP-запроса, которая указывает на ошибку клиента.
- `500 Internal Server Error`: HTTP-запрос успешен, но сервер сталкивается с ошибкой до отправки данных клиенту.
- 503: HTTP-запрос успешен, но FE не может предоставить сервис.

Заголовок ответа

`content-type` указывает формат тела ответа. Используется JSON с разделителями новой строки, что означает, что тело ответа состоит из нескольких JSON-объектов, разделенных `\n`.

	Описание
content-type	Формат - JSON с разделителями новой строки, по умолчанию "application/x-ndjson charset=UTF-8".
X-Selena-Query-Id	ID запроса.

Тело ответа

Сбой до отправки запроса

Запрос завершается с ошибкой на стороне клиента или сервер сталкивается с ошибкой до возврата данных клиенту. Тело ответа имеет следующий формат, где `msg` - это информация об ошибке.

```
{
   "status":"FAILED",
   "msg":"xxx"
}
```

Сбой после отправки запроса

Часть результата возвращается, и код состояния HTTP равен 200. Отправка данных приостанавливается, соединение закрывается, и ошибка регистрируется.

Успешное выполнение

Каждая строка в ответном сообщении является JSON-объектом. JSON-объекты разделены `\n`.

- Для оператора SELECT возвращаются следующие JSON-объекты.

Объект	Описание
`connectionId`	ID соединения. Вы можете отменить запрос, который ожидается долгое время, вызвав KILL `<connectionId>`.
`meta`	Представляет столбец. Ключ - `meta`, а значение - JSON-массив, где каждый объект в массиве представляет столбец.
`data`	Строка данных, где ключ - `data`, а значение - JSON-массив, который содержит строку данных.
`statistics`	Статистическая информация о запросе.

- Для оператора SHOW возвращаются `meta`, `data` и `statistics`.
- Для оператора EXPLAIN возвращается объект `explain` для отображения подробного плана выполнения запроса.

В следующем примере используется `\n` в качестве разделителя. Selena передает данные в режиме HTTP chunked. Каждый раз, когда FE получает порцию данных, он передает эту порцию данных клиенту. Клиент может анализировать данные построчно, что устраняет необходимость кэширования данных и необходимость ожидания всех данных, снижая потребление памяти для клиента.

```
{"connectionId": 7}\n
{"meta": [
    {
      "name": "stock_symbol",
      "type": "varchar"
    },
    {
      "name": "closing_price",
      "type": "decimal64(8, 2)"
    },
    {
      "name": "closing_date",
      "type": "datetime"
    }
  ]}\n
{"data": ["JDR", 12.86, "2014-10-02 00:00:00"]}\n
{"data": ["JDR",14.8, "2014-10-10 00:00:00"]}\n
...
{"statistics": {"scanRows": 0,"scanBytes": 0,"returnRows": 9}}
```

10.3 Примеры

Выполнение SELECT-запроса

- Запрос данных из внутренней таблицы Selena (`catalog_name` - это `default_catalog`).

```
curl -X POST
'http://127.0.0.1:8030/api/v1/catalogs/default_catalog/databases/test/sql
-u 'root:' -d '{"query": "select * from agg;"}' --header "Content-Type:
application/json"
```

Результат:

```
{"connectionId":49}
{"meta":[{"name":"no","type":"int(11)"},
{"name":"k","type":"decimal64(10, 2)"},
{"name":"v","type":"decimal64(10, 2)"}]}
{"data":[1,"10.00",null]}
{"data":[2,"10.00","11.00"]}
{"data":[2,"20.00","22.00"]}
{"data":[2,"25.00",null]}
{"data":[2,"30.00","35.00"]}
{"statistics":{"scanRows":0,"scanBytes":0,"returnRows":5}}
```

- Запрос данных из таблицы Iceberg.

```
curl -X POST
'http://172.26.93.145:8030/api/v1/catalogs/iceberg_catalog/databases/ywb/s
-u 'root:' -d '{"query": "select * from iceberg_analyze;"}' --header
"Content-Type: application/json"
```

Результат:

```
{"connectionId":13}
{"meta":[{"name":"k1","type":"int(11)"},
{"name":"k2","type":"int(11)"}]}
{"data":[1,2]}
{"data":[1,1]}
{"statistics":{"scanRows":0,"scanBytes":0,"returnRows":2}}
```

Отмена запроса

Чтобы отменить запрос, который выполняется неожиданно долго, вы можете закрыть соединение. Selena отменит этот запрос, когда обнаружит, что соединение закрыто.

Вы также можете вызвать KILL `connectionId` для отмены этого запроса. Например:

```
curl -X POST
'http://127.0.0.1:8030/api/v1/catalogs/default_catalog/databases/test/sql'
-u 'root:' -d '{"query": "kill 17;"}' --header "Content-Type:
application/json"
```

Вы можете получить `connectionId` из тела ответа или вызвав SHOW PROCESSLIST. Например:

```
curl -X POST
'http://127.0.0.1:8030/api/v1/catalogs/default_catalog/databases/test/sql'
-u 'root:' -d '{"query": "show processlist;"}' --header "Content-Type:
application/json"
```

Выполнение запроса с переменной сеанса, установленной для этого запроса

```
curl -X POST
'http://127.0.0.1:8030/api/v1/catalogs/default_catalog/databases/test/sql'
-u 'root:'  -d '{"query": "SHOW VARIABLES;", "sessionVariables":
{"broadcast_row_limit":14000000}}'  --header "Content-Type:
application/json"
```

11 Information Schema

Selena Information Schema — это база данных внутри каждого экземпляра Selena. Information Schema содержит несколько системных представлений только для чтения, в которых хранится обширная информация о метаданных всех объектов, поддерживаемых экземпляром Selena. Selena Information Schema основана на ANSI Information Schema стандарта SQL-92, но с добавлением представлений и функций, специфичных для Selena.

Начиная с версии v1.5.2, Selena Information Schema поддерживает управление метаданными для внешних каталогов.

11.1 Просмотр метаданных через Information Schema

Вы можете просматривать информацию о метаданных внутри экземпляра Selena, выполняя запросы к представлениям в Information Schema.

В следующем примере проверяется информация о метаданных таблицы с именем `table1` в Selena путем запроса к представлению `tables`.

```
MySQL > SELECT * FROM information_schema.tables WHERE TABLE_NAME like
'table1'\G
*************************** 1. row ***************************
  TABLE_CATALOG: def
   TABLE_SCHEMA: test_db
     TABLE_NAME: table1
     TABLE_TYPE: BASE TABLE
        ENGINE: Selena
        VERSION: NULL
     ROW_FORMAT:
     TABLE_ROWS: 4
 AVG_ROW_LENGTH: 1657
    DATA_LENGTH: 6630
MAX_DATA_LENGTH: NULL
   INDEX_LENGTH: NULL
      DATA_FREE: NULL
 AUTO_INCREMENT: NULL
    CREATE_TIME: 2023-06-13 11:37:00
    UPDATE_TIME: 2023-06-13 11:38:06
     CHECK_TIME: NULL
TABLE_COLLATION: utf8_general_ci
       CHECKSUM: NULL
 CREATE_OPTIONS:
  TABLE_COMMENT:
1 row in set (0.01 sec)
```

11.2 Представления в Information Schema

Selena Information Schema содержит следующие представления метаданных:

11.3 Analyze_status

`analyze_status` предоставляет информацию о статусе задач анализа.

В `analyze_status` представлены следующие поля:

Поле	Описание
Id	ID задачи анализа.
Catalog	Catalog, к которому принадлежит таблица.
Database	База данных, к которой принадлежит таблица.
Table	Название анализируемой таблицы.
Columns	Анализируемые столбцы.

Поле	Описание
Type	Тип задачи анализа. Допустимые значения: `FULL`, `SAMPLE`.
Schedule	Тип расписания задачи анализа. Допустимые значения: `ONCE`, `AUTOMATIC`.
Status	Статус задачи анализа. Допустимые значения: `PENDING`, `RUNNING`, `FINISH`, `FAILED`.
StartTime	Время начала задачи анализа.
EndTime	Время окончания задачи анализа.
Properties	Свойства задачи анализа.
Reason	Причина статуса задачи анализа.

11.4 Applicable_roles

`applicable_roles` предоставляет информацию о ролях, применимых к текущему пользователю.

В `applicable_roles` представлены следующие поля:

Поле	Описание
USER	Пользователь, к которому применима роль.
HOST	Хост, с которого подключается пользователь.
GRANTEE	Пользователь или роль, которая предоставила роль.
GRANTEE_HOST	Хост, с которого подключается предоставляющий.
ROLE_NAME	Название роли.
ROLE_HOST	Хост, связанный с ролью.
IS_GRANTABLE	Может ли роль быть предоставлена другим (`YES` или `NO`).
IS_DEFAULT	Является ли роль ролью по умолчанию (`YES` или `NO`).
IS_MANDATORY	Является ли роль обязательной (`YES` или `NO`).

11.5 Be_bvars

`be_bvars` предоставляет статистическую информацию о bRPC. Вы можете просматривать задержку RPC, QPS и другую статистику для некоторых компонентов Selena.

В `be_bvars` представлены следующие поля:

Поле	Описание
BE_ID	ID BE, где находится bvar.
NAME	Название bvar.
VALUE	Значение bvar.

11.6 Be_cloud_native_compactions

`be_cloud_native_compactions` предоставляет информацию о compaction-транзакциях, выполняющихся на CN (или BE для v1.5.2) shared-data cluster. Compaction-транзакция разделяется на несколько задач на уровне partition, и каждая строка в `be_cloud_native_compactions` соответствует задаче в compaction-транзакции.

В `be_cloud_native_compactions` предоставляются следующие поля:

Поле	Описание
BE_ID	ID CN (BE).
TXN_ID	ID compaction-транзакции. Может дублироваться, поскольку каждая compaction-транзакция может содержать несколько задач.
TABLET_ID	ID tablet, которому соответствует задача.
VERSION	Версия данных, которые поступают на вход задачи.
SKIPPED	Была ли задача пропущена.
RUNS	Количество выполнений задачи. Значение больше `1` указывает на то, что были повторные попытки.
START_TIME	Время начала задачи.
FINISH_TIME	Время завершения задачи. Возвращается `NULL`, если задача все еще выполняется.
PROGRESS	Процент выполнения, в диапазоне от 0 до 100.
STATUS	Статус задачи.
PROFILE	Профиль в реальном времени, включающий время чтения данных, время записи данных, время в очереди и т.д.

11.7 Be_compactions

`be_compactions` предоставляет статистическую информацию о задачах compaction.

В `be_compactions` предоставляются следующие поля:

Поле	Описание
BE_ID	ID BE.
CANDIDATES_NUM	Количество кандидатов для задач compaction.
BASE_COMPACTION_CONCURRENCY	Количество выполняющихся задач base compaction.

Поле	Описание
CUMULATIVE_COMPACTION_CONCURRENCY	Количество выполняющихся задач cumulative compaction.
LATEST_COMPACTION_SCORE	Оценка compaction последней задачи compaction.
CANDIDATE_MAX_SCORE	Максимальная оценка compaction среди кандидатов задач.
MANUAL_COMPACTION_CONCURRENCY	Количество выполняющихся задач manual compaction.
MANUAL_COMPACTION_CANDIDATES_NUM	Количество кандидатов для задач manual compaction.

11.8 Be_configs

`be_configs` предоставляет информацию о параметрах конфигурации каждого узла BE.

В `be_configs` предоставляются следующие поля:

Поле	Описание
BE_ID	ID узла BE.
NAME	Название параметра конфигурации.
VALUE	Текущее значение параметра конфигурации.
TYPE	Тип данных параметра конфигурации.
DEFAULT	Значение по умолчанию параметра конфигурации.
MUTABLE	Указывает, может ли параметр конфигурации быть изменен динамически (`true`) или нет (`false`).

11.9 Be_logs

`be_logs` предоставляет информацию о логах каждой BE-ноды.

В `be_logs` доступны следующие поля:

Поле	Описание
BE_ID	ID BE-ноды.
LEVEL	Уровень лога (например, `INFO`, `WARNING`, `ERROR`).
TIMESTAMP	Временная метка записи лога.
TID	ID потока, который создал запись лога.
LOG	Содержимое сообщения лога.

11.10 Be_metrics

`be_metrics` предоставляет информацию о метриках каждой BE-ноды.

В `be_metrics` доступны следующие поля:

Поле	Описание
BE_ID	ID BE-ноды.
NAME	Название метрики.
LABELS	Метки, связанные с метрикой.
VALUE	Текущее значение метрики.

11.11 Be_tablets

`be_tablets` предоставляет информацию о tablets на каждой BE-ноде.

В `be_tablets` доступны следующие поля:

Поле	Описание
BE_ID	ID BE-ноды.
TABLE_ID	ID таблицы, к которой принадлежит tablet.
PARTITION_ID	ID партиции, к которой принадлежит tablet.
TABLET_ID	ID tablet.
NUM_VERSION	Количество версий в tablet.
MAX_VERSION	Максимальная версия tablet.
MIN_VERSION	Минимальная версия tablet.
NUM_ROWSET	Количество rowset в tablet.
NUM_ROW	Количество строк в tablet.
DATA_SIZE	Размер данных tablet (в байтах).
INDEX_MEM	Использование памяти индексом tablet (в байтах).
CREATE_TIME	Время создания tablet (Unix timestamp в секундах).
STATE	Состояние tablet (например, `NORMAL`, `REPLICA_MISSING`).
TYPE	Тип tablet.
DATA_DIR	Директория данных, где хранится tablet.
SHARD_ID	Shard ID tablet.
SCHEMA_HASH	Schema hash tablet.
INDEX_DISK	Использование диска индексом tablet (в байтах).
MEDIUM_TYPE	Тип носителя tablet (например, `HDD`, `SSD`).
NUM_SEGMENT	Количество сегментов в tablet.

11.12 Be_threads

`be_threads` предоставляет информацию о потоках, запущенных на каждом узле BE.

В `be_threads` доступны следующие поля:

Поле	Описание
BE_ID	ID узла BE.
GROUP	Имя группы потоков.
NAME	Имя потока.
PTHREAD_ID	Pthread ID.
TID	Thread ID.
IDLE	Указывает, находится ли поток в режиме ожидания (`true`) или нет (`false`).
FINISHED_TASKS	Количество задач, завершенных потоком.
BOUND_CPUS	CPU, к которым привязан поток.

11.13 Be_txns

`be_txns` предоставляет информацию о транзакциях на каждом узле BE.

В `be_txns` доступны следующие поля:

Поле	Описание
BE_ID	ID узла BE.
LOAD_ID	ID задания загрузки.
TXN_ID	ID транзакции.
PARTITION_ID	ID раздела, задействованного в транзакции.
TABLET_ID	ID tablet, задействованного в транзакции.
CREATE_TIME	Время создания транзакции (Unix timestamp в секундах).
COMMIT_TIME	Время фиксации транзакции (Unix timestamp в секундах).
PUBLISH_TIME	Время публикации транзакции (Unix timestamp в секундах).
ROWSET_ID	ID rowset, задействованного в транзакции.
NUM_SEGMENT	Количество сегментов в rowset.
NUM_DELFILE	Количество файлов удаления в rowset.
NUM_ROW	Количество строк в rowset.
DATA_SIZE	Размер данных rowset (в байтах).
VERSION	Версия rowset.

11.14 Character_sets

`character_sets` определяет доступные наборы символов.

В `character_sets` доступны следующие поля:

Поле	Описание
CHARACTER_SET_NAME	Название набора символов.
DEFAULT_COLLATE_NAME	Правило сортировки по умолчанию для набора символов.
DESCRIPTION	Описание набора символов.
MAXLEN	Максимальное количество байтов, необходимое для хранения одного символа.

11.15 Collations

`collations` содержит доступные collations.

Следующие поля доступны в `collations`:

Поле	Описание
COLLATION_NAME	Имя collation.
CHARACTER_SET_NAME	Имя набора символов, с которым связан collation.
ID	ID collation.
IS_DEFAULT	Является ли collation дефолтным для его набора символов.
IS_COMPILED	Скомпилирован ли набор символов в сервер.
SORTLEN	Связано с объемом памяти, необходимой для сортировки строк, выраженных в наборе символов.

11.16 Column_privileges

> **ПРИМЕЧАНИЕ**
>
> Это представление не применяется к доступным функциям в Selena.

`column_privileges` идентифицирует все привилегии, предоставленные на столбцы текущей активной роли или текущей активной ролью.

Следующие поля доступны в `column_privileges`:

Поле	Описание
GRANTEE	Имя пользователя, которому предоставлена привилегия.

Поле	Описание
TABLE_CATALOG	Имя каталога, к которому принадлежит таблица, содержащая столбец. Это значение всегда `def`.
TABLE_SCHEMA	Имя базы данных, к которой принадлежит таблица, содержащая столбец.
TABLE_NAME	Имя таблицы, содержащей столбец.
COLUMN_NAME	Имя столбца.
PRIVILEGE_TYPE	Предоставленная привилегия. Значением может быть любая привилегия, которая может быть предоставлена на уровне столбца. Каждая строка перечисляет одну привилегию, поэтому для каждой привилегии столбца, принадлежащей получателю, существует одна строка.
IS_GRANTABLE	`YES`, если у пользователя есть привилегия `GRANT OPTION`, `NO` в противном случае. Вывод не содержит `GRANT OPTION` в виде отдельной строки с `PRIVILEGE_TYPE='GRANT OPTION'`.

11.17 Column_stats_usage

`column_stats_usage` предоставляет информацию об использовании статистики столбцов.

Следующие поля доступны в `column_stats_usage`:

Поле	Описание
TABLE_CATALOG	Имя каталога таблицы.
TABLE_DATABASE	Имя базы данных таблицы.
TABLE_NAME	Имя таблицы.
COLUMN_NAME	Имя столбца.
USAGE	Сценарий использования статистики этого столбца.
LAST_USED	Время последнего использования статистики этого столбца.
CREATED	Время создания статистики этого столбца.

11.18 Columns

`columns` содержит информацию обо всех столбцах таблиц (или столбцах представлений).

> **ПРИМЕЧАНИЕ**
>
> Метаданные синхронных materialized views не записываются в `columns`. Вы можете получить к ним доступ, выполнив `SHOW PROC '/dbs/db/table/index_schema'`.

Следующие поля доступны в `columns`:

Поле	Описание
TABLE_CATALOG	Имя каталога, к которому принадлежит таблица, содержащая столбец. Это значение всегда `NULL`.
TABLE_SCHEMA	Имя базы данных, к которой принадлежит таблица, содержащая столбец.
TABLE_NAME	Имя таблицы, содержащей столбец.
COLUMN_NAME	Имя столбца.
ORDINAL_POSITION	Порядковый номер столбца в таблице.
COLUMN_DEFAULT	Значение по умолчанию для столбца. Это `NULL`, если столбец имеет явное значение по умолчанию `NULL` или если определение столбца не включает предложение `DEFAULT`.
IS_NULLABLE	Допустимость NULL для столбца. Значение `YES`, если в столбце могут храниться значения `NULL`, `NO`, если нет.
DATA_TYPE	Тип данных столбца. Значение `DATA_TYPE` — это только имя типа без другой информации. Значение `COLUMN_TYPE` содержит имя типа и, возможно, другую информацию, такую как точность или длина.
CHARACTER_MAXIMUM_LENGTH	Для строковых столбцов — максимальная длина в символах.
CHARACTER_OCTET_LENGTH	Для строковых столбцов — максимальная длина в байтах.
NUMERIC_PRECISION	Для числовых столбцов — числовая точность.
NUMERIC_SCALE	Для числовых столбцов — числовой масштаб.
DATETIME_PRECISI	Для временных столбцов — точность дробных секунд.

Поле	Описание
ON	
CHARACTER_SET_NAME	Для строковых столбцов — имя набора символов.
COLLATION_NAME	Для строковых столбцов — имя collation.
COLUMN_TYPE	Тип данных столбца. Значение `DATA_TYPE` — это только имя типа без другой информации. Значение `COLUMN_TYPE` содержит имя типа и, возможно, другую информацию, такую как точность или длина.
COLUMN_KEY	Проиндексирован ли столбец: • Если `COLUMN_KEY` пуст, столбец либо не проиндексирован, либо проиндексирован только как вторичный столбец в многостолбцовом, неуникальном индексе. • Если `COLUMN_KEY` имеет значение `PRI`, столбец является `PRIMARY KEY` или одним из столбцов в многостолбцовом `PRIMARY KEY`. • Если `COLUMN_KEY` имеет значение `UNI`, столбец является первым столбцом `UNIQUE` индекса. (`UNIQUE` индекс допускает несколько значений `NULL`, но вы можете определить, допускает ли столбец `NULL`, проверив столбец `Null`.) • Если `COLUMN_KEY` имеет значение `DUP`, столбец является первым столбцом неуникального индекса, в котором разрешены множественные вхождения данного значения в столбце. Если к данному столбцу таблицы применяется более одного значения `COLUMN_KEY`, `COLUMN_KEY` отображает значение с наивысшим приоритетом в порядке `PRI`, `UNI`, `DUP`. `UNIQUE` индекс может отображаться как `PRI`, если он не может содержать значения `NULL` и в таблице нет `PRIMARY KEY`. `UNIQUE` индекс может отображаться как `MUL`, если несколько столбцов образуют составной `UNIQUE` индекс; хотя комбинация столбцов уникальна, каждый столбец все еще может содержать несколько вхождений данного значения.
EXTRA	Любая дополнительная информация, доступная о данном столбце.
PRIVILEGES	Привилегии, которые у вас есть для столбца.
COLUMN_COMMENT	Комментарии к столбцу.
COLUMN_SIZE	Максимальная длина столбца в байтах.
DECIMAL_DIGITS	Количество дробных цифр для числовых типов.
GENERATION_EXPRESSION	Выражение, используемое для генерации значения столбца.
SRS_ID	Идентификатор системы пространственных координат для пространственных столбцов.

Поле	Описание
COLUMN_SIZE	
DECIMAL_DIGITS	
GENERATION_EXPRESSION	Для сгенерированных столбцов отображает выражение, используемое для вычисления значений столбца. Пусто для несгенерированных столбцов.
SRS_ID	Это значение применяется к пространственным столбцам. Оно содержит значение `SRID` столбца, которое указывает систему пространственных координат для значений, хранящихся в столбце.

11.19 Engines

> **ПРИМЕЧАНИЕ**
>
> Это представление не применяется к доступным функциям в Selena.

`engines` предоставляет информацию о движках хранения.

В `engines` предоставляются следующие поля:

Поле	Описание
ENGINE	Название движка хранения.
SUPPORT	Уровень поддержки движка хранения сервером. Допустимые значения: • `YES`: движок поддерживается и активен. • `DEFAULT`: как `YES`, плюс это движок по умолчанию. • `NO`: движок не поддерживается. • `DISABLED`: движок поддерживается, но был отключен.
COMMENT	Краткое описание движка хранения.
TRANSACTIONS	Поддерживает ли движок хранения транзакции.
XA	Поддерживает ли движок хранения XA-транзакции.
SAVEPOINTS	Поддерживает ли движок хранения savepoints.

11.20 Events

> **ПРИМЕЧАНИЕ**
>
> Это представление не применяется к доступным функциям в Selena.

`events` предоставляет информацию о событиях Event Manager.

В `events` предоставляются следующие поля:

Поле	Описание
EVENT_CATALOG	Название каталога, к которому принадлежит событие. Это значение всегда `def`.
EVENT_SCHEMA	Название базы данных, к которой принадлежит событие.
EVENT_NAME	Название события.
DEFINER	Пользователь, указанный в предложении `DEFINER` (часто пользователь, создавший событие).
TIME_ZONE	Часовой пояс события, который используется для планирования события и действует во время выполнения события. Значение по умолчанию — `SYSTEM`.
EVENT_BODY	Язык, используемый для инструкций в предложении `DO` события. Значение всегда `SQL`.
EVENT_DEFINITION	Текст SQL-инструкции, составляющей предложение `DO` события; другими словами, инструкция, выполняемая этим событием.
EVENT_TYPE	Тип повторения события: `ONE TIME` (одноразовое) или `RECURRING` (повторяющееся).
EXECUTE_AT	Для одноразового события это значение `DATETIME`, указанное в предложении `AT` инструкции `CREATE EVENT`, использованной для создания события, или последней инструкции `ALTER EVENT`, изменившей событие. Значение, показанное в этом столбце, отражает добавление или вычитание любого значения `INTERVAL`, включенного в предложение `AT` события. Например, если событие создается с помощью `ON SCHEDULE AT CURRENT_DATETIME + '1:6' DAY_HOUR`, и событие было создано 2018-02-09 14:05:30, значение в этом столбце будет `'2018-02-10 20:05:30'`. Если время события определяется предложением `EVERY` вместо предложения `AT` (то есть если событие повторяющееся), значение этого столбца равно `NULL`.
INTERVAL	Для повторяющегося события — количество интервалов ожидания между выполнениями события. Для одноразового события значение всегда `NULL`.

Поле	Описание
_VALUE	
INTERVAL_FIELD	Единицы времени, используемые для интервала, в течение которого повторяющееся событие ожидает перед повторением. Для одноразового события значение всегда `NULL`.
SQL_MODE	SQL-режим, действовавший при создании или изменении события и в котором событие выполняется.
STARTS	Дата и время начала повторяющегося события. Отображается как значение `DATETIME` и равно `NULL`, если для события не определены дата и время начала. Для одноразового события этот столбец всегда `NULL`. Для повторяющегося события, определение которого включает предложение `STARTS`, этот столбец содержит соответствующее значение `DATETIME`. Как и в столбце `EXECUTE_AT`, это значение вычисляет любые используемые выражения. Если нет предложения `STARTS`, влияющего на время события, этот столбец равен `NULL`.
ENDS	Для повторяющегося события, определение которого включает предложение `ENDS`, этот столбец содержит соответствующее значение `DATETIME`. Как и в столбце `EXECUTE_AT`, это значение вычисляет любые используемые выражения. Если нет предложения `ENDS`, влияющего на время события, этот столбец равен `NULL`.
STATUS	Статус события. Один из: `ENABLED`, `DISABLED` или `SLAVESIDE_DISABLED`. `SLAVESIDE_DISABLED` указывает, что создание события произошло на другом сервере MySQL, действующем как источник репликации, и было реплицировано на текущий сервер MySQL, действующий как replica, но событие в настоящее время не выполняется на replica.
ON_COMPLETION	Допустимые значения: `PRESERVE` и `NOT PRESERVE`.
CREATED	Дата и время создания события. Это значение `DATETIME`.
LAST_ALTERED	Дата и время последнего изменения события. Это значение `DATETIME`. Если событие не было изменено с момента создания, это значение совпадает со значением `CREATED`.
LAST_EXECUTED	Дата и время последнего выполнения события. Это значение `DATETIME`. Если событие никогда не выполнялось, этот столбец равен `NULL`. `LAST_EXECUTED` указывает, когда событие запустилось. В результате столбец `ENDS` никогда не меньше `LAST_EXECUTED`.
EVENT_COM	Текст комментария, если событие имеет комментарий. Если нет, это значение пустое.

Поле	Описание
MENT	
ORIGINATOR	ID сервера MySQL, на котором было создано событие; используется в репликации. Это значение может быть обновлено с помощью `ALTER EVENT` на ID сервера, на котором выполняется эта инструкция, если она выполняется на источнике репликации. Значение по умолчанию — 0.
CHARACTER_SET_CLIENT	Значение сеанса системной переменной `character_set_client` при создании события.
COLLATION_CONNECTION	Значение сеанса системной переменной `collation_connection` при создании события.
DATABASE_COLLATION	Collation базы данных, с которой связано событие.

11.21 Fe_metrics

`fe_metrics` предоставляет информацию о метриках каждого FE-узла.

В `fe_metrics` предоставляются следующие поля:

Поле	Описание
FE_ID	ID FE-узла.
NAME	Название метрики.
LABELS	Метки, связанные с метрикой.
VALUE	Текущее значение метрики.

11.22 Fe_tablet_schedules

`fe_tablet_schedules` предоставляет информацию о задачах планирования tablet на узлах FE.

В `fe_tablet_schedules` доступны следующие поля:

Поле	Описание
TABLET_ID	ID Tablet.
TABLE_ID	ID таблицы, к которой принадлежит Tablet.
PARTITION_ID	ID партиции, к которой принадлежит Tablet.
TYPE	Тип задачи. Допустимые значения: `REPAIR` и `BALANCE`.
STATE	Статус задачи.
SCHEDULE_REASON	Причина планирования задачи.
MEDIUM	Носитель данных, на котором находится Tablet.
PRIORITY	Текущий приоритет задачи.
ORIG_PRIORITY	Исходный приоритет задачи.
LAST_PRIORITY_ADJUST_TIME	Время последней корректировки приоритета задачи.
VISIBLE_VERSION	Видимая версия данных Tablet.
COMMITTED_VERSION	Зафиксированная версия данных Tablet.
SRC_BE_ID	ID узла BE, на котором находится исходная replica.
SRC_PATH	Путь, где находится исходная replica.
DEST_BE_ID	ID узла BE, на котором находится целевая replica.
DEST_PATH	Путь, где находится целевая replica.
CREATE_TIME	Время создания задачи.
SCHEDULE_TIME	Время начала выполнения планирования задачи.
FINISH_TIME	Время завершения задачи.
CLONE_BYTES	Размер клонированного файла в байтах.
CLONE_DURATION	Время, затраченное на клонирование, в секундах.
CLONE_RATE	Скорость клонирования в МБ/с.
FAILED_SCHEDULE_COUNT	Количество неудачных попыток планирования задачи.
FAILED_RUNNING_COUNT	Количество неудачных попыток выполнения задачи.
MSG	Информация о планировании и выполнении задачи.

11.23 Global_variables

`global_variables` предоставляет информацию о глобальных переменных.

В `global_variables` доступны следующие поля:

Поле	Описание
VARIABLE_NAME	Имя переменной.
VARIABLE_VALUE	Значение переменной.

11.24 Key_column_usage

> **ПРИМЕЧАНИЕ**
>
> Это представление не применяется к доступным функциям в Selena.

`key_column_usage` идентифицирует все колонки, которые ограничены каким-либо ограничением уникальности, первичного ключа или внешнего ключа.

В `key_column_usage` предоставляются следующие поля:

Поле	Описание
CONSTRAINT_CATALOG	Имя catalog, к которому принадлежит ограничение. Это значение всегда `def`.
CONSTRAINT_SCHEMA	Имя базы данных, к которой принадлежит ограничение.
CONSTRAINT_NAME	Имя ограничения.
TABLE_CATALOG	Имя catalog, к которому принадлежит таблица. Это значение всегда `def`.
TABLE_SCHEMA	Имя базы данных, к которой принадлежит таблица.
TABLE_NAME	Имя таблицы, которая имеет ограничение.
COLUMN_NAME	Имя колонки, которая имеет ограничение. Если ограничение является внешним ключом, то это колонка внешнего ключа, а не колонка, на которую ссылается внешний ключ.
ORDINAL_POSITION	Позиция колонки в ограничении, а не позиция колонки в таблице. Позиции колонок нумеруются начиная с 1.
POSITION_IN_UNIQUE_CONSTRAINT	`NULL` для ограничений уникальности и первичного ключа. Для ограничений внешнего ключа это колонка является порядковой позицией в ключе таблицы, на которую делается ссылка.
REFERENCED_TABLE_SCHEMA	Имя схемы, на которую ссылается ограничение.
REFERENCED_TABLE_NAME	Имя таблицы, на которую ссылается ограничение.
REFERENCED_COLUMN_NAME	Имя колонки, на которую ссылается ограничение.

11.25 Load_tracking_logs

`load_tracking_logs` предоставляет журналы ошибок задач загрузки. Это представление поддерживается начиная с Selena v1.5.2.

В `load_tracking_logs` предоставляются следующие поля:

Поле	Описание
ID	ID загрузки.
JOB_ID	ID задачи загрузки (устарело).
LABEL	Метка задачи загрузки.
DATABASE_NAME	База данных, к которой принадлежит задача загрузки.
TRACKING_LOG	Ошибки (если есть) задачи загрузки.

ПОДСКАЗКА

Для запроса представления `load_tracking_logs` вам нужно будет фильтровать либо по `JOB_ID`, либо по `LABEL`.

Вы можете получить `JOB_ID` или `LABEL` задачи загрузки из представления `information_schema.loads`.

Пример:

```
SELECT * from information_schema.load_tracking_logs WHERE label
='user_behavior'\G
*************************** 1. row ***************************
       JOB_ID: 10141
        LABEL: user_behavior
DATABASE_NAME: mydatabase
 TRACKING_LOG: NULL
         TYPE: BROKER
1 row in set (0.02 sec)
```

11.26 Loads

`loads` предоставляет результаты задач загрузки. Это представление поддерживается начиная с Selena v1.5.2.

В `loads` предоставляются следующие поля:

Поле	Описание
ID	Глобально уникальный идентификатор.
LABEL	Метка задачи загрузки.

Поле	Описание
PROFILE_ID	ID профиля, который можно проанализировать через `ANALYZE PROFILE`.
DB_NAME	База данных, к которой принадлежит целевая таблица.
TABLE_NAME	Целевая таблица.
USER	Пользователь, который инициирует задачу загрузки.
WAREHOUSE	Warehouse, к которому принадлежит задача загрузки.
STATE	Состояние задачи загрузки. Допустимые значения: • `PENDING`/`BEGIN`: Задача загрузки создана. • `QUEUEING`/`BEFORE_LOAD`: Задача загрузки находится в очереди, ожидая планирования. • `LOADING`: Задача загрузки выполняется. • `PREPARING`: Транзакция находится в процессе предварительной фиксации. • `PREPARED`: Транзакция предварительно зафиксирована. • `COMMITED`: Транзакция зафиксирована. • `FINISHED`: Задача загрузки завершена успешно. • `CANCELLED`: Задача загрузки завершилась с ошибкой.
PROGRESS	Прогресс стадий ETL и LOADING задачи загрузки.
TYPE	Тип задачи загрузки. Для Broker Load возвращаемое значение - `BROKER`. Для INSERT возвращаемое значение - `INSERT`. Для Stream Load возвращаемое значение - `STREAM`. Для Routine Load возвращаемое значение - `ROUTINE`.
PRIORITY	Приоритет задачи загрузки. Допустимые значения: `HIGHEST`, `HIGH`, `NORMAL`, `LOW` и `LOWEST`.
SCAN_ROWS	Количество сканированных строк данных.
SCAN_BYTES	Количество сканированных байтов.
FILTERED_ROWS	Количество строк данных, отфильтрованных из-за недостаточного качества данных.
UNSELECTED_ROWS	Количество строк данных, отфильтрованных из-за условий, указанных в предложении WHERE.
SINK_ROWS	Количество загруженных строк данных.
RUNTIME_DE	Метаданные выполнения загрузки. Подробности см. в RUNTIME_DETAILS.

Поле	Описание
TAILS	
CREATE_TIME	Время создания задачи загрузки. Формат: `yyyy-MM-dd HH:mm:ss`. Пример: `2023-07-24 14:58:58`.
LOAD_START_TIME	Время начала стадии LOADING задачи загрузки. Формат: `yyyy-MM-dd HH:mm:ss`. Пример: `2023-07-24 14:58:58`.
LOAD_COMMIT_TIME	Время фиксации транзакции загрузки. Формат: `yyyy-MM-dd HH:mm:ss`. Пример: `2023-07-24 14:58:58`.
LOAD_FINISH_TIME	Время завершения стадии LOADING задачи загрузки. Формат: `yyyy-MM-dd HH:mm:ss`. Пример: `2023-07-24 14:58:58`.
PROPERTIES	Статические свойства задачи загрузки. Подробности см. в PROPERTIES.
ERROR_MSG	Сообщение об ошибке задачи загрузки. Если задача загрузки не столкнулась с ошибкой, возвращается `NULL`.
TRACKING_SQL	SQL-запрос, который можно использовать для запроса журнала отслеживания задачи загрузки. SQL-запрос возвращается только тогда, когда задача загрузки включает некачественные строки данных. Если задача загрузки не включает некачественные строки данных, возвращается `NULL`.
REJECTED_RECORD_PATH	Путь, по которому вы можете получить доступ ко всем некачественным строкам данных, отфильтрованным в задаче загрузки. Количество зарегистрированных некачественных строк данных определяется параметром `log_rejected_record_num`, настроенным в задаче загрузки. Вы можете использовать команду `wget` для доступа к пути. Если задача загрузки не включает некачественные строки данных, возвращается `NULL`.

RUNTIME_DETAILS

- Универсальные метрики:

Метрика	Описание
load_id	Глобально уникальный ID плана выполнения загрузки.
txn_id	ID транзакции загрузки.

- Специфические метрики для Broker Load, INSERT INTO и Spark Load:

Метрика	Описание
etl_info	Детали ETL. Это поле действительно только для задач Spark Load. Для других типов задач загрузки значение будет пустым.
etl_start_time	Время начала стадии ETL задачи загрузки. Формат: `yyyy-MM-dd HH:mm:ss`. Пример: `2023-07-24 14:58:58`.

Метрика	Описание
etl_start_time	Время завершения стадии ETL задачи загрузки. Формат: `yyyy-MM-dd HH:mm:ss`. Пример: `2023-07-24 14:58:58`.
unfinished_backends	Список BE с незавершенными выполнениями.
backends	Список BE, участвующих в выполнении.
file_num	Количество прочитанных файлов.
file_size	Общий размер прочитанных файлов.
task_num	Количество подзадач.

- Специфические метрики для Routine Load:

Метрика	Описание
schedule_interval	Интервал планирования Routine Load.
wait_slot_time	Время, прошедшее пока задача Routine Load ожидает слотов выполнения.
check_offset_time	Время, затраченное на проверку информации offset во время планирования задачи Routine Load.
consume_time	Время, затраченное задачей Routine Load на чтение исходных данных.
plan_time	Время генерации плана выполнения.
commit_publish_time	Время, затраченное на выполнение COMMIT RPC.

- Специфические метрики для Stream Load:

Метрика	Описание
timeout	Таймаут для задач загрузки.
begin_txn_ms	Время, затраченное на начало транзакции.
plan_time_ms	Время генерации плана выполнения.
receive_data_time_ms	Время получения данных.
commit_publish_time_ms	Время, затраченное на выполнение COMMIT RPC.
client_ip	IP-адрес клиента.

PROPERTIES

- Специфические свойства для Broker Load, INSERT INTO и Spark Load:

Свойство	Описание
timeout	Таймаут для задач загрузки.

Свойство	Описание
max_filter_ratio	Максимальное соотношение строк данных, отфильтрованных из-за недостаточного качества данных.

- Специфические свойства для Routine Load:

Свойство	Описание
job_name	Имя задачи Routine Load.
task_num	Количество подзадач, фактически выполняемых параллельно.
timeout	Таймаут для задач загрузки.

11.27 Materialized_views

`materialized_views` предоставляет информацию обо всех материализованных представлениях.

В `materialized_views` доступны следующие поля:

Поле	Описание
MATERIALIZED_VIEW_ID	ID материализованного представления.
TABLE_SCHEMA	База данных, в которой находится материализованное представление.
TABLE_NAME	Имя материализованного представления.
REFRESH_TYPE	Тип обновления материализованного представления. Допустимые значения: `ROLLUP` (синхронное материализованное представление), `ASYNC` (материализованное представление с асинхронным обновлением) и `MANUAL` (материализованное представление с ручным обновлением). Когда значение равно `ROLLUP`, все поля, связанные со статусом активации и обновлением, пусты.
IS_ACTIVE	Указывает, активно ли материализованное представление. Неактивные материализованные представления не могут быть обновлены или запрошены.
INACTIVE_REASON	Причина, по которой материализованное представление неактивно.
PARTITION_TYPE	Тип стратегии партиционирования для материализованного представления.
TASK_ID	ID задачи, отвечающей за обновление материализованного представления.
TASK_NAME	Имя задачи, отвечающей за обновление материализованного представления.
LAST_REFRESH_START_T	Время начала последней задачи обновления.

Поле	Описание
IME	
LAST_REFRESH_FINISHED_TIME	Время окончания последней задачи обновления.
LAST_REFRESH_DURATION	Продолжительность последней задачи обновления.
LAST_REFRESH_STATE	Состояние последней задачи обновления.
LAST_REFRESH_FORCE_REFRESH	Указывает, было ли последнее обновление принудительным.
LAST_REFRESH_START_PARTITION	Начальная партиция для последней задачи обновления.
LAST_REFRESH_END_PARTITION	Конечная партиция для последней задачи обновления.
LAST_REFRESH_BASE_REFRESH_PARTITIONS	Партиции базовой таблицы, задействованные в последней задаче обновления.
LAST_REFRESH_MV_REFRESH_PARTITIONS	Партиции материализованного представления, обновленные в последней задаче обновления.
LAST_REFRESH_ERROR_CODE	Код ошибки последней задачи обновления.
LAST_REFRESH_ERROR_MESSAGE	Сообщение об ошибке последней задачи обновления.
TABLE_ROWS	Количество строк данных в материализованном представлении на основе приблизительной фоновой статистики.
MATERIALIZED_VIEW_DEFINITION	SQL-определение материализованного представления.
EXTRA_MESSAGE	Дополнительное сообщение материализованного представления.

Поле	Описание
QUERY_REWRITE_STATUS	Статус переписывания запросов материализованного представления.
CREATOR	Создатель материализованного представления.
LAST_REFRESH_PROCESS_TIME	Время обработки последней задачи обновления.
LAST_REFRESH_JOB_ID	ID задания последней задачи обновления.

11.28 Partitions

> **ПРИМЕЧАНИЕ**
>
> Это представление не применяется к функциям, доступным в Selena.

`partitions` предоставляет информацию о партициях таблиц.

В `partitions` доступны следующие поля:

Поле	Описание
TABLE_CATALOG	Имя каталога, к которому принадлежит таблица. Это значение всегда `def`.
TABLE_SCHEMA	Имя базы данных, к которой принадлежит таблица.
TABLE_NAME	Имя таблицы, содержащей партицию.
PARTITION_NAME	Имя партиции.
SUBPARTITION_NAME	Если строка таблицы `PARTITIONS` представляет подпартицию, указывается имя подпартиции; в противном случае `NULL`. Для `NDB`: это значение всегда `NULL`.
PARTITION_ORDINAL_	Все партиции индексируются в том же порядке, в котором они определены, причем `1` — это номер, назначенный первой партиции. Индексация может изменяться по мере добавления, удаления и реорганизации партиций; номер, показанный в этом столбце, отражает текущий порядок с учетом любых изменений индексации.

Поле	Описание
POSITION	
PARTITION_METHOD	Допустимые значения: `RANGE`, `LIST`, `HASH`, `LINEAR HASH`, `KEY` или `LINEAR KEY`.
SUBPARTITION_METHOD	Допустимые значения: `HASH`, `LINEAR HASH`, `KEY` или `LINEAR KEY`
PARTITION_EXPRESSION	Выражение для функции партиционирования, используемой в операторе `CREATE TABLE` или `ALTER TABLE`, который создал текущую схему партиционирования таблицы.
SUBPARTITION_EXPRESSION	Работает таким же образом для выражения подпартиционирования, которое определяет подпартиционирование для таблицы, как `PARTITION_EXPRESSION` для выражения партиционирования, используемого для определения партиционирования таблицы. Если у таблицы нет подпартиций, этот столбец содержит `NULL`.
PARTITION_DESCRIPTION	Этот столбец используется для партиций RANGE и LIST. Для партиции `RANGE` он содержит значение, заданное в предложении `VALUES LESS THAN` партиции, которое может быть либо целым числом, либо `MAXVALUE`. Для партиции `LIST` этот столбец содержит значения, определенные в предложении `VALUES IN` партиции, которое представляет собой список целочисленных значений, разделенных запятыми. Для партиций, у которых `PARTITION_METHOD` отличается от `RANGE` или `LIST`, этот столбец всегда содержит `NULL`.
TABLE_ROWS	Количество строк таблицы в партиции.
AVG_ROW_LENGTH	Средняя длина строк, хранящихся в этой партиции или подпартиции, в байтах. Это то же самое, что `DATA_LENGTH`, деленное на `TABLE_ROWS`.
DATA_LENGTH	Общая длина всех строк, хранящихся в этой партиции или подпартиции, в байтах; то есть общее количество байтов, хранящихся в партиции или подпартиции.
MAX_DATA_LENGTH	Максимальное количество байтов, которое может быть сохранено в этой партиции или подпартиции.
INDEX_LENGTH	Длина файла индекса для этой партиции или подпартиции в байтах.
DATA_FREE	Количество байтов, выделенных для партиции или подпартиции, но не используемых.
CREATE_TIME	Время создания партиции или подпартиции.

Поле	Описание
UPDATE_TIME	Время последнего изменения партиции или подпартиции.
CHECK_TIME	Время последней проверки таблицы, к которой принадлежит эта партиция или подпартиция.
CHECKSUM	Значение контрольной суммы, если есть; в противном случае `NULL`.
PARTITION_COMMENT	Текст комментария, если у партиции есть комментарий. Если нет, это значение пусто.
NODEGROUP	Это группа узлов, к которой принадлежит партиция.
TABLESPACE_NAME	Имя табличного пространства, к которому принадлежит партиция.

11.29 Partitions_meta

`partitions_meta` предоставляет информацию о партициях таблиц.

В `partitions_meta` доступны следующие поля:

Поле	Описание
DB_NAME	Имя базы данных, к которой принадлежит партиция.
TABLE_NAME	Имя таблицы, к которой принадлежит партиция.
PARTITION_NAME	Имя партиции.
PARTITION_ID	ID партиции.
COMPACT_VERSION	Compact-версия партиции.
VISIBLE_VERSION	Видимая версия партиции.
VISIBLE_VERSION_TIME	Время видимой версии партиции.
NEXT_VERSION	Следующая версия партиции.
DATA_VERSION	Версия данных партиции.
VERSION_EPOCH	Эпоха версии партиции.
VERSION_TXN_TYPE	Тип транзакции версии партиции.
PARTITION_KEY	Ключ партиции.
PARTITION_VALUE	Значение партиции (например, `Range` или `List`).
DISTRIBUTION_KEY	Ключ распределения партиции.
BUCKETS	Количество бакетов в партиции.

Поле	Описание
REPLICATION_NUM	Число реплик партиции.
STORAGE_MEDIUM	Носитель хранения партиции.
COOLDOWN_TIME	Время охлаждения партиции.
LAST_CONSISTENCY_CHECK_TIME	Время последней проверки согласованности партиции.
IS_IN_MEMORY	Указывает, находится ли партиция в памяти (`true`) или нет (`false`).
IS_TEMP	Указывает, является ли партиция временной (`true`) или нет (`false`).
DATA_SIZE	Размер данных партиции.
ROW_COUNT	Количество строк в партиции.
ENABLE_DATACACHE	Указывает, включен ли кэш данных для партиции (`true`) или нет (`false`).
AVG_CS	Средний показатель компакции партиции.
P50_CS	50-й процентиль показателя компакции партиции.
MAX_CS	Максимальный показатель компакции партиции.
STORAGE_PATH	Путь хранения партиции.
STORAGE_SIZE	Размер хранилища партиции.
METADATA_SWITCH_VERSION	Версия переключения метаданных партиции.
TABLET_BALANCED	Указывает, сбалансировано ли распределение tablet в партиции.

11.30 Pipe_files

`pipe_files` предоставляет статус файлов данных, которые должны быть загружены через указанный pipe. Это представление поддерживается начиная с Selena v1.5.2.

В `pipe_files` доступны следующие поля:

Поле	Описание
DATABASE_NAME	Имя базы данных, в которой хранится pipe.
PIPE_ID	Уникальный ID pipe.
PIPE_NAME	Имя pipe.
FILE_NAME	Имя файла данных.
FILE_VERSION	Дайджест файла данных.
FILE_SIZE	Размер файла данных. Единица измерения: байты.
LAST_MODIFIED	Время последнего изменения файла данных. Формат: `yyyy-MM-dd HH:mm:ss`. Пример: `2023-07-24 14:58:58`.
LOAD_STATE	Статус загрузки файла данных. Допустимые значения: `UNLOADED`, `LOADING`, `FINISHED` и `ERROR`.

Поле	Описание
STAGED_TIME	Дата и время, когда файл данных был впервые зарегистрирован pipe. Формат: `yyyy-MM-dd HH:mm:ss`. Пример: `2023-07-24 14:58:58`.
START_LOAD_TIME	Дата и время начала загрузки файла данных. Формат: `yyyy-MM-dd HH:mm:ss`. Пример: `2023-07-24 14:58:58`.
FINISH_LOAD_TIME	Дата и время завершения загрузки файла данных. Формат: `yyyy-MM-dd HH:mm:ss`. Пример: `2023-07-24 14:58:58`.
ERROR_MSG	Детали ошибки загрузки для файла данных.

11.31 Pipes

`pipes` предоставляет информацию обо всех pipes, хранящихся в текущей или указанной базе данных. Это представление поддерживается начиная с Selena v1.5.2.

> **ПРИМЕЧАНИЕ**
>
> Вы также можете использовать SHOW PIPES для просмотра pipes, хранящихся в указанной базе данных или в текущей используемой базе данных. Эта команда также поддерживается начиная с v1.5.2.

В `pipes` доступны следующие поля:

Поле	Описание
DATABASE_NAME	Имя базы данных, в которой хранится pipe.
PIPE_ID	Уникальный ID pipe.
PIPE_NAME	Имя pipe.
PROPERTIES	Свойства pipe.
TABLE_NAME	Имя целевой таблицы. Формат: `<database_name>.<table_name>`.
STATE	Статус pipe. Допустимые значения: `RUNNING`, `FINISHED`, `SUSPENDED` и `ERROR`.
LOAD_STATUS	Общий статус файлов данных, загружаемых через pipe, включая следующие подполя: • `loadedFiles`: количество загруженных файлов данных. • `loadedBytes`: объем загруженных данных в байтах. • `loadingFiles`: количество файлов данных, которые в данный момент загружаются.
LAST_ERROR	Детали последней ошибки, произошедшей во время выполнения pipe. Значение по умолчанию: `NULL`.
CREATED_TIME	Дата и время создания pipe. Формат: `yyyy-MM-dd HH:mm:ss`. Пример: `2023-07-24 14:58:58`.

11.32 Recyclebin_catalogs

`recyclebin_catalogs` предоставляет информацию о метаданных удаленных баз данных, таблиц и партиций, временно хранящихся в корзине FE.

В `recyclebin_catalogs` доступны следующие поля:

Поле	Описание
TYPE	Тип удаленных метаданных, включая Database, Table и Partition.
NAME	Имя удаленного объекта.
DB_ID	DB ID удаленного объекта.
TABLE_ID	Table ID удаленного объекта.
PARTITION_ID	Partition ID удаленного объекта.
DROP_TIME	Время удаления объекта.

11.33 Referential_constraints

> **ПРИМЕЧАНИЕ**
>
> Это представление не применяется к доступным функциям в Selena.

`referential_constraints` содержит все ссылочные ограничения (foreign key).

В `referential_constraints` предоставляются следующие поля:

Поле	Описание
CONSTRAINT_CATALOG	Имя каталога, к которому принадлежит ограничение. Это значение всегда `def`.
CONSTRAINT_SCHEMA	Имя базы данных, к которой принадлежит ограничение.
CONSTRAINT_NAME	Имя ограничения.
UNIQUE_CONSTRAINT_CATALOG	Имя каталога, содержащего уникальное ограничение, на которое ссылается данное ограничение. Это значение всегда `def`.
UNIQUE_CONSTRAINT_SCHEMA	Имя схемы, содержащей уникальное ограничение, на которое ссылается данное ограничение.
UNIQUE_CONSTRAINT_NAME	Имя уникального ограничения, на которое ссылается данное ограничение.
MATCH_OPTION	Значение атрибута `MATCH` ограничения. В настоящее время единственное допустимое значение — `NONE`.

Поле	Описание
UPDATE_RULE	Значение атрибута `ON UPDATE` ограничения. Допустимые значения: `CASCADE`, `SET NULL`, `SET DEFAULT`, `RESTRICT` и `NO ACTION`.
DELETE_RULE	Значение атрибута `ON DELETE` ограничения. Допустимые значения: `CASCADE`, `SET NULL`, `SET DEFAULT`, `RESTRICT` и `NO ACTION`.
TABLE_NAME	Имя таблицы. Это значение совпадает со значением в таблице `TABLE_CONSTRAINTS`.
REFERENCED_TABLE_NAME	Имя таблицы, на которую ссылается ограничение.

11.34 Routine_load_jobs

`routine_load_jobs` предоставляет информацию о задачах routine load.

В `routine_load_jobs` предоставляются следующие поля:

Поле	Описание
ID	ID задачи routine load.
NAME	Имя задачи routine load.
CREATE_TIME	Время создания задачи routine load.
PAUSE_TIME	Время приостановки задачи routine load.
END_TIME	Время завершения задачи routine load.
DB_NAME	Имя базы данных, к которой принадлежит задача routine load.
TABLE_NAME	Имя таблицы, в которую загружаются данные.
STATE	Состояние задачи routine load.
DATA_SOURCE_TYPE	Тип источника данных.
CURRENT_TASK_NUM	Количество текущих задач.
JOB_PROPERTIES	Свойства задачи routine load.
DATA_SOURCE_PROPERTIES	Свойства источника данных.
CUSTOM_PROPERTIES	Пользовательские свойства задачи routine load.
STATISTICS	Статистика задачи routine load.
PROGRESS	Прогресс задачи routine load.
REASONS_OF_STATE_CHANGED	Причины изменения состояния.
ERROR_LOG_URLS	URL логов ошибок.
TRACKING_SQL	SQL-запрос для отслеживания.
OTHER_MSG	Прочие сообщения.
LATEST_SOURCE_POSITION	Последняя позиция источника в формате JSON.
OFFSET_LAG	Отставание offset в формате JSON.

© ООО «Платформа Данных Селена», 2026

Поле	Описание
TIMESTAMP_PROGRESS	Прогресс timestamp в формате JSON.

11.35 Routines

> **ПРИМЕЧАНИЕ**
>
> Это представление не применяется к доступным функциям в Selena.

`routines` содержит все хранимые подпрограммы (хранимые процедуры и хранимые функции).

В `routine` предоставляются следующие поля:

Поле	Описание
SPECIFIC_NAME	Имя подпрограммы.
ROUTINE_CATALOG	Имя каталога, к которому принадлежит подпрограмма. Это значение всегда `def`.
ROUTINE_SCHEMA	Имя базы данных, к которой принадлежит подпрограмма.
ROUTINE_NAME	Имя подпрограммы.
ROUTINE_TYPE	`PROCEDURE` для хранимых процедур, `FUNCTION` для хранимых функций.
DTD_IDENTIFIER	Если подпрограмма является хранимой функцией, тип данных возвращаемого значения. Если подпрограмма является хранимой процедурой, это значение пустое.
ROUTINE_BODY	Язык, используемый для определения подпрограммы. Это значение всегда `SQL`.
ROUTINE_DEFINITION	Текст SQL-запроса, выполняемого подпрограммой.
EXTERNAL_NAME	Это значение всегда `NULL`.
EXTERNAL_LANGUAGE	Язык хранимой подпрограммы.
PARAMETER_STYLE	Это значение всегда `SQL`.
IS_DETERMINISTIC	`YES` или `NO`, в зависимости от того, определена ли подпрограмма с характеристикой `DETERMINISTIC`.

Поле	Описание
SQL_DATA_ACCESS	Характеристика доступа к данным для подпрограммы. Значение может быть одним из: `CONTAINS SQL`, `NO SQL`, `READS SQL DATA` или `MODIFIES SQL DATA`.
SQL_PATH	Это значение всегда `NULL`.
SECURITY_TYPE	Характеристика `SQL SECURITY` подпрограммы. Значение может быть `DEFINER` или `INVOKER`.
CREATED	Дата и время создания подпрограммы. Это значение типа `DATETIME`.
LAST_ALTERED	Дата и время последнего изменения подпрограммы. Это значение типа `DATETIME`. Если подпрограмма не изменялась с момента создания, это значение совпадает со значением `CREATED`.
SQL_MODE	Режим SQL, действующий при создании или изменении подпрограммы, и при котором выполняется подпрограмма.
ROUTINE_COMMENT	Текст комментария, если у подпрограммы есть комментарий. Если нет, это значение пустое.
DEFINER	Пользователь, указанный в предложении `DEFINER` (часто пользователь, создавший подпрограмму).
CHARACTER_SET_CLIENT	
COLLATION_CONNECTION	
DATABASE_COLLATION	Сопоставление базы данных, с которой связана подпрограмма.

11.36 Schema_privileges

> **ПРИМЕЧАНИЕ**
>
> Этот view не применим к доступным функциям в Selena.

`schema_privileges` предоставляет информацию о привилегиях базы данных.

В `schema_privileges` представлены следующие поля:

Поле	Описание
GRANTEE	Имя пользователя, которому предоставлена привилегия.
TABLE_CATALOG	Имя каталога, к которому принадлежит схема. Это значение всегда `def`.
TABLE_SCHEMA	Имя схемы.

Поле	Описание
PRIVILEGE_TYPE	Предоставленная привилегия. Каждая строка содержит одну привилегию, поэтому для каждой привилегии схемы, которой обладает получатель, создаётся отдельная строка.
IS_GRANTABLE	`YES`, если пользователь имеет привилегию `GRANT OPTION`, в противном случае `NO`. В выводе `GRANT OPTION` не указывается как отдельная строка с `PRIVILEGE_TYPE='GRANT OPTION'`.

11.37 Schemata

`schemata` предоставляет информацию о базах данных.

В `schemata` представлены следующие поля:

Поле	Описание
CATALOG_NAME	Имя каталога, к которому принадлежит схема. Это значение всегда `NULL`.
SCHEMA_NAME	Имя схемы.
DEFAULT_CHARACTER_SET_NAME	Набор символов по умолчанию для схемы.
DEFAULT_COLLATION_NAME	Сопоставление по умолчанию для схемы.
SQL_PATH	Это значение всегда `NULL`.

11.38 Session_variables

`session_variables` предоставляет информацию о переменных сессии.

В `session_variables` представлены следующие поля:

Поле	Описание
VARIABLE_NAME	Имя переменной.
VARIABLE_VALUE	Значение переменной.

11.39 Statistics

> **ПРИМЕЧАНИЕ**
>
> Этот view не применим к доступным функциям в Selena.

`statistics` предоставляет информацию об индексах таблиц.

В `statistics` представлены следующие поля:

Поле	Описание
TABLE_CATALOG	Имя каталога, к которому принадлежит таблица, содержащая индекс. Это значение всегда `def`.
TABLE_SCHEMA	Имя базы данных, к которой принадлежит таблица, содержащая индекс.
TABLE_NAME	Имя таблицы, содержащей индекс.
NON_UNIQUE	0, если индекс не может содержать дубликаты, 1, если может.
INDEX_SCHEMA	Имя базы данных, к которой принадлежит индекс.
INDEX_NAME	Имя индекса. Если индекс является первичным ключом, имя всегда `PRIMARY`.
SEQ_IN_INDEX	Порядковый номер столбца в индексе, начиная с 1.
COLUMN_NAME	Имя столбца. См. также описание столбца `EXPRESSION`.
COLLATION	Порядок сортировки столбца в индексе. Возможные значения: `A` (по возрастанию), `D` (по убыванию) или `NULL` (без сортировки).
CARDINALITY	Оценка количества уникальных значений в индексе.
SUB_PART	Префикс индекса. То есть количество индексированных символов, если столбец индексирован частично, или `NULL`, если индексирован весь столбец.
PACKED	Указывает, как упакован ключ. `NULL`, если не упакован.
NULLABLE	Содержит `YES`, если столбец может содержать значения `NULL`, и `''`, если нет.
INDEX_TYPE	Используемый метод индексации (`BTREE`, `FULLTEXT`, `HASH`, `RTREE`).
COMMENT	Информация об индексе, не описанная в отдельном столбце, например `disabled`, если индекс отключён.
INDEX_COMMENT	Комментарий, указанный для индекса с атрибутом `COMMENT` при его создании.

11.40 Stream_loads

`stream_loads` предоставляет информацию о задачах потоковой загрузки (stream load).

В `stream_loads` представлены следующие поля:

Поле	Описание
LABEL	Метка задачи потоковой загрузки.

Поле	Описание
ID	Идентификатор задачи потоковой загрузки.
LOAD_ID	Идентификатор загрузки задачи потоковой загрузки.
TXN_ID	Идентификатор транзакции задачи потоковой загрузки.
DB_NAME	Имя базы данных, к которой принадлежит задача потоковой загрузки.
TABLE_NAME	Имя таблицы, в которую загружаются данные.
STATE	Состояние задачи потоковой загрузки.
ERROR_MSG	Сообщение об ошибке, если задача потоковой загрузки завершилась неудачно.
TRACKING_URL	URL для отслеживания задачи потоковой загрузки.
CHANNEL_NUM	Количество каналов, используемых в задаче потоковой загрузки.
PREPARED_CHANNEL_NUM	Количество подготовленных каналов в задаче потоковой загрузки.
NUM_ROWS_NORMAL	Количество загруженных нормальных строк.
NUM_ROWS_AB_NORMAL	Количество загруженных аномальных строк.
NUM_ROWS_UNSELECTED	Количество невыбранных строк.
NUM_LOAD_BYTES	Количество загруженных байт.
TIMEOUT_SECOND	Тайм-аут в секундах для задачи потоковой загрузки.
CREATE_TIME_MS	Время создания задачи потоковой загрузки (в миллисекундах).
BEFORE_LOAD_TIME_MS	Время до начала загрузки (в миллисекундах).
START_LOADING_TIME_MS	Время начала загрузки (в миллисекундах).
START_PREPARING_TIME_MS	Время начала подготовки (в миллисекундах).
FINISH_PREPARING_TIME_MS	Время завершения подготовки (в миллисекундах).
END_TIME_MS	Время завершения задачи потоковой загрузки (в миллисекундах).
CHANNEL_STATE	Состояние каналов.
TYPE	Тип задачи потоковой загрузки.
TRACKING_SQL	SQL-выражение для отслеживания задачи потоковой загрузки.

11.41 Table_constraints

> **ПРИМЕЧАНИЕ**
>
> Этот view не применим к доступным функциям в Selena.

`table_constraints` описывает, какие таблицы имеют ограничения.

В `table_constraints` представлены следующие поля:

Поле	Описание
CONSTRAINT_CATALOG	Имя каталога, к которому принадлежит ограничение. Это значение всегда `def`.
CONSTRAINT_SCHEMA	Имя базы данных, к которой принадлежит ограничение.
CONSTRAINT_NAME	Имя ограничения.
TABLE_SCHEMA	Имя базы данных, к которой принадлежит таблица.
TABLE_NAME	Имя таблицы.
CONSTRAINT_TYPE	Тип ограничения.

11.42 Table_privileges

> **ПРИМЕЧАНИЕ**
>
> Этот view не применим к доступным функциям в Selena.

`table_privileges` предоставляет информацию о привилегиях таблиц.

В `table_privileges` представлены следующие поля:

Поле	Описание
GRANTEE	Имя пользователя, которому предоставлена привилегия.
TABLE_CATALOG	Имя каталога, к которому принадлежит таблица. Это значение всегда `def`.
TABLE_SCHEMA	Имя базы данных, к которой принадлежит таблица.
TABLE_NAME	Имя таблицы.
PRIVILEGE_TYPE	Предоставленная привилегия. Значением может быть любая привилегия, которую можно предоставить на уровне таблицы.
IS_GRANTABLE	`YES`, если пользователь имеет привилегию `GRANT OPTION`, в противном случае `NO`. В выводе `GRANT OPTION` не указывается как отдельная строка с `PRIVILEGE_TYPE='GRANT OPTION'`.

11.43 Tables

`tables` предоставляет информацию о таблицах.

В `tables` представлены следующие поля:

Поле	Описание
TABLE_CATALOG	Имя каталога, в котором хранится таблица.
TABLE_SCHEMA	Имя базы данных, в которой хранится таблица.
TABLE_NAME	Имя таблицы.
TABLE_TYPE	Тип таблицы. Допустимые значения: `TABLE` (`BASE TABLE`), `VIEW` и `SYSTEM VIEW` (поддерживается только когда `mysql_server_version` равен `8.0.33`).
ENGINE	Тип движка таблицы. Допустимые значения: `Selena`, `MySQL`, `MEMORY` или пустая строка.
VERSION	Применимо к функции, недоступной в Selena.
ROW_FORMAT	Применимо к функции, недоступной в Selena.
TABLE_ROWS	Количество строк в таблице.
AVG_ROW_LENGTH	Средняя длина (размер) строки таблицы. Эквивалентно `DATA_LENGTH` / `TABLE_ROWS`. Единица измерения: байт.
DATA_LENGTH	Длина данных таблицы определяется суммированием длины данных таблицы по всем replica. Единица измерения: байт.
MAX_DATA_LENGTH	Применимо к функции, недоступной в Selena.
INDEX_LENGTH	Применимо к функции, недоступной в Selena.
DATA_FREE	Применимо к функции, недоступной в Selena.
AUTO_INCREMENT	Применимо к функции, недоступной в Selena.
CREATE_TIME	Время создания таблицы.
UPDATE_TIME	Время последнего обновления таблицы.
CHECK_TIME	Время последней проверки согласованности таблицы.
TABLE_COLLATION	Сопоставление таблицы по умолчанию.
CHECKSUM	Применимо к функции, недоступной в Selena.
CREATE_OPTIONS	Применимо к функции, недоступной в Selena.
TABLE_COMMENT	Комментарий к таблице.

11.44 Tables_config

`tables_config` предоставляет информацию о конфигурации таблиц.

В `tables_config` представлены следующие поля:

Поле	Описание
TABLE_SCHEMA	Имя базы данных, в которой хранится таблица.
TABLE_NAME	Имя таблицы.
TABLE_ENGINE	Тип движка таблицы.
TABLE_MODEL	Модель данных таблицы. Допустимые значения: `DUP_KEYS`, `AGG_KEYS`, `UNQ_KEYS` или `PRI_KEYS`.
PRIMARY_KEY	Первичный ключ таблицы с первичным ключом (Primary Key table) или таблицы с уникальным ключом (Unique Key table). Возвращается пустая строка, если таблица не является таблицей с первичным или уникальным ключом.
PARTITION_KEY	Столбцы партиционирования таблицы.
DISTRIBUTE_KEY	Столбцы бакетирования таблицы.
DISTRIBUTE_TYPE	Метод распределения данных таблицы.
DISTRIBUTE_BUCKET	Количество бакетов в таблице.
SORT_KEY	Ключи сортировки таблицы.
PROPERTIES	Свойства таблицы.
TABLE_ID	Идентификатор таблицы.

11.45 Task_runs

`task_runs` предоставляет информацию о выполнении асинхронных задач.

В `task_runs` представлены следующие поля:

Поле	Описание
QUERY_ID	Идентификатор запроса.
TASK_NAME	Имя задачи.
CREATE_TIME	Время создания задачи.
FINISH_TIME	Время завершения задачи.
STATE	Состояние задачи. Допустимые значения: `PENDING`, `RUNNING`, `FAILED` и `SUCCESS`. Начиная с версии 3.1.12 добавлено новое состояние `MERGED`, специально для задач обновления материализованных представлений. Когда новая задача обновления отправляется, а старая задача всё ещё находится в очереди ожидания, эти задачи объединяются с сохранением уровня приоритета.
CATALOG	Каталог, к которому принадлежит задача.
DATABASE	База данных, к которой принадлежит задача.
DEFINITION	SQL-определение задачи.
EXPIRE_TIME	Время истечения срока действия задачи.
ERROR_CODE	Код ошибки задачи.
ERROR_MESSAGE	Сообщение об ошибке задачи.
PROGRESS	Прогресс выполнения задачи.
EXTRA_MESSAGE	Дополнительное сообщение для задачи, например, информация о partition в задаче создания асинхронного материализованного представления.
PROPERTIES	Свойства задачи.

Поле	Описание
JOB_ID	Идентификатор задания.
PROCESS_TIME	Время обработки задачи.

Запись о выполнении задачи создаётся либо командой SUBMIT TASK, либо командой CREATE MATRIALIZED VIEW.

> **ПРИМЕЧАНИЕ**
>
> Операция с материализованным представлением может генерировать несколько записей о выполнении задач, и каждая запись представляет подзадачу обновления, которая разделяется в соответствии с конфигурацией `partition_refresh_number`.

EXTRA_MESSAGE

Для задачи выполнения материализованного представления поле `EXTRA_MESSAGE` содержит детальные сообщения о выполнении задачи материализованного представления. Более подробную информацию можно найти в materialized_view_task_run_details.

11.46 Tasks

`tasks` предоставляет информацию об асинхронных задачах.

В `tasks` представлены следующие поля:

Поле	Описание
TASK_NAME	Имя задачи.
CREATE_TIME	Время создания задачи.
SCHEDULE	Расписание задачи. Если задача запускается регулярно, это поле отображает `START xxx EVERY xxx`.
DATABASE	База данных, к которой принадлежит задача.
DEFINITION	SQL-определение задачи.
EXPIRE_TIME	Время истечения срока действия задачи.

Справка

Документация `SUBMIT_TASK`

11.47 Triggers

> **ПРИМЕЧАНИЕ**
>
> Этот view не применим к доступным функциям в Selena.

`triggers` предоставляет информацию о триггерах.

В `triggers` представлены следующие поля:

Поле	Описание
TRIGGER_CATALOG	Имя каталога, к которому принадлежит триггер. Это значение всегда `def`.
TRIGGER_SCHEMA	Имя базы данных, к которой принадлежит триггер.
TRIGGER_NAME	Имя триггера.
EVENT_MANIPULATION	Событие триггера. Это тип операции над связанной таблицей, для которой активируется триггер. Значение: `INSERT` (вставлена строка), `DELETE` (удалена строка) или `UPDATE` (изменена строка).
EVENT_OBJECT_CATALOG	Каждый триггер связан ровно с одной таблицей. Каталог, в котором находится эта таблица.
EVENT_OBJECT_SCHEMA	Каждый триггер связан ровно с одной таблицей. База данных, в которой находится эта таблица.
EVENT_OBJECT_TABLE	Имя таблицы, с которой связан триггер.
ACTION_ORDER	Порядковая позиция действия триггера в списке триггеров той же таблицы с одинаковыми значениями `EVENT_MANIPULATION` и `ACTION_TIMING`.
ACTION_CONDITION	Это значение всегда `NULL`.
ACTION_STATEMENT	Тело триггера; то есть выражение, выполняемое при активации триггера. Этот текст использует кодировку UTF-8.
ACTION_ORIENTATION	Это значение всегда `ROW`.
ACTION_TIMING	Активируется ли триггер до или после события-триггера. Значение: `BEFORE` или `AFTER`.
ACTION_REFERENCE_OLD_TABLE	Это значение всегда `NULL`.

Поле	Описание
ACTION_REFERENCE_NEW_TABLE	Это значение всегда `NULL`.
ACTION_REFERENCE_OLD_ROW	Идентификаторы старых столбцов. Значение всегда `OLD`.
ACTION_REFERENCE_NEW_ROW	Идентификаторы новых столбцов. Значение всегда `NEW`.
CREATED	Дата и время создания триггера. Это значение `DATETIME(2)` (с дробной частью в сотых долях секунды) для триггеров.
SQL_MODE	Режим SQL, действовавший при создании триггера и при котором триггер выполняется.
DEFINER	Пользователь, указанный в выражении `DEFINER` (часто это пользователь, создавший триггер).
CHARACTER_SET_CLIENT	
COLLATION_CONNECTION	
DATABASE_COLLATION	Сопоставление базы данных, с которой связан триггер.

11.48 User_privileges

> **ПРИМЕЧАНИЕ**
>
> Этот view не применим к доступным функциям в Selena.

`user_privileges` предоставляет информацию о привилегиях пользователей.

В `user_privileges` представлены следующие поля:

Поле	Описание
GRANTEE	Имя пользователя, которому предоставлена привилегия.
TABLE_CATALOG	Имя каталога. Это значение всегда `def`.
PRIVILEGE_TYPE	Предоставленная привилегия. Значением может быть любая привилегия, которую можно предоставить на глобальном уровне.
IS_GRANTABLE	`YES`, если пользователь имеет привилегию `GRANT OPTION`, в противном случае `NO`. В выводе `GRANT OPTION` не указывается как отдельная строка с `PRIVILEGE_TYPE='GRANT OPTION'`.

11.49 Verbose_session_variables

`verbose_session_variables` предоставляет подробную информацию о переменных сессии, включая их значения по умолчанию и информацию о том, были ли они изменены.

В `verbose_session_variables` представлены следующие поля:

Поле	Описание
VARIABLE_NAME	Имя переменной сессии.
VARIABLE_VALUE	Текущее значение переменной сессии.
DEFAULT_VALUE	Значение переменной сессии по умолчанию.
IS_CHANGED	Указывает, было ли значение переменной изменено относительно значения по умолчанию (`true`) или нет (`false`).

11.50 Views

`views` предоставляет информацию обо всех пользовательских представлениях (views).

В `views` представлены следующие поля:

Поле	Описание
TABLE_CATALOG	Имя каталога, к которому принадлежит представление. Это значение всегда `def`.
TABLE_SCHEMA	Имя базы данных, к которой принадлежит представление.
TABLE_NAME	Имя представления.
VIEW_DEFINITION	Выражение SELECT, которое определяет представление.
CHECK_OPTION	Значение атрибута `CHECK_OPTION`. Значение может быть одним из: `NONE`, `CASCADE` или `LOCAL`.
IS_UPDATABLE	Является ли представление обновляемым. Флаг установлен в `YES` (истина), если `UPDATE` и `DELETE` (и подобные операции) допустимы для представления. В противном случае флаг установлен в `NO` (ложь). Если представление не обновляемое, такие выражения как `UPDATE`, `DELETE` и `INSERT` недопустимы и будут отклонены.
DEFINER	Пользователь, создавший представление.

Поле	Описание
SECURITY_TYPE	
CHARACTER_SET_CLIENT	
COLLATION_CONNECTION	

11.51 Warehouse_metrics

`warehouse_metrics` предоставляет информацию о метриках каждого warehouse.

В `warehouse_metrics` представлены следующие поля:

Поле	Описание
WAREHOUSE_ID	Идентификатор warehouse.
WAREHOUSE_NAME	Имя warehouse.
QUEUE_PENDING_LENGTH	Количество ожидающих запросов в очереди.
QUEUE_RUNNING_LENGTH	Количество выполняющихся запросов в очереди.
MAX_PENDING_LENGTH	Максимально допустимое количество ожидающих запросов в очереди.
MAX_PENDING_TIME_SECOND	Максимальное время ожидания в секундах для запросов в очереди.
EARLIEST_QUERY_WAIT_TIME	Время ожидания самого раннего запроса.
MAX_REQUIRED_SLOTS	Максимальное требуемое количество слотов.
SUM_REQUIRED_SLOTS	Сумма требуемых слотов.
REMAIN_SLOTS	Оставшиеся слоты.
MAX_SLOTS	Максимальное количество слотов.
EXTRA_MESSAGE	Дополнительное сообщение.

11.52 Warehouse_queries

`warehouse_queries` предоставляет информацию о запросах, выполняющихся на каждом warehouse.

В `warehouse_queries` представлены следующие поля:

Поле	Описание
WAREHOUSE_ID	Идентификатор warehouse.

Поле	Описание
WAREHOUSE_NAME	Имя warehouse.
QUERY_ID	Идентификатор запроса.
STATE	Состояние запроса (например, `PENDING`, `RUNNING`, `FINISHED`).
EST_COSTS_SLOTS	Оценочная стоимость слотов для запроса.
ALLOCATE_SLOTS	Выделенные слоты для запроса.
QUEUED_WAIT_SECONDS	Время ожидания запроса в очереди в секундах.
QUERY	Строка SQL-запроса.
QUERY_START_TIME	Время начала запроса.
QUERY_END_TIME	Время завершения запроса.
QUERY_DURATION	Продолжительность запроса.
EXTRA_MESSAGE	Дополнительное сообщение.

© ООО «Платформа Данных Селена», 2026

12 Системная метабаза данных

В этом разделе описывается, как просматривать эксклюзивные метаданные вашего Selena cluster через системные представления.

Каждый Selena cluster поддерживает базу данных `sys`, которая содержит несколько системных представлений только для чтения. Эти представления метаданных предоставляют унифицированный и удобный интерфейс, который позволяет получить общее представление о структуре привилегий, зависимостях объектов и другой информации в вашем Selena cluster.

12.1 Просмотр метаданных через sys

Вы можете просматривать эксклюзивную информацию о метаданных в экземпляре Selena, запрашивая содержимое представлений в `sys`.

Следующий пример проверяет привилегии, предоставленные пользовательским ролям, запрашивая представление `grants_to_roles`.

```
MySQL > SELECT * FROM sys.grants_to_roles LIMIT 5\G
*************************** 1. row ***************************
        GRANTEE: role_test
 OBJECT_CATALOG: default_catalog
OBJECT_DATABASE: db_test
    OBJECT_NAME: tbl1
    OBJECT_TYPE: TABLE
 PRIVILEGE_TYPE: SELECT, ALTER
   IS_GRANTABLE: NO
*************************** 2. row ***************************
        GRANTEE: role_test
 OBJECT_CATALOG: default_catalog
OBJECT_DATABASE: db_test
    OBJECT_NAME: tbl2
    OBJECT_TYPE: TABLE
 PRIVILEGE_TYPE: SELECT
   IS_GRANTABLE: YES
*************************** 3. row ***************************
        GRANTEE: role_test
 OBJECT_CATALOG: default_catalog
OBJECT_DATABASE: db_test
    OBJECT_NAME: mv_test
    OBJECT_TYPE: MATERIALIZED VIEW
 PRIVILEGE_TYPE: SELECT
   IS_GRANTABLE: YES
*************************** 4. row ***************************
        GRANTEE: role_test
 OBJECT_CATALOG: NULL
OBJECT_DATABASE: NULL
    OBJECT_NAME: NULL
    OBJECT_TYPE: SYSTEM
 PRIVILEGE_TYPE: CREATE RESOURCE GROUP
   IS_GRANTABLE: NO
```

12.2 Представления в sys

`sys` содержит следующие представления метаданных:

Представление	Описание
grants_to_roles	Записывает информацию о привилегиях, предоставленных пользовательским ролям.
grants_to_users	Записывает информацию о привилегиях, предоставленных пользователям.
role_edges	Записывает получателей ролей.

Представление	Описание
object_dependencies	Записывает отношения зависимости асинхронных материализованных представлений.
policy_references	Записывает объекты, на которые ссылаются политики доступа к строкам и политики маскирования столбцов.
fe_locks	Записывает информацию о блокировках метаданных в Selena FE.

> **ПРИМЕЧАНИЕ**
>
> В соответствии со сценариями их применения, представления в `sys` по умолчанию доступны только некоторым ролям `admin`. В зависимости от ваших конкретных потребностей вы можете предоставить другим пользователям привилегию SELECT на эти представления.

12.3 Fe_locks

`fe_locks` предоставляет информацию о блокировках метаданных в Selena FE.

В `fe_locks` предоставляются следующие поля:

Поле	Описание
lock_type	Тип блокировки (например, "DATABASE").
lock_object	Идентификатор объекта, который заблокирован (имя базы данных, имя таблицы и т.д.).
lock_mode	Режим блокировки. Допустимые значения: `EXCLUSIVE`, `SHARED`.
start_time	Когда была получена блокировка.
hold_time_ms	Продолжительность удержания блокировки (в миллисекундах).
thread_info	JSON-строка, содержащая информацию о потоке (threadId, threadName).
granted	Получена ли блокировка в настоящее время.
waiter_list	Список потоков, ожидающих эту блокировку, разделенный запятыми.

Режимы блокировки

- **EXCLUSIVE**: Только один поток может удерживать эту блокировку одновременно.
- **SHARED**: Несколько потоков могут удерживать эту блокировку одновременно.

Типы блокировок

- **DATABASE**: Блокировки на уровне базы данных (когда `lock_manager_enabled` установлен в false).

- **TABLE**: Блокировки на уровне таблицы (когда `lock_manager_enabled` установлен в true).

Конфигурация

Поведение `fe_locks` зависит от параметра конфигурации `lock_manager_enabled`:

- Когда `lock_manager_enabled = true`: Использует новый Lock Manager для централизованного управления блокировками с детализацией на уровне таблиц.
- Когда `lock_manager_enabled = false`: Использует традиционные блокировки на уровне базы данных.

Примеры

Поиск долго выполняющихся блокировок

```
SELECT lock_object, lock_mode, hold_time_ms, thread_info
FROM information_schema.fe_locks
WHERE hold_time_ms > 10000  -- Блокировки, удерживаемые более 10 секунд
ORDER BY hold_time_ms DESC;
```

Проверка конкуренции за блокировки

```
SELECT lock_object, COUNT(*) as lock_count
FROM information_schema.fe_locks
WHERE granted = true
GROUP BY lock_object
HAVING COUNT(*) > 1;
```

Поиск ожидающих потоков

```
SELECT lock_object, waiter_list
FROM information_schema.fe_locks
WHERE waiter_list != '';
```

Примечания

- Запрос `fe_locks` требует привилегию `OPERATE`.
- Представление предоставляет информацию в реальном времени, но может влиять на производительность систем с высокой активностью блокировок.

- Используйте соответствующие предложения WHERE для фильтрации результатов при работе с большим количеством блокировок.

12.4 Grants_to_roles

Вы можете просматривать привилегии, предоставленные пользовательским ролям, запрашивая представление `grants_to_roles`.

> **ПРИМЕЧАНИЕ**
>
> По умолчанию доступ к этому представлению имеют только пользователи или роли с ролью `user_admin`. Вы можете предоставить другим пользователям привилегию SELECT на это представление, используя GRANT.

В `grants_to_roles` предоставляются следующие поля:

Поле	Описание
GRANTEE	Роль, которой предоставлена эта привилегия. В этом представлении будут перечислены только пользовательские роли, системные роли не отображаются.
OBJECT_CATALOG	Catalog, к которому принадлежит объект. Возвращается `NULL`, если привилегия уровня SYSTEM, RESOURCE GROUP, RESOURCE, USER или GLOBAL FUNCTION.
OBJECT_DATABASE	База данных, к которой принадлежит объект. Возвращается `NULL`, если привилегия уровня SYSTEM, RESOURCE GROUP, RESOURCE, USER, GLOBAL FUNCTION или CATALOG.
OBJECT_NAME	Таблица, к которой принадлежит объект. Возвращается `NULL`, если привилегия уровня SYSTEM, RESOURCE GROUP, RESOURCE, USER, GLOBAL FUNCTION, CATALOG или DATABASE.
OBJECT_TYPE	Тип объекта.
PRIVILEGE_TYPE	Тип привилегии. Различные привилегии на один и тот же объект будут объединены и возвращены в одной строке. Как показано в следующем примере, `role_test` имеет привилегии SELECT и ALTER на `default_catalog.db_test.tbl1`.
IS_GRANTABLE	Имеет ли получатель возможность передачи привилегии (grant option).

Пример:

```
MySQL > SELECT * FROM sys.grants_to_roles LIMIT 5\G
*************************** 1. row ***************************
        GRANTEE: role_test
 OBJECT_CATALOG: default_catalog
OBJECT_DATABASE: db_test
    OBJECT_NAME: tbl1
    OBJECT_TYPE: TABLE
 PRIVILEGE_TYPE: SELECT, ALTER
   IS_GRANTABLE: NO
*************************** 2. row ***************************
        GRANTEE: role_test
 OBJECT_CATALOG: default_catalog
OBJECT_DATABASE: db_test
    OBJECT_NAME: tbl2
    OBJECT_TYPE: TABLE
 PRIVILEGE_TYPE: SELECT
   IS_GRANTABLE: YES
*************************** 3. row ***************************
        GRANTEE: role_test
 OBJECT_CATALOG: default_catalog
OBJECT_DATABASE: db_test
    OBJECT_NAME: mv_test
    OBJECT_TYPE: MATERIALIZED VIEW
 PRIVILEGE_TYPE: SELECT
   IS_GRANTABLE: YES
*************************** 4. row ***************************
        GRANTEE: role_test
 OBJECT_CATALOG: NULL
OBJECT_DATABASE: NULL
    OBJECT_NAME: NULL
    OBJECT_TYPE: SYSTEM
 PRIVILEGE_TYPE: CREATE RESOURCE GROUP
   IS_GRANTABLE: NO
```

12.5 Grants_to_users

Вы можете просматривать привилегии, предоставленные пользователям, запрашивая представление `grants_to_users`.

> **ПРИМЕЧАНИЕ**
>
> По умолчанию доступ к этому представлению имеют только пользователи или роли с ролью `user_admin`. Вы можете предоставить другим пользователям привилегию SELECT на это представление, используя GRANT.

В `grants_to_users` предоставляются следующие поля:

Поле	Описание
GRANTEE	Пользователь, которому предоставлена эта привилегия.
OBJECT_CATALOG	Catalog, к которому принадлежит объект. Возвращается `NULL`, если привилегия уровня SYSTEM, RESOURCE GROUP, RESOURCE, USER или GLOBAL FUNCTION.
OBJECT_DATABASE	База данных, к которой принадлежит объект. Возвращается `NULL`, если привилегия уровня SYSTEM, RESOURCE GROUP, RESOURCE, USER, GLOBAL FUNCTION или CATALOG.
OBJECT_NAME	Таблица, к которой принадлежит объект. Возвращается `NULL`, если привилегия уровня SYSTEM, RESOURCE GROUP, RESOURCE, USER, GLOBAL FUNCTION, CATALOG или DATABASE.
OBJECT_TYPE	Тип объекта.
PRIVILEGE_TYPE	Тип привилегии. Различные привилегии на один и тот же объект будут объединены и возвращены в одной строке. Как показано в следующем примере, `'user1'@'%'` имеет привилегии SELECT и DROP на `default_catalog.db_test.view_test`.
IS_GRANTABLE	Имеет ли получатель возможность передачи привилегии (grant option).

Пример:

```
MySQL > SELECT * FROM sys.grants_to_users LIMIT 5\G
*************************** 1. row ***************************
        GRANTEE: 'user1'@'%'
 OBJECT_CATALOG: default_catalog
OBJECT_DATABASE: db_test
    OBJECT_NAME: view_test
    OBJECT_TYPE: VIEW
 PRIVILEGE_TYPE: SELECT, DROP
   IS_GRANTABLE: NO
*************************** 2. row ***************************
        GRANTEE: 'user2'@'%'
 OBJECT_CATALOG: default_catalog
OBJECT_DATABASE: simo
    OBJECT_NAME: view_test
    OBJECT_TYPE: VIEW
 PRIVILEGE_TYPE: SELECT
   IS_GRANTABLE: NO
```

12.6 Object_dependencies

Вы можете просматривать зависимости асинхронных материализованных представлений, запрашивая представление `object_dependencies`.

В `object_dependencies` предоставляются следующие поля:

Поле	Описание
object_id	ID объекта.
object_name	Имя объекта.
object_database	База данных, в которой находится объект.
object_catalog	Catalog, в котором находится объект. Это поле всегда содержит `default_catalog`.
object_type	Тип объекта. Это поле всегда содержит `MATERIALIZED_VIEW`.
ref_object_id	ID объекта, на который ссылаются.
ref_object_name	Имя объекта, на который ссылаются.
ref_object_database	База данных, в которой находится объект, на который ссылаются.
ref_object_catalog	Catalog, в котором находится объект, на который ссылаются.
ref_object_type	Тип объекта, на который ссылаются.

Пример:

```
MySQL > SELECT * FROM sys.object_dependencies\G
*************************** 1. row ***************************
          object_id: 11115
        object_name: mv2
    object_database: test_db
     object_catalog: default_catalog
        object_type: MATERIALIZED_VIEW
      ref_object_id: 11092
    ref_object_name: mv1
ref_object_database: test_db
 ref_object_catalog: default_catalog
    ref_object_type: MATERIALIZED_VIEW
*************************** 2. row ***************************
          object_id: 11092
        object_name: mv1
    object_database: test_db
     object_catalog: default_catalog
        object_type: MATERIALIZED_VIEW
      ref_object_id: 11074
    ref_object_name: test_tbl
ref_object_database: test_db
 ref_object_catalog: default_catalog
    ref_object_type: OLAP
```

Приведенный выше пример показывает, что материализованное представление `mv1` создано на основе внутренней таблицы Selena `test_tbl`, а материализованное представление `mv2` создано на основе материализованного представления `mv1`.

12.7 Policy_references

> **ПРИМЕЧАНИЕ**
>
> Это представление относится к функции, недоступной в Selena.

Вы можете просматривать объекты, на которые ссылаются политики доступа к строкам (row access policies) и политики маскирования столбцов (column masking policies), запрашивая представление `policy_references`.

В `policy_references` доступны следующие поля:

Поле	Описание
POLICY_DATABASE	База данных, в которой находится политика. Пространством имён политики является база данных, что допускается только в рамках `default_catalog`.
POLICY_NAME	Имя политики.
POLICY_TYPE	Тип политики. Допустимые значения: `Row Access` и `Column Masking`.
REF_CATALOG	Каталог, в котором на политику есть ссылка.
REF_DATABASE	База данных, в которой на политику есть ссылка.
REF_OBJECT_NAME	Таблица, в которой на политику есть ссылка.
REF_COLUMN	Столбец, в котором на политику есть ссылка. Это поле всегда NULL для политик доступа к строкам.

Пример:

```
mysql > SELECT * FROM policy_references\G
*************************** 1. row ***************************
POLICY_DATABASE: test_db
    POLICY_NAME: region_data
    POLICY_TYPE: Row Access
    REF_CATALOG: default_catalog
   REF_DATABASE: test_db
REF_OBJECT_NAME: test1 -- 表名
     REF_COLUMN: NULL
*************************** 2. row ***************************
POLICY_DATABASE: test_db
    POLICY_NAME: phone_mask
    POLICY_TYPE: Column Masking
    REF_CATALOG: default_catalog
   REF_DATABASE: test_db
REF_OBJECT_NAME: test
     REF_COLUMN: phone
2 rows in set (0.03 sec)
```

12.8 Role_edges

Вы можете просматривать получателей ролей (grantees of roles), запрашивая представление `role_edges`.

> **ПРИМЕЧАНИЕ**
>
> По умолчанию только пользователи или роли с ролью `user_admin` имеют доступ к этому представлению. Вы можете предоставить другим пользователям привилегию SELECT на это представление с помощью GRANT.

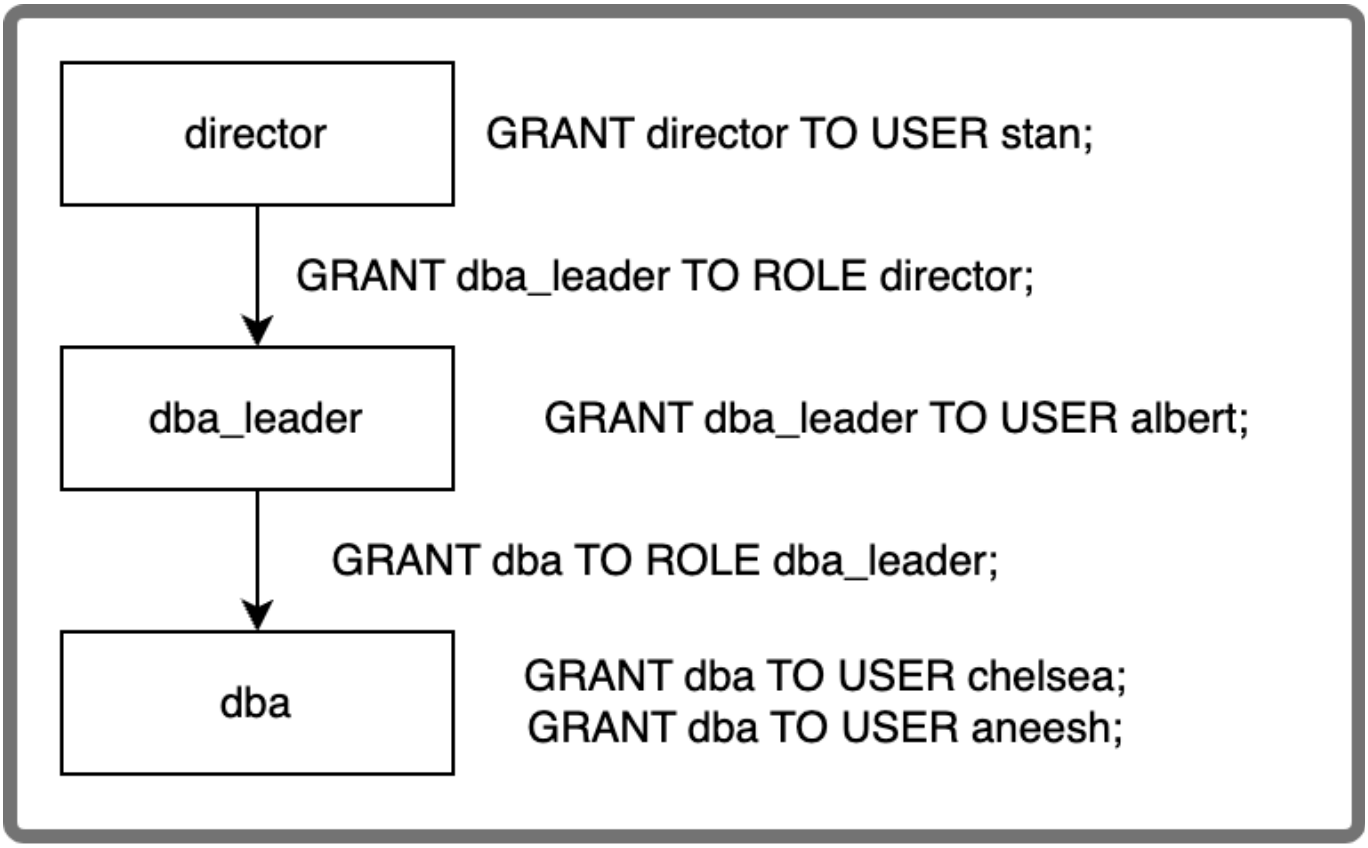
В `role_edges` доступны следующие поля:

Поле	Описание
FROM_ROLE	Роль, которая предоставляется. Может быть возвращено несколько записей с одинаковым `FROM_ROLE`, поскольку роль может быть предоставлена нескольким ролям и пользователям.
TO_ROLE	Роль, которой предоставлена текущая `FROM_ROLE`. Возвращается `NULL`, если текущая `FROM_ROLE` предоставлена пользователю.
TO_USER	Пользователь, которому предоставлена текущая `FROM_ROLE`. Возвращается `NULL`, если текущая `FROM_ROLE` предоставлена роли.

Пример:

```
MySQL > SELECT * FROM sys.role_edges;
+------------+------------+---------------+
| FROM_ROLE  | TO_ROLE    | TO_USER       |
+------------+------------+---------------+
| dba_leader | director   | NULL          |
| dba        | dba_leader | NULL          |
| dba        | NULL       | 'aneesh'@'%'  |
| dba        | NULL       | 'chelsea'@'%' |
| dba_leader | NULL       | 'albert'@'%'  |
| director   | NULL       | 'stan'@'%'    |
| root       | NULL       | 'root'@'%'    |
+------------+------------+---------------+
```

Следующая диаграмма иллюстрирует отношение вложенности предоставленных ролей в предыдущем примере:


director
GRANT director TO USER stan;
GRANT dba_leader TO ROLE director;
dba_leader
GRANT dba_leader TO USER albert;
GRANT dba TO ROLE dba_leader;
dba
GRANT dba TO USER chelsea;
GRANT dba TO USER aneesh;

13 Шаблон для описания параметров FE/BE и переменных

> Когда вы добавляете, изменяете или удаляете параметр FE/BE или переменную в коде, не забудьте обновить документацию: Конфигурация FE, Конфигурация BE, Системные переменные.

Описание параметра или переменной обычно содержит следующие поля:

- Default: *значение по умолчанию для этого параметра. Если значение по умолчанию различается в разных версиях, четко укажите эту информацию.*
- Type: *допустимый тип данных*
- Unit: *Вы можете оставить это поле пустым, если у параметра нет единицы измерения.*
- Value range: *требуется только если у параметра есть диапазон значений*
- Is mutable: *может ли этот параметр быть динамически изменен, требуется только для параметров FE/BE*
- Description: *описание параметра, включая его значение, допустимые значения, зависимости, риски, метод настройки, заявление об устаревании.*
- Introduced in: *с какой версии введен этот параметр, с точностью до 3-значной версии.*

13.1 Описание параметра

Значение

- Для чего используется этот параметр, в каком сценарии он применяется.
- Если этот параметр похож на существующий параметр, в чем их различие?

Зависимости

- Зависит ли этот параметр от других параметров (то есть, этот параметр не может вступить в силу, если другие параметры не установлены или не включены)
- Является ли этот параметр предварительным условием для других параметров
- Является ли этот параметр взаимоисключающим с другими параметрами

Допустимые значения

Если у параметра есть несколько значений, объясните каждое значение.

Риски

- Есть ли какие-либо риски при использовании этого параметра
- Если у параметра есть диапазон значений, что произойдет, если будет превышен верхний/нижний порог
- что произойдет, если параметр установлен слишком большим или слишком маленьким

Метод настройки

Когда возникает ошибка, как настроить этот параметр.

Заявление об устаревании

Если этот параметр удален:

- Объясните причину, по которой он удален
- Заменен ли он другим параметром
- Функции, которые также будут удалены вместе с этим параметром

13.2 Изменение значения по умолчанию/единицы измерения

Если значение по умолчанию или единица измерения параметра изменены, объясните, почему это изменено.

13.3 Изменение имени параметра

- Почему имя параметра изменено
- Будут ли проблемы с совместимостью? Как система решает проблемы совместимости.

13.4 Параметры, невидимые для пользователей

Даже если новый параметр или переменная невидимы для пользователей, мы все равно рекомендуем добавить описание в документацию и **закомментировать его**. Это позволяет скрыть этот параметр от пользователей, но помогает внутренним пользователям (RD и DBA) понять значение и использование этого параметра.

14 Пользовательские переменные

В этой теме описано, как объявлять и использовать пользовательские переменные.

Selena 2.5 и более поздние версии поддерживают пользовательские переменные. Эти переменные используются для хранения определенных значений, на которые ссылаются в последующих SQL-операторах, тем самым упрощая написание SQL-операторов и избегая дублирования вычислений.

14.1 Примечания по использованию

- Пользовательские переменные - это переменные, которые могут быть созданы пользователем и существуют в рамках сессии. Это означает, что никто не может получить доступ к пользовательским переменным, которые были объявлены другим пользователем, и когда сессия закрывается, эти переменные истекают.
- Selena не поддерживает использование оператора SHOW для отображения существующих пользовательских переменных.
- Следующие типы значений не могут быть объявлены как пользовательские переменные: BITMAP, HLL и PERCENTILE. Пользовательские переменные типа JSON преобразуются в тип STRING для хранения.
- Начиная с версии v1.5.2, Selena поддерживает тип ARRAY в качестве пользовательской переменной.

14.2 Объявление пользовательских переменных

Синтаксис

```
SET @var_name = expr [, ...];
```

> **ПРИМЕЧАНИЕ**
>
> - Все переменные должны начинаться с одного знака @ (at sign).
> - Несколько переменных могут быть объявлены в одном операторе SET и должны быть разделены запятыми (`,`).
> - Вы можете объявить одну и ту же переменную несколько раз. Вновь объявленное значение перезаписывает исходное значение.

> - Если используется необъявленная переменная, значение переменной по умолчанию равно `NULL`, а тип NULL - STRING.

Параметры

Параметр	Обязательный	Описание
var_name	Да	Имя пользовательской переменной. Правила именования следующие: • Имя может содержать буквы, цифры (0-9) и подчеркивания (_). • Имя не может превышать 64 символа в длину. Вы можете объявить строку как пользовательскую переменную, например @'my-var', @"my-var" и @`my-var`. Пользовательские переменные типа STRING могут содержать символы, отличные от букв, цифр и подчеркиваний (_), такие как точки (.).
expr	Да	Значение пользовательской переменной. Вы можете указать число (например, 43) или сложное выражение (например, значение, возвращаемое оператором SELECT) для этого параметра. Тип данных переменной совпадает с типом данных результата, возвращаемого выражением.

Примеры

Пример 1: Объявление числа как пользовательской переменной.

```
SET @var = 43;
```

Пример 2: Объявление значения, возвращаемого SELECT-запросом, как пользовательской переменной.

```
SET @var = (SELECT SUM(v1) FROM test);
```

Пример 3: Объявление нескольких пользовательских переменных в одном операторе SET.

```
SET @v1=1, @v2=2;
```

14.3 Использование пользовательских переменных в SQL

- Упрощает написание SQL-операторов. Например, при выполнении следующего оператора SELECT, Selena интерпретирует `@var` как `1`.

```
SET @var = 1;
SELECT @var, v1 from test;
```

- Позволяет избежать дублирования вычислений. Например, при выполнении следующего оператора SELECT, Selena интерпретирует `@var` как результат, возвращаемый командой `select sum(c1) from tbl`.

```
SET @var = (select sum(c1) from tbl);
SELECT @var, v1 from test;
```